



Supervised and transfer learning for industrial equipment remaining useful life prediction under limited data

T Maré



[orcid.org/ 0009-0000-0807-124X](https://orcid.org/0009-0000-0807-124X)

Dissertation accepted in fulfilment of the requirements for the degree
Master of Engineering in Industrial Engineering
at the North-West University

Supervisor: Prof JH Bührmann

Graduation: September 2026

Acknowledgements

I would firstly like to thank my supervisor, Prof Joke H. Bührmann, for introducing me to the world of Predictive Maintenance. Her guidance and valuable feedback ensured the continual progression and improvement of this project.

Next, I would like to express my gratitude and love towards both my parents. Without their patience, support and positivity, the completion of this project would not have been possible.

Most importantly, I would like to thank God for being the cornerstone of my life and for providing me with the peace and confidence to enable me to face every challenge that was thrown my way.

Abstract

Remaining useful life (RUL) prediction is a key component of predictive maintenance (PdM) systems, where data-driven models are used to identify degradation patterns and predict impending equipment failure. Despite the advances of Industry 4.0 and the increased availability of sensor data, limited labelled degradation data remains a challenge in PdM. However, limited research has systematically assessed how different machine learning models perform as labelled target data become increasingly scarce. Moreover, when practitioners have access to only a limited amount of labelled target data, related source-domain datasets can be used. However, the domain shift between source and target datasets can affect model performance. This study therefore evaluates classical regression, tree-based, and neural-network-based models under varying data-availability constraints. Feature selection and hybrid feature extraction architectures are also investigated to determine their effect on model performance (across the varying data availability constraints). In addition, three supervised fine-tuning strategies, namely full fine-tuning, layer freezing, and progressive unfreezing, are evaluated for scenarios where source-domain data are available. For the case where no labelled target data are available, adversarial domain adaptation is implemented using a domain adversarial neural network (DANN). The experiments are conducted using the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) datasets, with FD001 used as the target dataset and FD002 and FD003 used as source-domain datasets.

The results show that, within the investigated C-MAPSS datasets and experimental design, no single modelling strategy performed best across all data-availability scenarios. Neural-network-based models generally performed best when sufficient labelled target data were available. Among the evaluated neural-network-based models, the Transformer showed the greatest robustness as target training data became scarce. Under extreme training-data scarcity, the evaluated linear and tree-based models became increasingly competitive and outperformed the neural-network-based models. Furthermore, feature selection was found to mainly improve computational efficiency, particularly reducing the recurrent models' training time by approximately 87% to 97%, although its effect on predictive accuracy varied across the evaluated models. The proposed AUC-RMSE metric further showed that conclusions based solely on aggregate RMSE can differ from those ob-

tained when prediction performance is evaluated across different stages of the equipment lifecycle.

For transfer learning, the supervised fine-tuning techniques were most advantageous in the lower-data scenarios, while negative transfer was observed under some higher-data conditions. The greatest benefits were observed for the more similar of the investigated source-target domains, where all three fine-tuning strategies achieved RMSE improvements of more than 50% over the supervised target-only baseline in the data-scarce scenarios. When labelled target data were unavailable, source-only transfer remained competitive in the smaller domain-shift setting, whereas DANN provided greater benefit in the larger of the investigated domain-shift settings. In this setting, DANN improved RMSE over the source-only baseline by more than 27% across all evaluated data ranges, with the highest improvement of 38% observed under extreme scarcity. Under very low and extreme labelled target-data settings, source-only and DANN-based transfer models also became competitive with, and in some cases outperformed, the supervised GRU baseline. These findings demonstrate predictive improvements within the investigated C-MAPSS benchmark scenarios. Future work should evaluate the investigated models on additional datasets to assess the generalisability of the findings. Moreover, the evaluation on real-world industrial degradation data could further assess their practical applicability.

Keywords: Remaining useful life prediction; Predictive Maintenance; Machine Learning; Transfer Learning; Domain Adaptation; Data Scarcity;

Contents

1	Introduction	1
1.1	Background	1
1.1.1	Problem Background	6
1.1.2	Problem Statement	7
1.1.3	Aim	7
1.1.4	Objectives	7
1.1.5	Research Questions	8
1.2	Structure of Dissertation	9
2	Theoretical Background	10
2.1	RUL Prediction Algorithms	10
2.1.1	Regression Models	11
2.1.2	Tree-based Models	14
2.1.3	Artificial Neural-Networks	19
2.2	Feature Extraction	26
2.2.1	Convolutional neural-network	27
2.2.2	Temporal Convolutional Network	28
2.2.3	Autoencoders	31
2.3	Hyperparameter Tuning	32

2.3.1	Bayesian Optimisation Algorithm	34
2.4	Transfer Learning	35
2.4.1	Maximum Mean Discrepancy	36
2.4.2	fine-tuning	37
2.4.3	Domain Adversarial neural-network	39
2.5	Common Limitations and Challenges in PdM	40
2.5.1	Data Availability and Labelling Challenges	40
2.5.2	Industrial Equipment Complexity and Variability	40
2.5.3	Uncertainty and Prediction Reliability	41
2.6	Synthesis of Literature	41
2.6.1	Data Used for RUL Prediction	42
2.6.2	Predictive Modelling Approaches for RUL Prediction	42
2.6.3	Feature Extraction	43
2.6.4	Optimisation and Hyperparameter Tuning	43
2.6.5	Performance Evaluation	43
2.6.6	Future Research Directions	44
3	Data Set Selection and Analysis	45
3.1	Dataset Selection Criteria and Motivation	45
3.1.1	Dataset Identification and Motivation	46
3.2	Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)	47
3.3	Data Preparation and Exploratory Data Analysis.	50
3.3.1	Data Preprocessing	55
4	Overview of the Methodology	58
4.1	Data Preparation	60

4.2	Baseline Architecture and Hyperparameter Tuning	62
4.3	Feature Selection	63
4.4	Feature Extraction	63
4.5	Transfer Learning	64
4.5.1	Supervised Transfer Learning	64
4.5.2	Unsupervised Domain Adaptation	64
4.5.3	Model Validation and Hyperparameter Tuning	65
4.5.4	Evaluation Metrics	67
5	Implementation and Validation	72
5.1	Baseline Degradation Modelling	73
5.1.1	Baseline Model Implementation	73
5.1.2	Hyperparameter Tuning Implementation	75
5.1.3	Grid Search	76
5.1.4	Optuna	78
5.2	Feature Selection	85
5.2.1	Correlation-based Feature Selection	86
5.2.2	Tree-based feature selection	86
5.2.3	Feature Selection Validation	88
5.3	Feature Extraction	89
5.3.1	CNN-GRU	90
5.3.2	TCN-GRU	91
5.3.3	AE-GRU	93
5.3.4	Feature Extraction Validation	94
5.4	Fine-tuning Transfer Learning Implementation	97

5.4.1	Maximum Mean Discrepancy Analysis	98
5.4.2	Fine-tuning Implementation	99
5.5	Source-Only Transfer and Adversarial Domain Adaptation Implementation	104
5.5.1	Input Mini-Batch Preparation	105
5.5.2	DANN Model Architecture	106
5.5.3	DANN Training Implementation	107
5.5.4	Validation of the DANN model	109
6	Results and Discussion	114
6.1	Baseline Model Performance Under Data Scarcity	115
6.1.1	AUC-RMSE Analysis	121
6.2	Feature Engineering Impact Under Data Scarcity	124
6.2.1	Feature Selection on Baseline Models Results	124
6.2.2	AUC-RMSE Analysis for FS	128
6.2.3	Hybrid neural-network Feature Extraction Results	130
6.3	Comparison with Published FD001 Results	134
6.4	Supervised Transfer Learning Results	136
6.4.1	AUC-RMSE Analysis of Supervised TL strategies	140
6.5	Unsupervised Adaptation Results	142
6.5.1	AUC-RMSE Analysis of Domain Adaptation TL Strategies	147
6.6	Discussion	158
6.6.1	Model Robustness Under Data Scarcity	158
6.6.2	Effect of Feature Engineering Under Low-Data Conditions	159
6.6.3	Comparison Between AUC-RMSE and Traditional Metrics	159
6.6.4	Effectiveness of Supervised Transfer Learning	160

6.6.5	Source-Only Transfer and Unsupervised Domain Adaptation	161
6.6.6	Sensitivity Analysis	162
6.6.7	Limitations and Future Work	164
6.6.8	Synthesis of Findings	166
7	Conclusion	167
7.1	Achievement of Research Objectives	167
7.2	Key Findings and Contributions	168
7.2.1	Practical Strategy for RUL Model Selection	169
A	SLR of techniques used for RUL Prediction	192
A.1	Introduction	192
A.2	Overview of the SLR	193
A.3	Systematic Literature Review Methodology	194
A.3.1	Records identified in Databases	195
A.3.2	Screening the Identified Publication	196
A.3.3	Selected Publications	197
A.3.4	Geographical distribution and prominence of the selected works . . .	197
A.4	Limitations and Challenges Commonly Found in Predictive Data Maintenance (PdM)	200
A.5	Types of data used	200
A.6	Feature extraction techniques	203
A.6.1	Convolutional neural networks	204
A.6.2	Temporal convolutional networks	205
A.6.3	Autoencoders	206
A.6.4	Traditional Statistics	207

A.6.5	Transformers	207
A.7	Prediction models	208
A.7.1	LSTM for RUL prediction	209
A.7.2	Dense feed-forward neural network for remaining useful life prediction	211
A.7.3	GPR for remaining useful life prediction	212
A.7.4	ESN for remaining useful life prediction	213
A.7.5	GRU for remaining useful life prediction	214
A.7.6	FE-LSTM and FE-Dense ANN Architectures	214
A.8	Optimisers	215
A.8.1	Adam optimiser	215
A.8.2	Stochastic gradient descent	216
A.9	Hyperparameter tuning techniques	217
A.9.1	Grid Search	217
A.9.2	Genetic Algorithm	218
A.9.3	Bayesian optimisation algorithm	219
A.9.4	Random Search	219
A.9.5	Manual Tuning	219
A.10	Performance Evaluation Metrics	220
A.10.1	Accuracy/ Error metrics	222
A.10.2	Computational Efficiency	223
A.10.3	Reliability Metrics	224
A.10.4	Fit Metrics	224
A.10.5	Generalisation Metrics	224
A.10.6	Interpretability and Operational Effectiveness Metrics	225
A.11	Future recommendations of studies	226

A.11.1 Subdomain Adaptation	226
A.11.2 Application to New Domains	227
A.11.3 Hyperparameter tuning	227
A.11.4 Model Uncertainty	228
A.11.5 Practical Application	228
A.11.6 Additional Future Works	229
A.12 Discussion	229
A.13 Conclusion	231
B Exploratory Data Analysis	234
B.1 Evolution of the RUL (Cycles) of all CMAPSS FD001 sensors	234
C Implementation and Validation	237
C.1 Hyperparameter search spaces used for the baseline models	237
C.2 Classical Regression Baseline Selected Hyperparameters (Grid Search) . . .	239
C.3 Selected Baseline Hyperparameters from Optuna	239
C.4 Hybrid FE Hyperparameter Search Spaces	241
C.5 Selected Hybrid Feature Extraction Hyperparameters	242
D DANN Implementation	243
D.1 DANN Training Curves.	243
D.2 DANN Batch Generator Implementation	246
D.3 Build DANN Model	250
D.4 The Gradient Reversal Layer	253

List of Figures

1.1	Key characteristics of the CMMS, adapted from Rachidi et al. (2013). . . .	2
1.2	Equipment remaining useful life visualisation, adapted from Wang et al. (2020)	4
1.3	Evolution of the fundamental industrial equipment maintenance strategies, adapted from Nunes et al. (2023)	5
2.1	Illustrative structure of a regression tree for RUL prediction, where sensor features are successively compared with learned split thresholds until a terminal RUL prediction is reached.	15
2.2	Visual representation of RF Regression, adapted from Lemeš and Akagic (2023).	16
2.3	High-level comparison of bagging and boosting, showing independent learner construction in bagging and sequential learner dependence in boosting, adapted from Yang et al. (2019).	17
2.4	Level-wise vs Leaf-Wise Tree Growth, adapted from Bui et al. (2021). . . .	18
2.5	Simplified structures of feed-forward and RNN, adapted from Ma et al. (2019).	20
2.6	Recurrent sequence structure of an LSTM network, adapted from Fu et al. (2019).	21
2.7	Internal structure and gating mechanisms of an LSTM unit (Fu et al., 2019).	22
2.8	General GRU Structure, adapted from Riaz et al. (2020).	23
2.9	General Transformer Structure, adapted from Vaswani et al. (2017)	25
2.10	The general structure of convolution and causal convolution in TCNs, adapted from Hsu et al. (2022)	29

2.11 Dilated-causal convolution visualisation in TCNs, adapted from Duc et al. (2020)	30
2.12 General structure of an AE, adapted from Lu et al. (2021)	31
2.13 Architecture of DANN, adapted from Da Costa et al. (2020)	39
3.1 Simplified cross-sectional view of the internal structure and components of a typical commercial aircraft engine, adapted from Wickert and Lewis (2017).	47
3.2 Preprocessing Steps for the C-MAPSS dataset.	51
3.3 Maximum operational cycles per engine unit in the FD001 training dataset.	52
3.4 Evolution of four representative sensors (s_1, s_2, s_7, s_{14}) over RUL, illustrating distinct degradation patterns.	53
3.5 Correlation matrix of C-MAPSS sensors and RUL for the FD001 training dataset.	54
3.6 Distribution of the C-MAPSS FD001 Representative Sensors.	55
4.1 Overview of the experimental design.	59
4.2 Conceptual illustration of AUC-RMSE	69
5.1 Hyperparameter Optimisation Curve RMSE for A)Lasso, B)Ridge and C) ElasticNet	77
5.2 Total hyperparameter tuning time per algorithm using Optuna	83
5.3 Best-so-far validation RMSE versus Optuna trial for tree-based and neural-network models	84
5.4 Ranked Random Forest feature importance scores for the FD001 training dataset	87
5.5 Training time comparison for selected models at 80 engines using the full 24-feature baseline, the 14-feature correlation-based subset, and the 12-feature Random Forest importance subset.	89
5.6 CNN-GRU architecture, training and hyperparameter tuning flow.	90
5.7 TCN-GRU FE architecture.	92

5.8	AE-GRU architecture, training and hyperparameter tuning flow.	93
5.9	AE-GRU correlation of 14 originally selected features compared with the correlation of the latent features generated by the AE.	95
5.10	Training durations of the selected models (80 engines) for GRU CNN-GRU, TCN-GRU, and AE-GRU.	96
5.11	Optuna hyperparameter optimisation history for the CNN-GRU, TCN-GRU, and AE-GRU models.	97
5.12	Overview of the supervised transfer-learning workflow used for the GRU model. In Step 1, the GRU model is pretrained on a source dataset (FD002 or FD003). In Step 2, one of three fine-tuning strategies is configured: (A) full fine-tuning, where all pretrained GRU layers remain trainable; (B) layer freezing, where the pretrained GRU layers are frozen and only the output dense layer remains trainable; or (C) gradual unfreezing, where training begins with the pretrained GRU layers frozen and these layers are progressively unfrozen during fine-tuning. In Step 3, the configured model is fine-tuned using labelled target-domain data from FD001. Here, lr denotes the learning rate, while “Frozen” and “Trainable” indicate whether the parameters of the corresponding layer are updated during fine-tuning.	100
5.13	Training times of the fine-tuning strategies VS the GRU baseline.	101
5.14	The Validation RMSE results for TL1, TL2 and TL3 and GRU baselines trained independently on engine counts of 80, 40, 20 10 and 5 scenarios . . .	102
5.15	Implementation of DANN.	105
5.16	Training procedure implementation of DANN.	108
5.17	The training times of LSTM-DANN, GRU-DANN as well as the source-only GRU baseline.	110
5.18	Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 80 unlabelled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).	111
5.19	Validation RMSE performance across varying target engine counts for domain adaptation from FD002 and FD003 to FD001, comparing supervised learning, source-only transfer, and DANN variants.	112

6.1	Relative RMSE degradation across data availability scenarios (including 40 to 20 engines; 20 to 10 engines; 10 to 5 engines and less than or equal to 5 engines) compared with the high-data baseline range of 80 to 40 engines. . .	120
6.2	Number of test windowed observations for true RUL Label.	121
6.3	AUC-RMSE lifecycle curves (solid) and standard RMSE (stippled) for GRU, BiLSTM, Transformer, and MLR across (A) high-data, (B) moderate-data, (C) low-data, and (D) extreme-data scarcity.	122
6.4	AUC-RMSE from very low to extreme data scarcity (10 to 1 training engines scenarios) for GRU, BiLSTM, Transformer and MLR.	123
6.5	AUC-RMSE lifecycle curves (solid) and standard RMSE (stippled) for CNN-GRU, TCN-GRU and AE-GRU as well as the GRU and Lasso models trained on correlated selected features across (A) high-data, (B) moderate-data, (C) low-data, and (D) extreme-data scarcity.	133
6.6	Improvement of supervised TL models over the GRU baseline, where TL1, TL2 and TL3 refer to full fine-tuning, layer freezing and progressive unfreezing strategies, respectively.	139
6.7	Test RMSE across target-engine-count scenarios for supervised TL strategies using FD002 and FD003 as source domains, compared with the GRU baseline	140
6.8	Test AUC-RMSE across target-engine-count scenarios for supervised TL strategies using FD002 and FD003 as source domains, compared with the GRU baseline.	141
6.9	AUC-RMSE lifecycle curves (solid) and standard RMSE (stippled) for TL1, TL2 and TL3, as well as the GRU and Lasso models, across four data-availability scenarios using correlation-selected features as input: (A) 80 engines, (B) 40 engines, (C) 10 engines and (D) 5 engines.	142
6.10	Improvement of unsupervised TL, including GRU-DANN and LSTM-DANN models, over the source-only GRU	146
6.11	Test RMSE results for all independent engine-count scenarios for GRU-DANN, LSTM-DANN and the source-only GRU, as well as the supervised GRU baseline.	147
6.12	The test AUC-RMSE results for all independent engine count scenarios for GRU-DANN, LSTM-DANN and source-only GRU as well as the supervised GRU baseline.	148

- 6.13 AUC-RMSE lifecycle curves (solid lines) and standard RMSE values (stippled lines) for GRU-DANN, LSTM-DANN and source-only GRU trained on **FD002**, compared with the supervised GRU baseline trained on labelled FD001 target-domain-data. Results are shown for four target-data availability scenarios using the correlation-based selected feature subset: (A) 80 engines, (B) 40 engines, (C) 10 engines and (D) 5 engines. 149
- 6.14 Predicted versus actual RUL on the FD001 test set for the **GRU-DANN** model using **FD002** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL. 150
- 6.15 Predicted versus actual RUL on the FD001 test set for the **LSTM-DANN** model using **FD002** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL. 151
- 6.16 Predicted versus actual RUL on the FD001 test set for the **source-only GRU** model using **FD002** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL. 152
- 6.17 AUC-RMSE lifecycle curves (solid lines) and standard RMSE values (stippled lines) for GRU-DANN, LSTM-DANN and source-only GRU trained on **FD003**, compared with the supervised GRU baseline trained on labelled FD001 target-domain-data. Results are shown for four target-data availability scenarios using the correlation-based selected feature subset: (A) 80 engines, (B) 40 engines, (C) 10 engines and (D) 5 engines. 153
- 6.18 Predicted versus actual RUL on the FD001 test set for the **GRU-DANN** model using **FD003** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL. 154
- 6.19 Predicted versus actual RUL on the FD001 test set for the **LSTM-DANN** model using **FD003** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL. 155
- 6.20 Predicted versus actual RUL on the FD001 test set for the **source-only GRU** model using **FD003** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL. 156

6.21 Predicted versus actual RUL on the FD001 test set for the Supervised-GRU model using labelled FD001 training data. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.	157
7.1 Practical high-level strategy for RUL model selection based on data availability, source-target domain shift, and application-specific evaluation requirements.	170
A.1 The typical machine-learning workflow used to structure the SLR, adapted from, Alzubi et al. (2018).	193
A.2 PRISMA 2020 flow diagram of methodology for the systematic literature review, adapted from Page et al. (2021).	194
A.3 Filtering process of identified publications	197
A.4 Number of publications per year and corresponding references	197
A.5 The choropleth map that represents the geographical distribution of the identified articles	198
A.6 Pie chart showing the proportion of citations across selected studies	199
A.7 Proportion of studies using synthetic vs real-world data	201
A.8 Distribution of industrial equipment sources represented in the reviewed studies	202
A.9 Distribution of algorithms/techniques found in the literature for feature extraction	204
A.10 Distribution of algorithms/techniques found in the literature for RUL prediction	208
A.11 Distribution of different feature extraction (FE) and LSTM (A) and Dense ANN (B) prediction model architecture combinations.	214
A.12 Distribution of optimiser algorithms found in the literature.	215
A.13 of algorithms/techniques found in the literature for hyperparameter optimisation.	217
A.14 Proportion of studies that used classification metrics or regression metrics.	221

A.15	Distribution of evaluation metrics used for RUL prediction models.	222
A.16	Distribution of the proposed future works.	226
B.1	Evolution of the RUL (Cycles) of all CMAPSS FD001 sensors.	235
B.2	Distributions of sensor features in C-MAPSS FD001.	236
D.1	Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 80 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).	243
D.2	Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 40 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).	244
D.3	Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 10 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).	245
D.4	Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 5 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).	246

List of Tables

2.1	Main steps of the genetic algorithm for hyperparameter optimisation, adapted from Beg and Islam (2016).	34
3.1	C-MAPSS inputs used to simulate degradation scenarios across rotating engine components, adapted from Saxena et al. (2008b).	48
3.2	C-MAPSS sensor parameters provided to participants.	49
3.3	Overview of C-MAPSS sub-datasets and experimental setup.	49
3.4	Operational settings in the C-MAPSS dataset.	50
4.1	Number of observations available at each engine-count condition.	61
5.1	Summary of baseline models, libraries and the main objects used.	74
5.2	The number of explored hyperparameters per model.	75
5.3	Selected hyperparameter configurations for the baseline models.	80
5.4	Baseline validation RMSE results at 80 engines with 14 241 total observations.	85
5.5	Validation RMSE at 80 engines across feature selection methods.	88
5.6	Optuna-selected configurations for hybrid GRU models.	95
5.7	Pairwise Maximum Mean Discrepancy (MMD ²) values between C-MAPSS domains using an RBF kernel.	99
5.8	Validation RMSE of GRU fine-tuning strategies using the full FD001 training set.	101

5.9	DANN hyperparameter configurations for the FD002 to FD001 and FD003 to FD001 transfer settings based on Da Costa et al. (2020).	109
6.1	Baseline model test performance averaged across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the model rankings according to RMSE and AUC-RMSE, respectively, while Diff indicates the change in ranking when evaluated using AUC-RMSE instead of RMSE (negative values indicate an improved rank, positive values a decreased rank, and zero no change). Time(s) denotes model training time in seconds.	115
6.2	Percentage change in test performance after applying correlation- and RF-based feature selection relative to the corresponding full-feature baseline, averaged across five training-data scarcity ranges. For RMSE, AUC-RMSE, MAE, C-MAPSS, and training time, negative values indicate an improvement, while for R^2 , positive values indicate an improvement.	125
6.3	Comparison of RMSE and AUC-RMSE rankings for feature-selected models across five training-data scarcity ranges. R and A denote the RMSE and AUC-RMSE ranks, respectively, while D indicates the change in rank when evaluated using AUC-RMSE instead of RMSE. Negative values (green) indicate an improved rank under AUC-RMSE, positive values (red) indicate a decreased rank, and zero values (yellow) indicate no change. Corr. and RF denote correlation-based and Random Forest-based feature selection, respectively.	129
6.4	Test performance of the CNN-GRU, TCN-GRU, and AE-GRU hybrid feature-extraction architectures compared with the GRU baseline across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the rankings according to RMSE and AUC-RMSE, respectively, and Time(s) denotes model training time in seconds. .	130
6.5	Contextual comparison of selected RUL-prediction architectures evaluated on the C-MAPSS FD001 dataset.	135

6.6	Test performance of the supervised transfer-learning strategies TL1, TL2, and TL3 using FD002 and FD003 as source domains, compared with the supervised GRU baseline trained directly on FD001, across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the rankings according to RMSE and AUC-RMSE, respectively, and Time(s) denotes model training time in seconds.	137
6.7	Test performance of the source-only and DANN domain-adaptation models using FD002 and FD003 as source domains, compared with the supervised GRU baseline trained directly on FD001, across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the rankings according to RMSE and AUC-RMSE, respectively, and Time(s) denotes model training time in seconds.	143
6.8	Contextual comparison of LSTM-DANN test RMSE with the results reported by Da Costa et al. Da Costa et al. (2020).	162
6.9	Sensitivity analysis of test RMSE across selected training-engine counts, reported as mean $\pm$ standard deviation across five different seeded runs.	163
A.1	List of the databases, search queries and number of results identified.	195
A.2	Citation distribution of the selected works.	198
A.3	Non-proprietary datasets	202
C.1	Hyperparameter search spaces used for the baseline models.	237
C.2	Selected best hyperparameters for the classical regression baseline models.	239
C.3	Selected best hyperparameters for the remaining baseline models.	240
C.4	Hyperparameter search spaces used for the hybrid feature extraction models.	241
C.5	Selected best hyperparameters for the hybrid feature extraction models.	242

Table of abbreviations

AE	Autoencoder
AE-GRU	Autoencoder-Gated Recurrent Unit
ANN	Artificial Neural Network
AUC-RMSE	Area Under the Curve Root Mean Square Error
BiLSTM	Bidirectional Long Short-Term Memory
BOA	Bayesian Optimisation Algorithm
CBM	Condition-Based Maintenance
C-MAPSS	Commercial Modular Aero-Propulsion System Simulation
CM	Corrective Maintenance
CMMS	Computerised Maintenance Management Systems
CNN	Convolutional Neural Network
CNN-GRU	Convolutional Neural Network-Gated Recurrent Unit
CUDA	Compute Unified Device Architecture
DANN	Domain-Adversarial Neural Network
EDA	Exploratory Data Analysis
EOL	End-of-Life
FE	Feature Extraction
FS	Feature Selection
GA	Genetic Algorithm
GPU	Graphics Processing Unit
GPR	Gaussian Process Regression
GRL	Gradient Reversal Layer
GRU	Gated Recurrent Unit
GRU-DANN	Gated Recurrent Unit Domain-Adversarial Neural Network
GS	Grid Search
HPC	High-Pressure Compressor
HPT	High-Pressure Turbine

IoT	Internet of Things
LGBM	Light Gradient Boosting Machine
LPC	Low-Pressure Compressor
LPT	Low-Pressure Turbine
LSTM	Long Short-Term Memory
LSTM-DANN	Long Short-Term Memory Domain-Adversarial Neural Network
MAE	Mean Absolute Error
ML	Machine Learning
MLR	Multiple Linear Regression
MMD	Maximum Mean Discrepancy
MSE	Mean Squared Error
NASA	National Aeronautics and Space Administration
PCA	Principal Component Analysis
PCoE	Prognostics Center of Excellence
PdM	Predictive Maintenance
RAM	Random Access Memory
RBF	Radial Basis Function
RBM	Risk-Based Maintenance
RCM	Reliability-Centred Maintenance
RF	Random Forest
RKHS	Reproducing Kernel Hilbert Space
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RUL	Remaining Useful Life
SD	Standard Deviation
SGD	Stochastic Gradient Descent
SLR	Systematic Literature Review
TCN	Temporal Convolutional Network
TCN-GRU	Temporal Convolutional Network-Gated Recurrent Unit
TL	Transfer Learning
TPM	Total Productive Maintenance
TPE	Tree-Structured Parzen Estimator
TRA	Throttle Resolver Angle
XGB	Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Background

With advances in science and technology, the maintenance of modern complex industrial equipment becomes a growing challenge. Under the combined impact of internal and external factors, the condition of these complex systems inevitably tends to deteriorate (Pei, 2019). Subsequently, the requirement to closely monitor the health of these industrial equipment plays a vital role in maintenance. Equipment failure can lead to unplanned downtime and costly replacement, leading to reduced operational efficiency and profitability (Ojeda et al., 2025).

Maintenance of equipment has been prevalent for thousands of years. According to Poór (2019), there are records of maintenance documents that date back to 600 BC in ancient Egypt. Furthermore, before the industrial revolution maintenance was mainly approached reactively (Poór, 2019). This approach, referred to as “corrective maintenance” (CM), involves the repair of the equipment only after significant faults or breakdowns have occurred. The goal of this maintenance strategy is to minimise the cost of maintaining the equipment while maximising the time between maintenance actions (Mołęda et al., 2023). However, corrective maintenance can cause equipment damage and unavailability leading to unexpected costs (Mołęda et al., 2023).

Preventive maintenance addresses this challenge directly by avoiding failures through periodic maintenance schedules with the aim of avoiding unexpected equipment failures (Carvalho et al., 2019). However, this strategy does not fully prevent unexpected failures from occurring, as scheduled maintenance is a preset plan that may not be able to take care of faults that occur earlier than expected (Mołęda et al., 2023). Furthermore, Carvalho et al. (2019) note that this approach often results in unnecessary expenses due to maintenance actions that may be deemed unnecessary.

As the complexity and scale of industrial systems grew, so did the demand for more advanced strategies. This led to the creation of frameworks such as computerised maintenance management systems (CMMS), Total Productive Maintenance (TPM), reliability-centred maintenance (RCM), and Risk-Based Maintenance (RBM).

The growing role of information as a prerequisite for effective maintenance decision-making popularised the adoption of CMMS that emerged in the 1960s (Kans, 2009). These information systems are made up of functions that capture and process maintenance data to generate information and indicators that support maintenance activities and strategy (Lopes et al., 2016). Furthermore, CMMS reduces manual paper trail and tracking, centralises maintenance data and frees maintenance personnel, and therefore subsequently shifts the focus to execution and continuous improvement (Shankar et al., 2023).

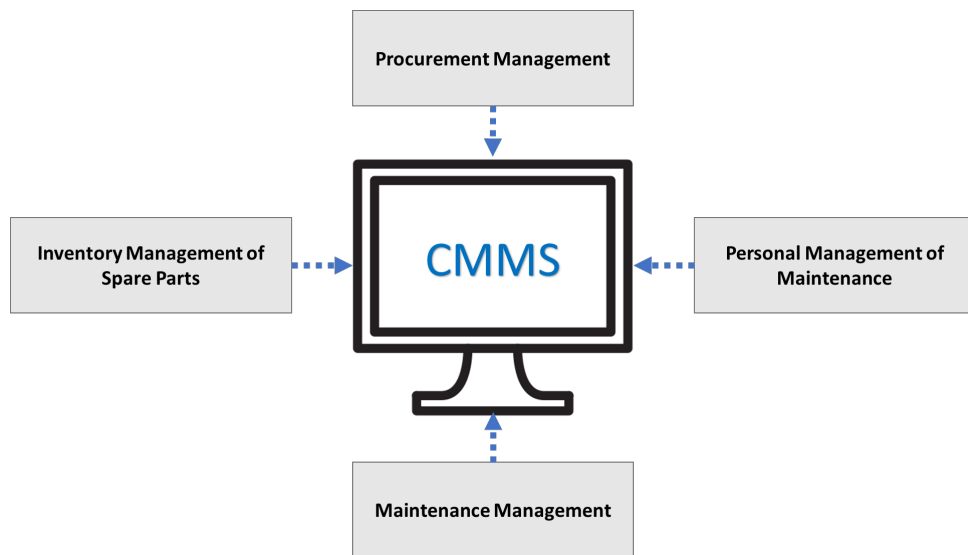


Figure 1.1: Key characteristics of the CMMS, adapted from Rachidi et al. (2013).

CMMS integrates four key components that help plan and take care of maintenance. This includes maintenance management that involves preventive interventions in industrial systems. Furthermore, it involves maintenance personnel management, which includes planning and assignments as well as training management. In addition, spare parts inventory management includes reserve stock control, threshold alerts, and parts reception. Then the procurement management component focuses on the coverage of purchase orders, the management of suppliers and pricing, and billing (Rachidi et al., 2013).

Although CMMS improves maintenance data management, it is still a challenge to transform maintenance plans into actual execution, as well as to ensure that continuous improvement principles are applied. This requirement leads to maintenance methodologies such as Total Productive Maintenance (TPM). TPM is a Japanese maintenance philosophy that became prevalent in the 1970s. It focuses on the effectiveness of the equipment, the

elimination of malfunctions, and the autonomous maintenance of the operators through the participation of the entire workforce (Ahuja and Khamba, 2008).

TPM is implemented as a multi-departmental approach that builds a collaborative relationship between production and maintenance for continuous improvement. It does this by combining the 'total maintenance system' with 'total participation' through autonomous maintenance and small-group activities that involve all employees (Chan et al., 2005).

To operationalise these maintenance practices, appropriate maintenance tasks must be selected in defined intervals and prioritisation between various industrial equipment and failure modes (Rouzbeh et al., 2016). This motivates frameworks that formalise maintenance decision-making. Reliability-centred maintenance (RCM) is an analytical process to determine the maintenance requirements of an asset within its operating context (published in 1978 as an aircraft maintenance-based approach) (Nowlan and Heap, 1978). Gomaa (2025) further describes RCM as a structured methodology in which failure modes are identified, their consequences are analysed, and criticality analysis (which involves ranking unwanted incidents by importance) is used to prioritise failure modes and select maintenance actions.

In the mid 1980s, Risk-Based Maintenance (RBM) became a formalised approach to make maintenance decisions (Da Silva et al., 2023). RBM involves quantifying values that represent risk and then, based on prioritisation, perform inspection and maintenance actions. Furthermore, it provides recommendations on the timing and maintenance type of preventive tasks to reduce the probability that unexpected failures occur (Khan and Haddara, 2003).

Schedule based maintenance strategies do have limitations as fixed schedules may fail to capture occurring faults or led to unnecessary maintenance actions. Condition-based maintenance (CBM) involves preventive maintenance actions based on monitored condition data. In addition, it accounts for the past and current health status of an asset while also forecasting its future health state, and subsequently reduces the risk of failure and unnecessary maintenance costs (Zhang et al., 2025). CBM systems often involve techniques such as anomaly/fault detection and diagnostic reasoning, as well as prognostic health status functions (Thakkar and Chaoui, 2022).

Traditionally, CBM approaches involve physics-based modelling using first-principle physics (Daigle and Goebel, 2021). However, this restricts practical implementation by necessitating extensive domain expertise and significant computational resources (Sharma, 2025). Data-driven approaches are often the preferred and more practical approach. This is largely due to advances in computing hardware, together with rapid developments in machine learning (ML) and deep learning, which have made it possible to design and deploy such algorithms in a wide range of domains (Thakkar and Chaoui, 2022).

Predictive Maintenance (PdM) has emerged as a widely used proactive maintenance technique (Carvalho et al., 2019; Molęda et al., 2023). The approach utilises data-driven ML models to infer equipment health and anticipate failures, so maintenance is performed only when necessary (Zonta et al., 2020). PdM monitors indicators of degradation and schedules proactive interventions before failure (Zhang et al., 2019), often requiring lead times of weeks to support investigation and maintenance planning (Toothman et al., 2023). In practice, PdM decisions are often driven by estimates of the monitored assets' Remaining Useful Life (RUL). RUL quantifies the time (or cycles) remaining before an asset reaches an unacceptable performance threshold, commonly referred to as its End-Of-Life (EOL) (Kang et al., 2021). Figure 1.2 visualises the degradation of RUL over time.

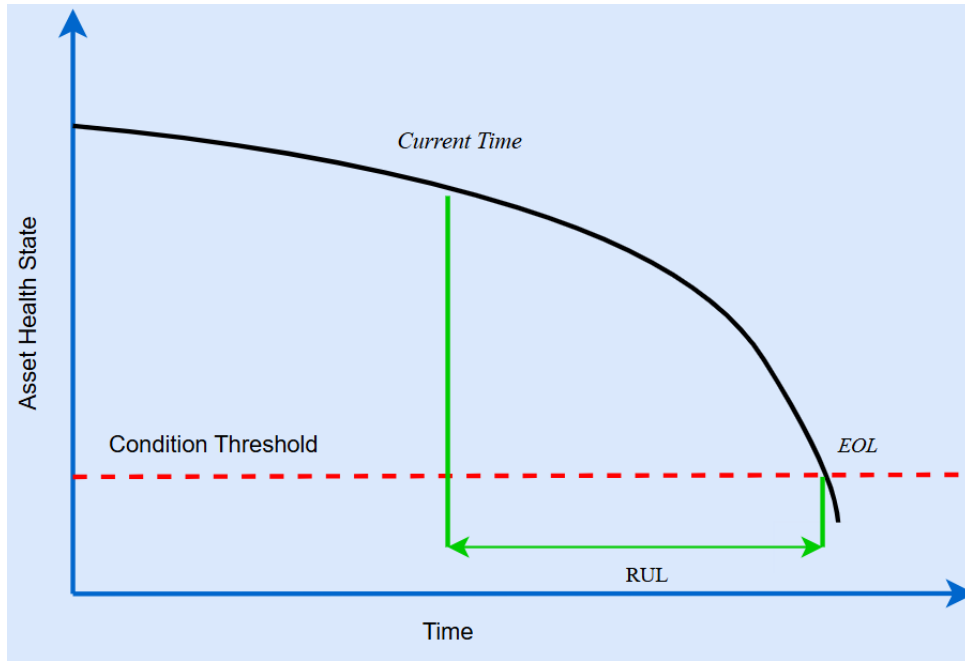


Figure 1.2: Equipment remaining useful life visualisation, adapted from Wang et al. (2020)

In addition, prescriptive maintenance is built directly on the capabilities established by CBM and PdM practices. Specifically, CBM provides the sensor-enabled condition monitoring infrastructure required to capture historical degradation, which is a fundamental prerequisite for PdM models. The impending-failure predictions and RUL forecasts produced by PdM are required inputs for modern prescriptive maintenance systems that automate the recommendations of specific actions (Nunes et al., 2023). Figure 1.3 depicts this progression and layered dependency for the fundamental maintenance strategies.

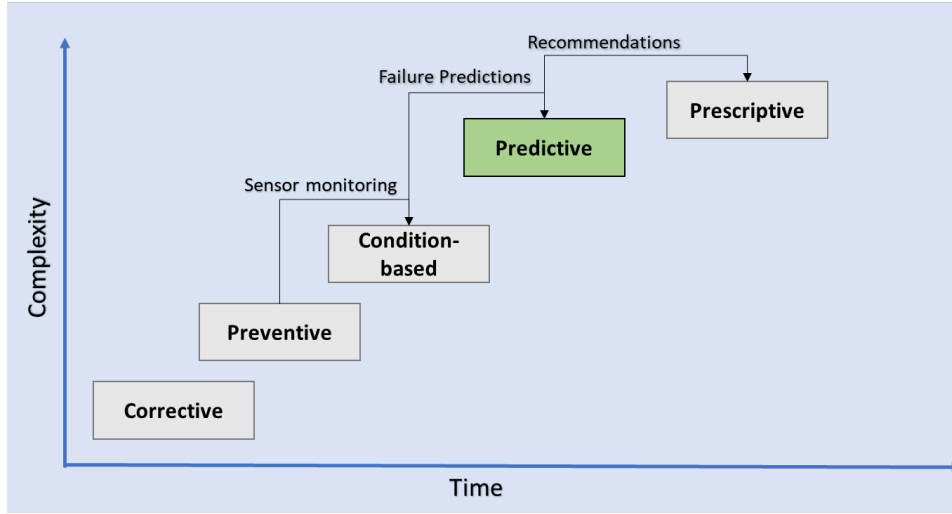


Figure 1.3: Evolution of the fundamental industrial equipment maintenance strategies, adapted from Nunes et al. (2023)

This study focuses on the predictive layer, which is the fundamental element that enables downstream prescriptive decision-making (Nunes et al., 2023). The implementation of PdM in industries has been shown to serve numerous benefits. Nunes et al. (2023) state that PdM contributes to extending the life of the equipment and improving the understanding of it. Wickern (2019) highlights how these practices reduce downtime and increase the quality of the product. Additionally, Dalzochio et al. (2020) note a reduction in system faults and failures as a significant benefit of PdM. In addition, it has been shown to improve operational efficiency and optimise supply chain management (Daily and Peterson, 2017). A commonly agreed-upon advantage of PdM implementation is the significant reduction in costs (Dalzochio et al., 2020; Nunes et al., 2023; Wickern, 2019; Daily and Peterson, 2017).

Appendix A presents a systematic literature review (SLR) of the techniques used for RUL prediction. It discusses the common limitations and challenges faced in the literature in section A.4 (as well as later discussed in section 2.5), the types of data used and the equipment from which the data is sourced in section A.5, and the feature extraction (FE) techniques used in section A.6. In addition, it discusses the RUL prediction models used in section A.7, the training optimisers and hyperparameter tuning techniques used in section A.8 and section A.9, respectively, as well as the traditional performance evaluation metrics used in section A.10 and the future work commonly proposed for selected articles in section A.11.

1.1.1 Problem Background

The application of RUL prediction can help plan PdM to reduce unplanned downtime and maintenance cost and increase asset lifespan (Achouch et al., 2022). The use of RUL prediction in PdM has attracted increased attention among researchers over the last decade (subsection A.3.3; Figure A.4). The transition from measuring model performance in academic research environments to achieving consistent/reliable model performance in industrial environments remains challenging. Models developed in controlled laboratory/benchmarking environments have been shown to degrade when operated in other real-world operating distributions (Asutkar and Tallur, 2023). This is largely due to the fundamental challenge apparent in PdM research, which is the lack of availability of labelled data that reflects real world distributions.

Data limitations are especially prevalent when models are used in equipment that exhibits new modes of operation or previously unseen faults, for which target-domain labels might not be accessible, despite existing source-domain labelled data (Fan et al., 2020). With respect to these new operating conditions or faults, historical labelled data may exist for a related source domain, while target-domain labels may be unavailable. For example, Da Costa et al. (2020) formulates the RUL transfer-learning problem under the assumption that labels are available only for the source domain and not for the target domain. Thus, evidently, PdM can involve varying degrees of availability of labelled target-domain-data. This can range from sufficiently labelled target data (for supervised learning) to limited labelled target data to scenarios where there are no labelled target data, where only unlabelled sensor data are available for the target domain. This motivates the need to understand which modelling approaches remain effective under different levels of labelled target-domain-data availability.

Furthermore, even when labelled data do exist, additional difficulties arise with regard to the complexity and variability inherent in the equipment being monitored. Physics-based models (physics-of-failure) are often based on simplifying assumptions made because the underlying physical mechanisms leading to failure are often unknown. These simplifying assumptions may not always apply in practice (Coble et al., 2020). These factors can contribute to the domain shift between the training and real-world data distributions. The degree of this domain shift is significant since the model performance depends on how closely the source and target domains are related. When the source and target domains are not sufficiently similar, transfer learning techniques can degrade performance rather than improve it, leading to negative transfer (Wang et al., 2019). Therefore, developing a suitable modelling strategy depends upon the level of availability of labelled target domain data and the similarity between the source and target domains.

Most PdM researchers focus on neural-network architectures to predict RUL, as is evident in section A.7. However, few studies have systematically compared different architecture

types (i.e. regression models, tree-based models, and deep learning models) when using different levels of available data. Furthermore, model evaluation is typically reported using a single measure of accuracy in a single training scenario. Moreover, most studies cited in the SLR prioritise point-accuracy metrics, with RMSE and scoring functions being the most frequently used (section A.10). Robustness in different data availability scenarios is reported less often as a performance metric. This is evident, since only one of the thirty reviewed studies evaluated performance under reduced training data fractions (subsection A.10.5), leaving uncertainty as to which model architectures remain robust and which evaluation approaches are most informative as the amount of training data becomes limited.

1.1.2 Problem Statement

In PdM practices, the deployment of RUL prediction models to new equipment is often constrained by limited or unavailable labelled target-domain-data. Although labelled source-domain data may be available, domain shifts between source and target distributions can reduce model generalisation, with the severity of this shift varying across implementation scenarios. However, there is limited controlled evidence on how different model classes and transfer strategies perform across varying levels of target training data availability and domain shift severity. This leaves practitioners with insufficient guidance for model selection in real-world PdM deployment contexts.

1.1.3 Aim

The aim of this study is to address the challenge of implementing RUL prediction models in PdM scenarios where labelled target domain degradation data are limited or unavailable, and the source-target-domain shift reduces model generalisation. To achieve this, the study systematically assesses how different RUL prediction strategies perform at varying levels of target-data availability and domain shift severity. The study specifically investigates whether feature selection (FS), FE, supervised transfer learning, and unsupervised domain adaptation can improve performance under these constraints. This subsequently aims to provide insight into which approaches remain most robust under these varying conditions, thereby helping practitioners make more informed decisions when selecting RUL prediction models.

1.1.4 Objectives

The objectives of this study are to:

1. Conduct an SLR of techniques used for RUL prediction in PdM to identify key challenges, commonly applied prediction, feature engineering and optimisation approaches, and evaluation metrics, and use these findings to inform the selection of methods evaluated experimentally in this study.
2. Identify an appropriate benchmark data set with multiple operating conditions and variants of the fault mode to enable the analysis of varying ML models, as well as domain shift analysis and transfer learning techniques.
3. Create an experimental methodology for the study to ensure consistency across model families, including data preprocessing and evaluation.
4. Establish the evaluation metrics used, as well as introduce a metric that can quantify the robustness of the RUL prediction over the equipment's lifecycle.
5. Establish the theoretical basis for the modelling classes evaluated, including traditional regression, tree-based, deep learning, FE, and transfer learning/domain adaptation techniques, and highlight the key limitations and common challenges associated with RUL prediction.
6. Train and evaluate baseline model classes across multiple levels of training-data availability in order to identify which models remain most robust as data becomes scarce.
7. Assess FS approaches and quantify their impact on performance at different levels of training-data scarcity.
8. Assess neural-network-based FE techniques and quantify their impact on performance at different levels of training data scarcity.
9. Evaluate supervised transfer learning strategies that utilise source-domain knowledge when limited labelled target data are available and quantify performance across varying target data quantities.
10. Evaluate unsupervised domain adaptation strategies that operate when only unlabelled target data is available and establish source-only transfer baselines (the model only trained on the source domain to make predictions on the target domain without additional adaptation techniques) for comparison.
11. Investigate the severity of the domain shift as a determinant of the effectiveness of the transfer strategy.

1.1.5 Research Questions

The objectives outlined in [subsection 1.1.4](#) aim to answer the research questions below.

- R1** Out of the evaluated set of models, which perform the best at low-data settings?
- R2** Do feature engineering techniques such as FS and FE reduce or improve model performance?
- R3** Does the proposed metric lead to different conclusions compared to traditionally used metrics?
- R4** Do supervised transfer learning strategies that use source-domain knowledge improve performance when limited labelled target data are available?
- R5** Can direct application of source-domain models (without adaptation) provide competitive performance on target domains when target data is not available?
- R6** Can unsupervised domain adaptation strategies that function without labelled target data achieve better performance than models trained only on source-domain data and perform competitively compared to supervised approaches?
- R7** How does domain shift severity between source and target domains affect the relative effectiveness of different transfer learning strategies at different levels of data scarcity?

1.2 Structure of Dissertation

The dissertation consists of six additional chapters. [Chapter 2](#) presents the theoretical background of the RUL prediction algorithms used, FE, hyperparameter tuning, transfer learning, Maximum Mean Discrepancy (MMD), the metric used to analyse the domain shift, and transfer learning approaches used. The chapter also discusses the common limitations and challenges associated with PdM and RUL prediction. [Chapter 3](#) introduces the Commercial Modular Aero-Propulsion System Simulation dataset. Furthermore, the motivation for the selection of this dataset is discussed followed by data preprocessing steps and exploratory data analysis. In [Chapter 4](#) an overview of the methodology is provided.

[Chapter 5](#) discusses the implementation and validation of the models and strategies used in the study. This includes baseline models, FS, hybrid FE, supervised fine-tuning transfer learning, source-only transfer, and adversarial domain adaptation. [Chapter 6](#) presents the results and a discussion of the performance of the models in unseen data. This chapter also discusses the limitations of the study and identifies possible areas of future work. The dissertation is then concluded in [Chapter 7](#).

Chapter 2

Theoretical Background

This chapter outlines the theoretical overview for the methods used in this study to predict the RUL under limiting data. In line with the consolidation approach taken in the SLR, it is organised by the typical ML workflow (refer to [Figure A.1](#)). However, since the objectives (see [subsection 1.1.4](#)) consider the investigation of TL, this flow is expanded to include domain assessment and learning techniques alongside the conventional feature engineering and prediction modelling and training techniques used. Furthermore, the common limitations and challenges associated with PdM and RUL prediction are discussed.

It should be noted that pre-processing and selection of the data set are described separately in [Chapter 3](#) and evaluation metrics and procedures are explained in [Chapter 4](#). This chapter introduces three key families of ML algorithms, namely RUL prediction algorithms based on regression, tree-based, and deep-learning. This is followed by FE methodologies used to extract embedded data representations. Thereafter, optimisation methods for model training and hyperparameter tuning are discussed. Finally, the chapter considers domain assessment and transfer learning strategies to improve predictive performance when target-domain training data is limited.

2.1 RUL Prediction Algorithms

ML is the study of algorithms that have the ability to learn complicated relationships in data through the use of optimisation strategies (Hu et al., [2016](#)). PdM involves the use of these ML pipelines to combine industrial IoT technologies and forecast the time that manufacturing equipment needs maintenance. The predictive model algorithms used in these pipelines can vary greatly, each with their limitations and advantages. For a comprehensive cross-comparison of predictive algorithms, the work involves three ML families, namely, classical regression models, tree-based models, and neural-network-based mod-

els. This section provides a theoretical overview of each algorithm used in this paper. Moreover, while this section provides a theoretical overview, [section A.7](#) provides actual implementations for each of the models discussed in the context of PdM research, as well as for other algorithms commonly used in the literature.

2.1.1 Regression Models

Regression is a supervised learning approach that identifies relationships between independent and continuous dependent variables (Hassanat et al., [2024](#)). Specifically, the model aims to use the continuous features (independents) to predict the target value (dependents) (Ciampiconi et al., [2023](#)).

Multiple Linear Regression

One of the most fundamental and well-known ML algorithms is multiple linear regression (MLR) (Tran et al., [2025](#)). MLR can be described as an extension of simple linear regression (Tranmer and Elliot, [2008](#)) and is formulated as follows:

$$y_t = \beta_1 + \beta_2 x_{t2} + \cdots + \beta_n x_{tn} + u_t \quad (2.1)$$

where y_t is the dependent variable at time t , β_1 is the intercept term, $\beta_2, \dots, \beta_n$ are the regression coefficients associated with the explanatory variables, $x_{t2}, \dots, x_{tn}$, and u_t is the error term

This extension enables multiple independent variables to be included in one model, allowing each predictor x to contribute independently to the estimation of the target y (Castro and Ferreira, [2022](#)).

The key advantage of this algorithm is its simplicity to implement and its interpretability (Dar and Chand, [2024](#); Li and Li, [2026](#)). Moreover, this algorithm typically does not require hyperparameter tuning (see [section 2.3](#) on hyperparameter tuning). As a result, training is computationally efficient compared to most other ML algorithms (Li and Li, [2026](#)). Moreover, although MLR is known to underfit complex non-linear data, it can also often overfit high-dimensional datasets. In particular, when the number of predictors is large relative to the number of observations (Özden et al., [2024](#)).

Lasso regression

To address this limitation, penalised regression methods have been proposed to shrink coefficients toward zero (Özden et al., 2024). This essentially functions as a feature selector by shrinking the coefficients of unimportant features to zero so that they have little to no impact and subsequently reduce the dimension (Fahimifar et al., 2023).

Lasso regression is a variant of MLR that incorporates a regulariser variable. This can be derived by converting Equation 2.1 to an objective function as described in Equation 2.2:

$$\min_{\beta_0, \vec{\beta}} \sum_{t=1}^n \left(y_t - \beta_0 - \sum_{j=1}^p \beta_j x_{tj} \right)^2 \quad (2.2)$$

where $\min_{\beta_0, \vec{\beta}}$ indicates that the objective is to choose the scalar intercept β_0 and coefficient vector $\vec{\beta}$ that minimise the expression, $\sum_{j=1}^p \beta_j x_{tj}$ is the linear predictor, $y_t - \beta_0 - \sum_{j=1}^p \beta_j x_{tj}$ is the residual for observation t , and $\sum_{t=1}^n$ indicates that the squared residuals are summed over all n observations, where n denotes the total number of observations.

Then Lasso can be expressed by adding a penalty term that functions as a regulariser (Equation 2.3):

$$\min_{\beta_0, \vec{\beta}} \left\{ \sum_{t=1}^n \left(y_t - \beta_0 - \sum_{j=1}^p \beta_j x_{tj} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| \right\} \quad (2.3)$$

where: $\sum_{t=1}^n (\cdot)^2$ represents the sum of squared residuals over all n observations, λ is the regularisation parameter controlling the strength of the penalty, $\sum_{j=1}^p |\beta_j|$ is the penalty term (referred to as L1) that shrinks the coefficients toward zero.

One notable limitation of Lasso regression is that, due to the regulariser, the model can behave poorly on highly correlated data, where Lasso can select one of the several correlated variables arbitrarily (Jardillier et al., 2022). Subsequently, this leads to discarded information (predictors) that could potentially have been useful for prediction regardless of collinearity (Wang et al., 2011).

Ridge regression

To prevent information loss, algorithms such as ridge regression use a different approach to shrink coefficients. Although Lasso typically selects one variable from a correlated set, Ridge regression shrinks them toward each other (Engelbrechtsen and Bohlin, 2019). This

results in observed performance improvements for Ridge specifically in highly correlated datasets (Engebretsen and Bohlin, 2019).

Ridge regression builds on top of Lasso regression by replacing the absolute value penalty with a squared penalty which shrinks coefficients towards zero but never sets them exactly to zero. The equation can be formulated as follows (Equation 2.4):

$$\min_{\beta_0, \vec{\beta}} \left\{ \sum_{t=1}^n \left(y_t - \beta_0 - \sum_{j=1}^p \beta_j x_{tj} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right\} \quad (2.4)$$

where $\sum_{j=1}^p \beta_j^2$ denotes the L2 penalty term that shrinks the coefficients towards zero.

Elastic Net

Although ridge regression mitigates the information loss in highly correlated data, it trades off the functionality that Lasso has which enables variable selection (i.e. completely dropping features when not relevant). The Elastic Net algorithm overcomes this limitation by combining their penalties (L1 and L2) to allow the selection of subsets and improve performance when handling highly correlated features (Morozova et al., 2015). This algorithm can be expressed as follows Equation 2.5:

$$\min_{\beta_0, \vec{\beta}} \left\{ \sum_{t=1}^n \left(y_t - \beta_0 - \sum_{j=1}^p \beta_j x_{tj} \right)^2 + \lambda_1 \sum_{j=1}^p |\beta_j| + \lambda_2 \sum_{j=1}^p \beta_j^2 \right\} \quad (2.5)$$

where $\lambda_1 \sum_{j=1}^p |\beta_j|$ is the L1 penalty term, which encourages sparsity by shrinking some coefficients exactly to zero, and $\lambda_2 \sum_{j=1}^p \beta_j^2$ is the L2 penalty term, which smoothly shrinks coefficients toward zero and helps stabilise estimates when predictors are highly correlated. The addition of L1 and L2 penalties allows Elastic Net to perform variable selection while also retaining groups of correlated predictors more effectively than Lasso alone.

Polynomial Regression

Up to this point, the algorithms discussed (MLR, Lasso, Ridge, and Elastic Net) have all been based on the assumption of an underlying linear relationship of the predictors and response variable (Hazra and Gogtay, 2017; Kwak, 2025; Rubin et al., 2022). Polynomial regression is an algorithm that introduces non-linear polynomial terms of the predictors up to the n th degree. This allows for more flexible functional forms to fit non-linear data (Sagar et al., 2021). The equation describing polynomial regression is expressed as follows (Equation 2.6):

$$y_t = \beta_0 + \beta_1 x_t + \beta_2 x_t^2 + \cdots + \beta_k x_t^k + u_t \quad (2.6)$$

where $x_t^2, \dots, x_t^k$ are higher-order polynomial transformations of the explanatory variable, allowing the predictor to enter the model at increasing powers in order to capture non-linear patterns in the data, k denotes the degree of the polynomial, specified as a hyperparameter, which determines the highest power of the predictor included in the model.

The key limitation of using polynomial regression is its tendency to create models prone to overfit as the degree of the polynomial increases (Cohen, 2024). This leads to an increase in both the number of model parameters (increasing complexity) and the computational resources required (Vyas and Shah, 2022). Moreover, in a multivariate feature set, the proliferation of coefficients can grow exponentially, creating what is known as the "curse of dimensionality" (which is the exponential increase in the feature space as the dimension increases) (Ayvaz and De Lathauwer, 2022). This is because, for example, if you start with n input features and use degree k , the number of terms grows combinatorially (see Equation 2.7):

$$\text{Number of terms} = \binom{d+k}{k} \quad (2.7)$$

2.1.2 Tree-based Models

There are families of algorithms that can inherently handle non-linear patterns without the requirement of transforming predictors to the n th degree. This includes tree-based models that function by building many decision trees that split data based on feature values and combine their predictions to produce a more accurate result (Rokach, 2016). This section explains what a decision tree is and covers the three tree-based algorithms used in this study, namely Random Forest (RF), Extreme Gradient Boosting (XGBoost) and Light Gradient Boosting Machine (LGBM).

The Decision Tree

A decision tree is a tree-structured model consisting of nodes, branches, and leaf nodes. Specifically, each node represents a test on an attribute, the branches represent the outcome of that test, and each leaf node represents a class (classification) (Han et al., 2012). In regression trees, the same tree-like structure applies, but the leaf node represents predicted numerical values instead of a class label (Dobra and Gehrke, 2002). Figure 2.1 visualises this tree-like structure for an illustrative RUL regression problem. Here, $X_1, \dots, X_7$ represent sensor-input features that may be selected at the respective decision nodes. At

each node, the selected feature is compared with a threshold value learned during tree construction, and the observation follows the corresponding branch depending on whether the feature value is greater than or equal to, or below, that threshold. This process continues until a leaf node is reached. The RUL boxes at the bottom of the figure represent the numerical RUL prediction associated with each terminal leaf.

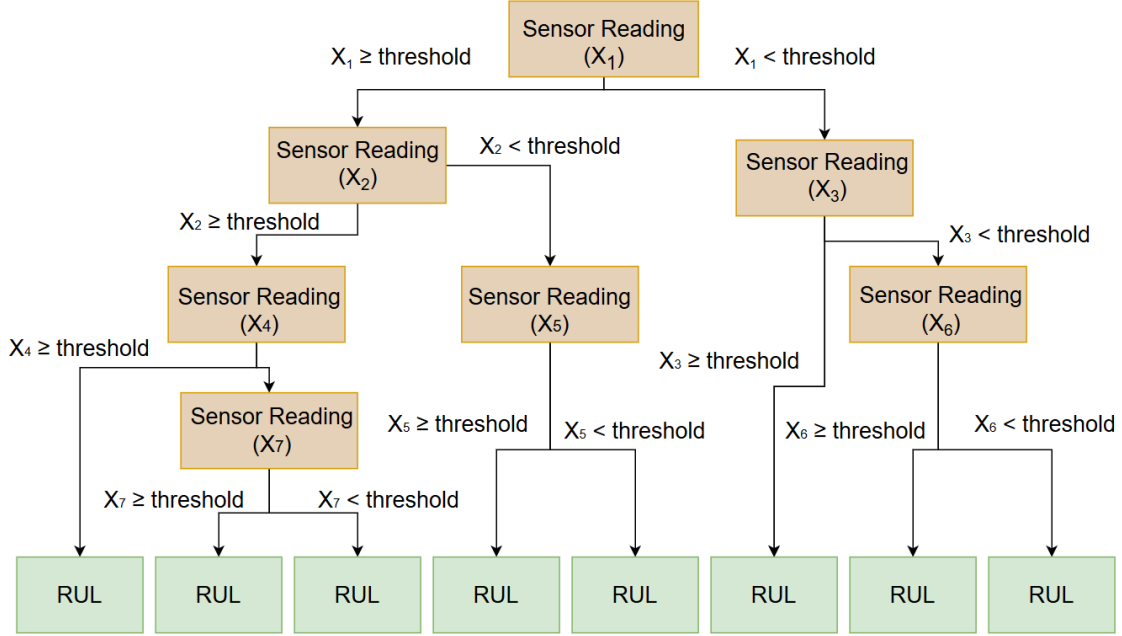


Figure 2.1: Illustrative structure of a regression tree for RUL prediction, where sensor features are successively compared with learned split thresholds until a terminal RUL prediction is reached.

Random Forest

RF regression builds on decision trees by forming multiple decision trees (a forest) to make a single classification or regression prediction. Figure 2.2 provides a visual representation of RF regression. The same sensor-input observation is evaluated by multiple independently trained regression trees. Each tree produces an individual RUL prediction value, illustrated as RUL_1 , RUL_2 , ..., RUL_B , where B denotes the total number of trees in the forest and the ellipses represent the additional trees omitted from the illustration. The individual RUL predictions are then averaged to obtain the final RF prediction.

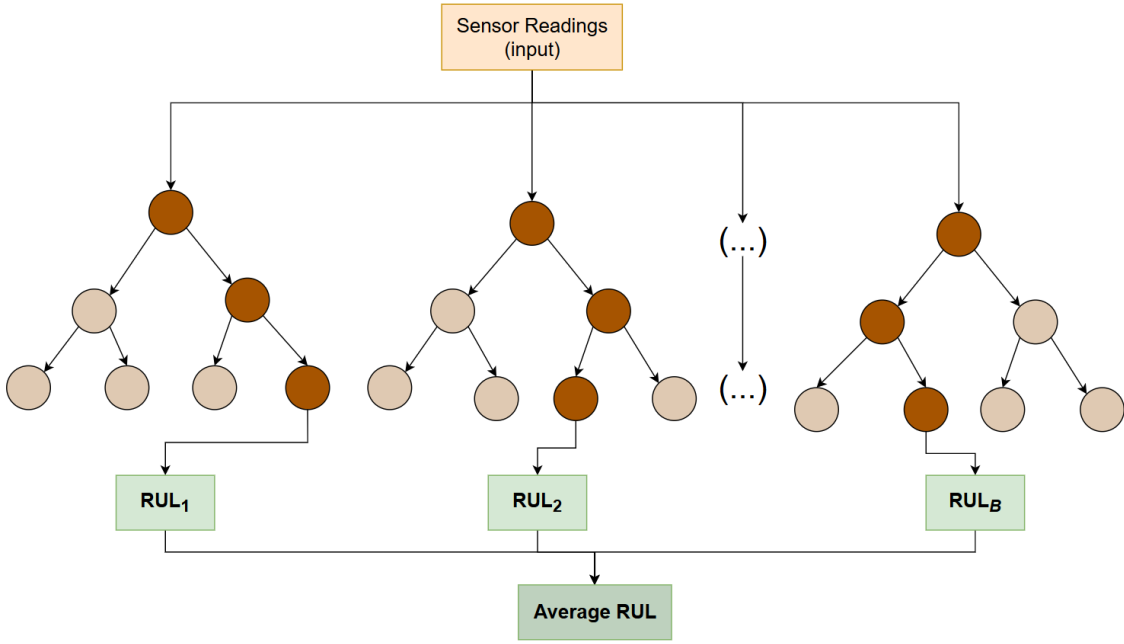


Figure 2.2: Visual representation of RF Regression, adapted from Lemeš and Akagic (2023).

Each tree in RF models is independently trained in different bootstrap (random selection of samples with replacement) samples of training data (Liaw and Wiener, 2002). The prediction of RF can typically be denoted as the average results (based on the bootstrap samples generated) of all trees (refer to Equation 2.8, taken from Mushagalusa et al. (2024):

$$\hat{\theta}^B(x) = \frac{1}{B} \sum_{b=1}^B t_b^*(x) \quad (2.8)$$

where $\hat{\theta}^B(x)$ is the RF prediction for input x based on an ensemble of B trees, B is the total number of decision trees in the forest, $t_b^*(x)$ denotes the prediction made for input x by the b -th decision tree, and $\frac{1}{B} \sum_{b=1}^B t_b^*(x)$ denotes the final prediction, which is the average of the predictions from all B trees.

Through the aggregation of diverse decision boundaries in the ensemble, the algorithm can capture complex non-linear patterns without requiring explicit feature engineering (Yang and Wang, 2025). Another key advantage of RF models is their strong performance on high-dimensional datasets. Biau (2012) motivates this by showing that the algorithm depends on the number of strong features, rather than the number of noise variables present. In addition, Breiman (2001) demonstrated that as the number of trees increases, RF predictions generally become more stable and less likely to over-fit.

Extreme Gradient Boosting

As discussed, RF constructs each tree independently, where each contributes equally to the final prediction. This approach is commonly referred to as bagging (Yang et al., 2019). In contrast, boosting methods build trees sequentially, with each new tree learning from the errors of its prior trees. Specifically, boosting iteratively applies a base learner to generate new prediction rules, which are then combined into a single rule that is expected to be more accurate than any individual base rule (Schapire, 2003). Figure 2.3 visualises the difference between bagging and boosting. In the figure, D represents the original training dataset, while $D_1, \dots, D_t$ represent the derived training samples used to train the individual predictors $P_1, \dots, P_t$, with t denoting the learner index and the ellipses indicating omitted intermediate learners. In bagging, the training samples and their corresponding predictors are generated independently and the resulting predictions are combined. In boosting, the predictors are instead constructed sequentially, with the performance of each preceding learner influencing the training of the subsequent learner, as indicated by the red arrows. The individual learners are then combined to generate the final prediction.

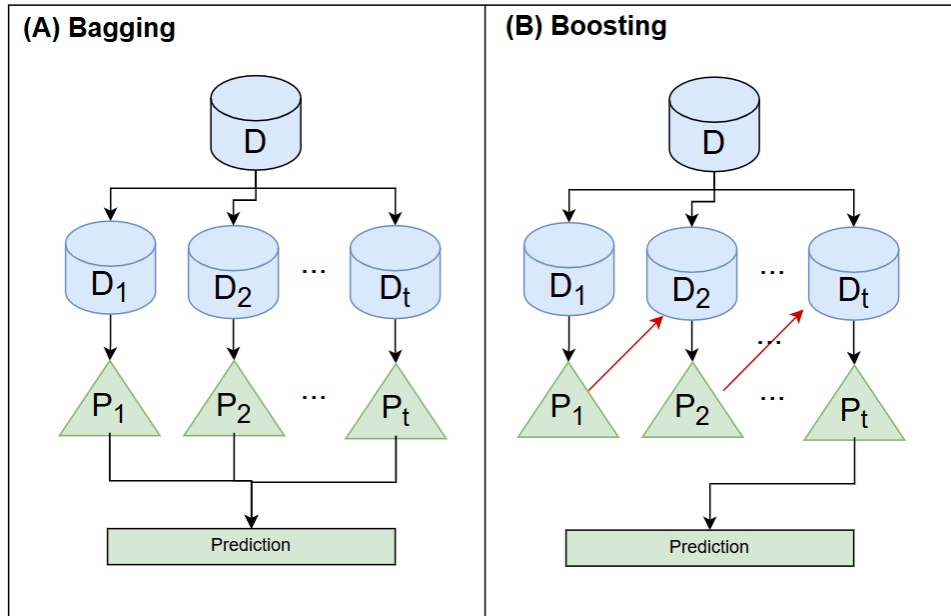


Figure 2.3: High-level comparison of bagging and boosting, showing independent learner construction in bagging and sequential learner dependence in boosting, adapted from Yang et al. (2019).

Similarly to bagging, boost methods create multiple datasets through the replacement of random sampling. However, this sampling is performed over weighted training data so that observations given a greater weight are more likely to be selected. The learners are then built sequentially and combined using a weighted averaging strategy (Schapire, 2003).

One of the most widely used boosting algorithms is XGB, commonly referred to as XGBoost. It is a regularised and computationally efficient variant of gradient-boosted trees. Its implementation involves trees added in a stage-wise manner that minimise a regularised objective (refer to Equation 2.9; taken from Chen and Guestrin (2016)).

$$\mathcal{L}^{(t)} = \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(\vec{x}_i)\right) + \Omega(f_t) \quad (2.9)$$

where: $\mathcal{L}^{(t)}$ is the objective function, $l\left(y_i, \hat{y}_i^{(t-1)} + f_t(\vec{x}_i)\right)$ denotes the loss calculated for the i -th observation, y_i is the true target value, $\hat{y}_i^{(t-1)}$ is the prediction value, $f_t(\vec{x}_i)$ represents the correction of the new tree, and $\Omega(f_t)$ is the regularisation term that penalises trees that are large (many splits, large weights, etc.).

Light Gradient Boosting Machine

LGBM is a gradient boosting method that, unlike XGBoost, does not evaluate every possible split point for every feature. Instead, it uses a histogram-based algorithm that groups continuous values in a limited number of bins and searches among these groups (Ke et al., 2017). This means that LGBM grows leaf-wise instead of level-wise (see Figure 2.4; adapted from Bui et al. (2021))

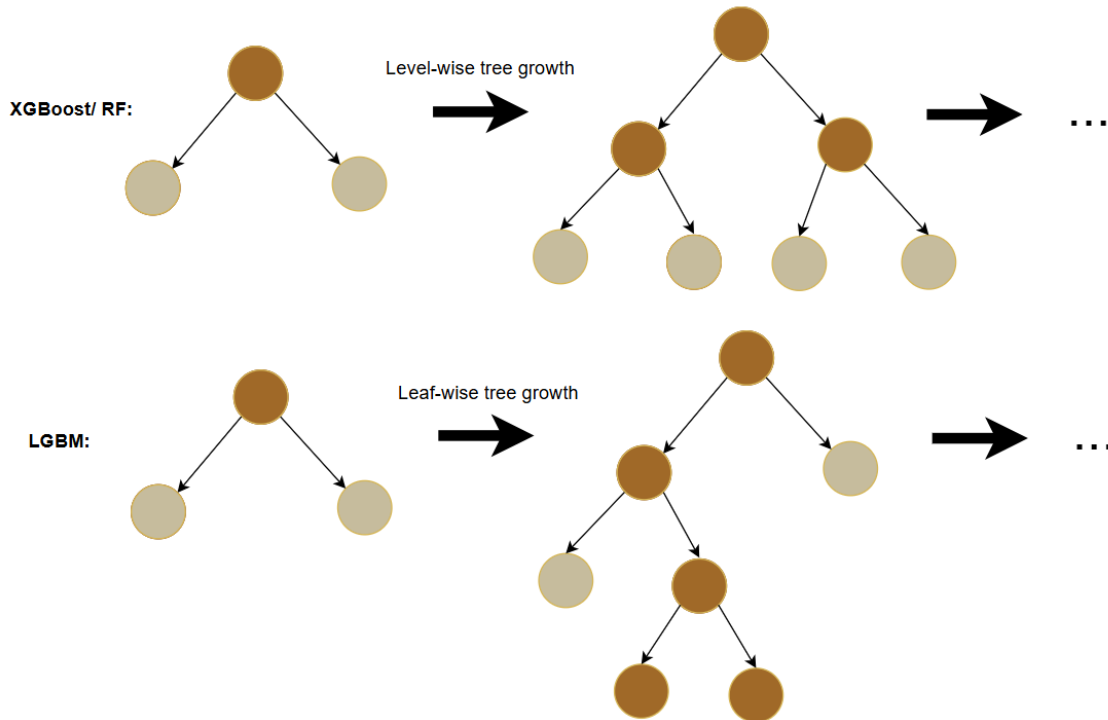


Figure 2.4: Level-wise vs Leaf-Wise Tree Growth, adapted from Bui et al. (2021).

Due to this structure/architecture, LGBM is known to be a more efficient boost method in both memory usage and training speed (Bui et al., 2021).

2.1.3 Artificial Neural-Networks

Artificial neural-networks (ANN) are commonly used for RUL prediction because they can model complex, non-linear relationships between inputs and outputs Gianola et al. (2011). The SLR, [section A.7](#) identified a diverse range of ANN-based models applied to the prediction of RUL. Among these, Long Short-Term Memory (LSTM) networks and Dense ANNs are the most used, each appearing in 23% of the selected studies. These are followed by Echo State Networks (ESNs), Gated Recurrent Units (GRUs), and Transformer-based models, each of which appeared in 7% of the reviewed studies.

This section provides an overview of the algorithms based on ANN selected for experimental evaluation. Specifically, it discusses the key foundations of ANNs and recurrent neural-networks (RNNs), and subsequently introduces the specific variants used in this work, namely Dense ANNs, LSTMs, GRUs and Transformers. Although ESNs have also demonstrated promising performance in RUL prediction, they are not included in the experimental evaluation conducted in this study. However, additional details on the application of ESNs to RUL prediction are provided in [subsection A.7.4](#). In addition, for the scope of this study, Gaussian-based algorithms, which have also been successfully implemented for RUL prediction (as evident in the SLR findings), are not included in the evaluation. Further discussion of Gaussian Process Regression (GPR) for RUL prediction is provided in [subsection A.7.5](#). It is further acknowledged that a wide variety of other algorithms have also achieved success in PdM contexts (refer to [Figure A.10](#) showing the distribution of algorithms/techniques found in the literature for RUL prediction) but fall outside the scope of this work. However, their inclusion would constitute a valuable direction for future research to investigate their performance under different levels of data scarcity.

Feed forward and Recurrent neural-networks

Conceptually, an ANN is an ML model inspired by a simplified abstraction of biological neural systems (Chen and Chen, 2026). These models are typically made up of interconnected artificial neurones that are represented by an activation function. Each of these neurones can accept multiple inputs, then multiply them by associated weights, sum the results, and produce an output when the resulting value passes the activation threshold (Jiang et al., 2024). An activation function is a mathematical transformation applied to the neurone's input so that each network can represent non-linear relationships (Dubey et al., 2022).

These neural-networks are trained using an algorithm known as "backpropagation". Backpropagation provides an efficient method for precisely computing the derivatives of a target quantity with respect to a large number of input quantities (Werbos, 2002). The gradients computed by backpropagation are then used by optimisers to update the parameters of neural-network-based models. This study uses the Adaptive Moment Estimation (Adam) algorithm. However, there are several optimiser algorithms that can be used to train networks. [section A.8](#) discusses optimisers in more detail in the context of PdM.

The fundamental variant of ANNs is feed-forward neural-networks. Its general structure comprises multiple perceptrons with input x_i , a hidden layer h_i (deep networks may have multiple layers) and an output layer O_i (Ma et al., 2019). In the context of RUL prediction, it can involve multiple inputs (sensor measurements) and only one output, which is the predicted RUL of the equipment.

A key limitation of feed-forward networks is that they treat the input as independent and cannot retain a state, which is necessary to capture information from long context windows (Lipton et al., 2015). An RNN is a type of neural-network that incorporates feedback connections, which enables it to handle sequential or time-series data. These connections give the model the ability to retain information from previous inputs and use it to inform subsequent outputs (Das et al., 2023). The fundamental difference between the ANN and RNN models is that RNN uses hidden layers that make current outputs relate to previous outputs through recurrent connections. This makes it particularly suitable for time series data (Si et al., 2011). The general structures of ANNs and RNNs are presented in [Figure 2.4](#).

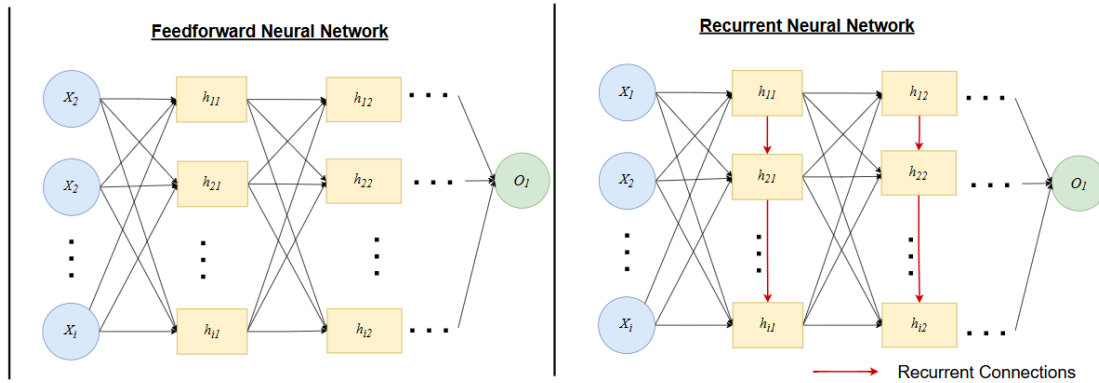


Figure 2.5: Simplified structures of feed-forward and RNN, adapted from Ma et al. (2019).

For RNNs, hidden layers maintain a memory of previous inputs over time, which enables the network to capture the temporal dependencies in sequential data (Tai et al., 2016). However, a limitation of RNNs is that these algorithms often struggle with long-term dependencies due to the problem of vanishing gradients (Si et al., 2011). More specifically,

during backpropagation, gradients (the derivative of loss with respect to a parameter) can decay or explode, causing instability in updating the weights in the neural-network (Ye et al., 2024).

Long Short Term Memory

RNNs that incorporate LSTM units have become an effective and scalable approach for a variety of learning tasks that involve sequential data (Greff et al., 2017). Although RNNs are historically difficult to train, advances in architectures and optimisation have enabled large-scale training, allowing LSTM and bidirectional RNNs to achieve strong performance across many tasks (Lipton et al., 2015). The general LSTM structure is presented in Figure 2.6. Here, x_i denotes the input vector sent to the LSTM at time step i , h_i denotes the corresponding hidden state, and o_i denotes the output produced at that time step. The terms x_{i-1} and h_{i-1} therefore represent the input and hidden state from the preceding time step, respectively. The recurrent connections allow information from earlier time steps to influence subsequent processing, with the long-term memory path representing information retained across the sequence and the short-term memory path representing the recurrent hidden-state information passed between adjacent time steps.

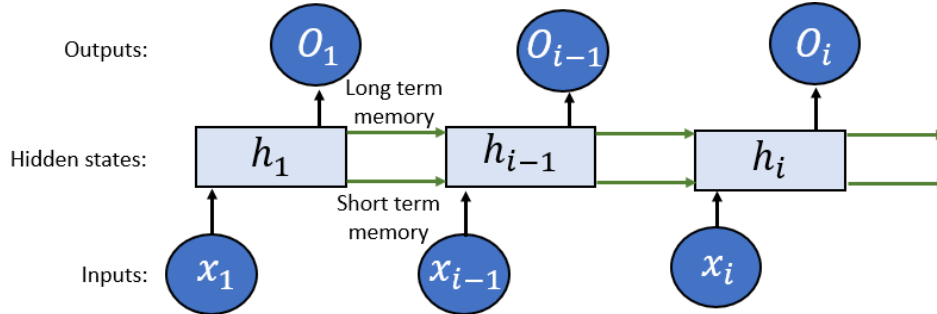


Figure 2.6: Recurrent sequence structure of an LSTM network, adapted from Fu et al. (2019).

LSTM architecture is made of a collection of recurrently connected sub-networks referred to as memory blocks. The purpose of each memory block is to preserve its state over time while controlling the flow of information through non-linear gating units (Van Houdt et al., 2020). Figure 2.7 illustrates the internal structure and gating mechanisms of an LSTM unit. The input represents the current input vector x_i , while the recurrent connection carries the hidden-state information h_{i-1} from the preceding time step. The cell represents the internal cell state, which carries longer-term information through the sequence. The forget, input, and output gates regulate the retention, addition, and release of information, respectively. In the figure, σ denotes the sigmoid gate activation, $\tanh$ denotes the hyperbolic tangent activation, the $+$ symbol represents summation, and the

filled multiplication symbol represents element-wise multiplication. The branch points indicate where information is passed to more than one subsequent operation.

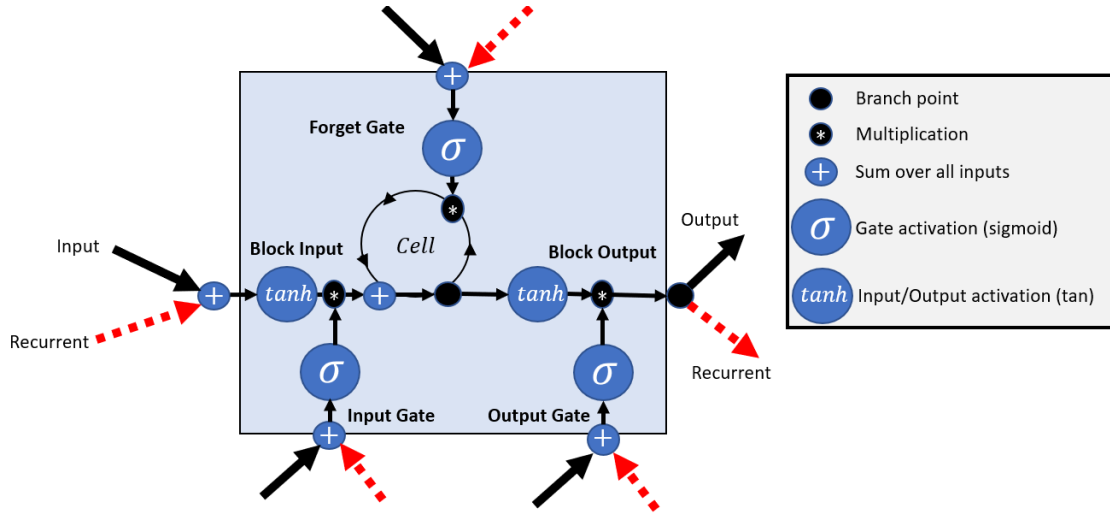


Figure 2.7: Internal structure and gating mechanisms of an LSTM unit (Fu et al., 2019).

These memory blocks consist of an input block and an input gate. The input block handles the processing of candidate data, and the input gate functions to regulate the read and write access of candidate content (Staudemeyer and Morris, 2019). For LSTM to create long-term memory, it uses the forget gate, which learns how long previously stored information should be retained in the cell state (Mojtahedi et al., 2025). The cell state functions as the memory component that preserves and carries information from one time step to the next (Hu et al., 2025). The cell's output is obtained by taking the cell state and multiplying it by the activation function of the output gate (Staudemeyer and Morris, 2019).

Bidirectional Long Short Term Memory

Bidirectional Long Short Term Memory (BiLSTM) utilises the same architecture as LSTM but additionally allows for additional training involving the transversal of data forward and backward (Siarni-Namini et al., 2019). This approach allows more key information to be extracted from the data (Yang and Wang, 2022).

Gated Recurrent Units

The Gated Recurrent Unit (GRU) is a simplified variant of the LSTM architecture. Structurally, GRU only uses two gating units while LSTM uses 3, and therefore has fewer parameters (Niu et al., 2023). This makes it a more computationally efficient variant of an

RNN (Diao et al., 2019) while still performing similarly to LSTM (Yunita et al., 2025). Figure 2.8 shows the general structure of the GRU, adapted from Riaz et al. (2020). The hidden state is passed through the reset and update gating mechanisms, which control how previously retained information is combined with the candidate hidden state. In the figure, σ denotes the sigmoid activation function, $\tanh$ denotes the hyperbolic tangent activation function, the multiplication symbols represent element-wise multiplication, the $+$ symbol represents addition, and $1 -$ represents the complement of the update-gate value. The resulting combination produces the updated hidden-state output.

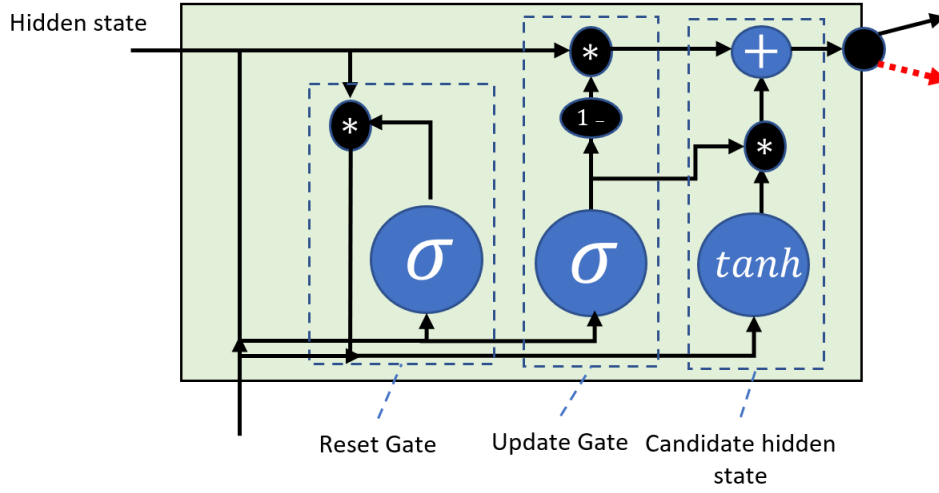


Figure 2.8: General GRU Structure, adapted from Riaz et al. (2020).

The reset gate allows the GRU to discard or overlook past hidden states whenever this is considered necessary (Chung et al., 2015). This can be formulated as follows (adapted from Chung et al. (2014)):

$$\vec{r}_t = \sigma \left(\mathbf{W}_r \vec{x}_t + \mathbf{U}_r \vec{h}_{t-1} \right), \quad (2.10)$$

where $\vec{r}_t$ is the reset gate vector, $\vec{x}_t$ is the input vector at time t , $\vec{h}_{t-1}$ is the previous hidden state vector, and the weight matrices are represented by $\mathbf{W}_r$ and $\mathbf{U}_r$. The sigmoid activation, σ , constrains the gate values between 0 and 1, and subsequently the reset gate determines how strongly information from the previous hidden state contributes when the candidate hidden state is formed.

The reset gate is applied element-wise to the recurrent contribution of the previous hidden state and subsequently influences the computation of the candidate hidden state, which is given by (adapted from Chung et al. (2014)):

$$\tilde{\vec{h}}_t = \tanh \left(\mathbf{W} \vec{x}_t + \vec{r}_t \odot \left(\mathbf{U} \vec{h}_{t-1} \right) \right). \quad (2.11)$$

where $\tilde{\vec{h}}_t$ is the candidate hidden-state vector at time step t , $\vec{x}_t$ is the input vector, $\vec{h}_{t-1}$ is the hidden-state vector from the previous time step, $\vec{r}_t$ is the reset-gate vector, $\mathbf{W}$ and $\mathbf{U}$ are learned weight matrices, $\odot$ denotes element-wise multiplication, and $\tanh(\cdot)$ denotes the hyperbolic tangent activation function.

The update gate then controls how much of the previous hidden state should be preserved and how much new information should be incorporated (Jafari and Byun, 2023). This can be formulated as follows:

$$\vec{z}_t = \sigma \left(\mathbf{W}_z \vec{x}_t + \mathbf{U}_z \vec{h}_{t-1} \right), \quad (2.12)$$

where $\vec{z}_t$ is the update-gate vector and $\mathbf{W}_z$ and $\mathbf{U}_z$ are the weight matrices associated with the update gate.

The update gate is then used to combine the previous hidden state with the candidate hidden state to produce the updated hidden state (Chung et al., 2014):

$$\vec{h}_t = (1 - \vec{z}_t) \odot \vec{h}_{t-1} + \vec{z}_t \odot \tilde{\vec{h}}_t, \quad (2.13)$$

where $\vec{h}_t$ is the updated hidden-state vector.

The advantage of the GRU structure is that it retains the ability to model long-term dependencies through its gating mechanisms, while its simpler architecture (with fewer gates than an LSTM) generally makes it more computationally efficient.

Transformers

The Transformer has become a major breakthrough architecture in fields such as natural language processing (Radford et al., 2018) and computer vision (Dosovitskiy et al., 2021). Through its use of attention mechanisms, it has been demonstrated to improve parallelisation and reduce training time relative to recurrent and convolution-based models (Vaswani et al., 2017). Transformers can model long-range dependencies and interactions, which makes them suitable for time series applications such as forecasting and anomaly detection (Wen et al., 2022). By using an attention mechanism rather than sequentially propagating hidden states across time steps as recurrent models do, the transformer reduces the path length between distant positions to a constant (Vaswani et al., 2017). The general structure of a transformer architecture is shown in Figure 2.9. In the encoder, the input sequence is first transformed through embedding and positional encoding before being processed by multi-head self-attention and a feed-forward network. Normalisation is applied around these processing blocks. The decoder follows a similar structure, but begins with the previous target values and applies masked multi-head self-attention. The

decoder then uses encoder-decoder multi-head attention to incorporate the representation produced by the encoder, after which the resulting representation passes through a feed-forward network and final linear layer to produce the forecast.

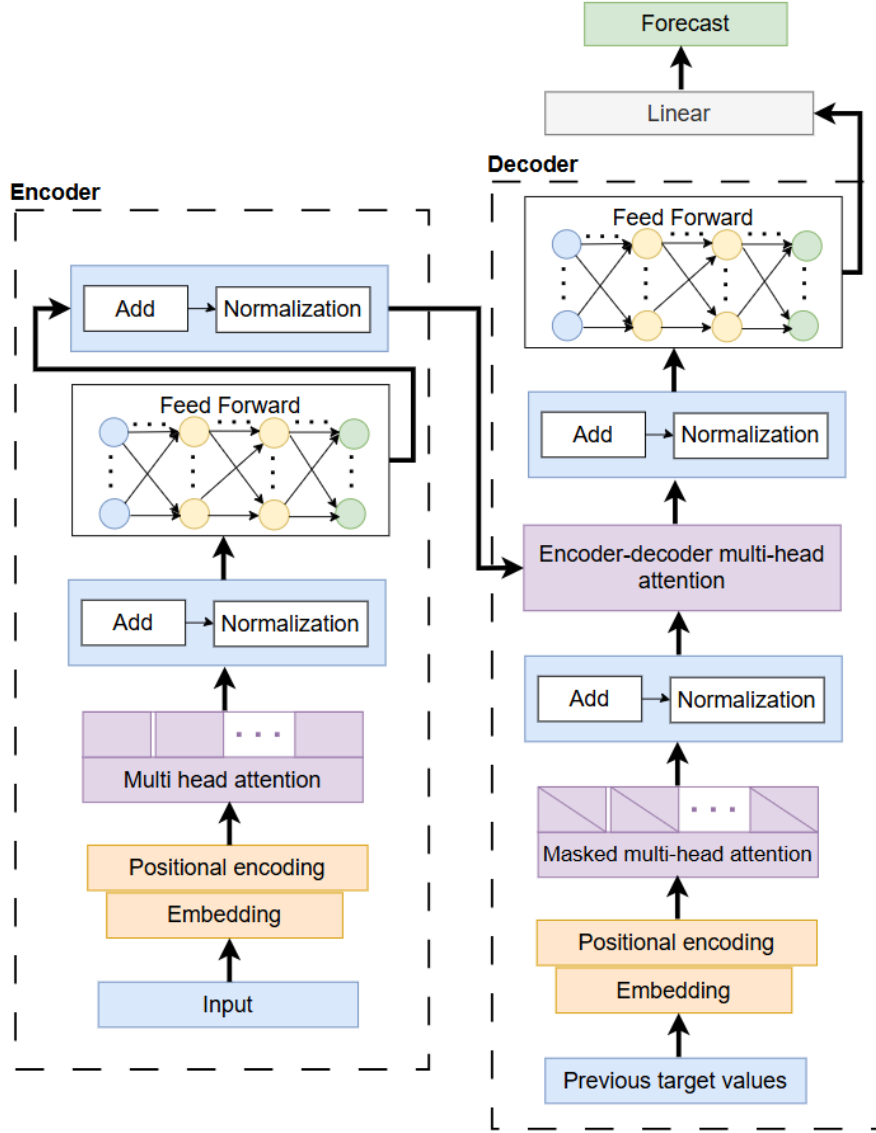


Figure 2.9: General Transformer Structure, adapted from Vaswani et al. (2017)

Transformer-based architectures typically consist of an encoder and decoder initially designed for sequence-to-sequence tasks such as machine translation (Vaswani et al., 2017). The encoder takes an input sequence and then uses multi-head self-attention as well as position-wise fully connected feed-forward network layers to encode representations of the input. The decoder blocks contain an additional cross-attention sublayer between the multi-head self-attention sublayer and the position-wise feed-forward network. Moreover, the self-attention of the decoder is masked to prevent each position from attending future positions in the sequence (Lin et al., 2022).

Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. It functions by applying several attention operations using different learnt linear projections of queries, keys, and values and concatenates the results into an output (Vaswani et al., 2017). Vaswani et al. (2017) derive the multi-head as follows:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\mathbf{H}_1, \dots, \mathbf{H}_h) \mathbf{W}^O \quad (2.14)$$

$$\mathbf{H}_i = \text{Attention}(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V) \quad (2.15)$$

where $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$, $\mathbf{W}_i^V$ are the learnt projection matrices for the queries, keys, and values in attention head i , where $\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$, $\mathbf{W}^O$ is the output projection matrix applied after concatenating the attention heads, with $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$, d_{model} denotes the model dimension, d_k is the dimensionality of the queries and keys, d_v is the dimensionality of the values, h is the number of attention heads, $\mathbf{H}_i$ is the matrix output of the i -th attention head, and $\text{Concat}(\mathbf{H}_1, \dots, \mathbf{H}_h)$ denotes the concatenation of the outputs from all attention heads.

The resulting representation is passed through a position-wise feed-forward network. This network applies the same two fully connected transformations to each position independently, with a ReLU activation function between them (Chen et al., 2022).

$$\text{FFN}(\vec{x}) = \text{ReLU}(\vec{x}\mathbf{W}_1 + \vec{b}_1)\mathbf{W}_2 + \vec{b}_2 \quad (2.16)$$

Although the encoder-decoder architecture works well for many types of sequence data tasks, time series-based tasks do not always require generating a complete output sequence. Instead, such as in the case of RUL prediction, it requires mapping a fixed-length window of past values to a single future value point. For these types of applications, the complexity of the architecture can be reduced to simply include an encoder (Chen et al., 2022). Transformers are also often implemented as feature extractors in ML frameworks due to their ability to capture long-term dependencies between elements in a sequence regardless of their distance (see subsection A.6.5 for more details on the implementation of transformers in PdM).

2.2 Feature Extraction

FE serves as a fundamental component in numerous ML pipelines (Mutlag et al., 2020). It is regarded a preprocessing step in which data is extracted and transformed into features

for further processing (Usmani and Shamsi, 2021). FE can greatly improve the performance of learning algorithms by identifying implicit, informative and meaningful features, while simultaneously minimising noise and redundancy Tesfay (2023).

There are several FE techniques, as identified in section A.6, Figure A.9. Traditional techniques involve the extraction of key statistical characteristics from the data, including root mean square, standard deviation (SD), mean, kurtosis, etc. These techniques can be effective but often fail to capture deeper relationships in data. More advanced techniques include projection-based methods such as Principal Component Analysis (PCA) or, for example, signal-processing methods such as wavelet transforms (Kuizinienė et al., 2024; Chin et al., 2024).

A key benefit of neural-network-based models is that they can learn feature representations and optimise prediction together within a single end-to-end differentiable architecture (Samanta et al., 2023). Deep learning allows computational models composed of multiple processing layers to learn representations of data with multiple levels of abstraction. This is achieved using the backpropagation algorithm, which indicates how the model should adjust its internal parameters to compute increasingly abstract representations across layers (LeCun et al., 2015).

There have been several high-performance hybrid architectures (FE and prediction algorithms). Some of the most widely featured extractors used in hybrid neural-network algorithms is used in PdM are Convolutional Neural-Networks (CNN), Temporal Convolutional Networks (TCN), and Autoencoders (AE), (as seen in Figure A.11, which shows the distribution of different FE and prediction model architecture combinations used in the selected papers in the SLR). In particular, CNN has been commonly used in both LSTM and dense ANN (feed-forward) architectures.

2.2.1 Convolutional neural-network

CNNs have become widely known, largely due to their remarkable performance in computer vision tasks, studies such as Krizhevsky et al. (2012) demonstrating their ability to learn progressively more abstract feature representations directly from raw pixel data for image classification. For time-series tasks, CNNs have also proved effective because the same convolutional architecture can be applied along a one-dimensional (1D) temporal axis and automatically learn complex representations from sensor signals (Duc et al., 2020).

CNNs consist of three layers, including convolutional, pooled, and fully connected layers. For one-dimensional (1D) CNNs the input is in the form of a 1D array. The convolutional layers then identify features by applying convolutional kernels which perform element-wise multiplication and summation operations on the input array. Then, the pooling

layers are commonly applied to reduce the dimensionality of the extracted features, while the fully connected layers merge the information from the convolutional and pooling layers to produce the final output (Wang et al., 2024).

A convolutional layer is characterised by locally connected neurones and shared weights (Li et al., 2019). The convolution operation can be expressed as follows (taken from Li et al. (2019))

$$x_{k'}^L = \phi \left(\sum_{k \in M_j} w_{kk'} * x_k^{L-1} + b_{k'}^L \right) \quad (2.17)$$

where x_k^{L-1} is the output of the k -th feature map in layer $L-1$, $w_{kk'}$ denotes the convolution kernel connecting the k -th feature map in layer $L-1$ to the k' -th feature map in layer L , $*$ denotes the convolution operation, $b_{k'}^L$ denotes the bias term associated with the k' -th feature map in layer L , ϕ represents the non-linear activation function, $x_{k'}^L$ denotes the output of the k' -th feature map in layer L , M_j is the set, or number, of input feature maps connected to the j -th local receptive field.

The pooling layers reduce the dimensionality of feature maps and improve the processing of noise data. Common implementations include mean-pooling and max-pooling, where max-pooling retains the largest value in each local receptive field (Yang et al., 2020). This can be calculated through Equation 2.18, adapted from Yang et al. (2020).

$$p_k^L(j) = \max_{(j-1)W+1 \leq t \leq jW} \left\{ x_k^{L-1}(t) \right\} \quad (2.18)$$

where $x_k^{L-1}(t)$ denotes activation at position t in the k -th feature map of layer $L-1$, $p_k^L(j)$ denotes the pooled output at position j in the k -th feature map of layer L , W denotes the width of the pooling window, j denotes the index of the pooling region.

2.2.2 Temporal Convolutional Network

TCNs are specifically designed for time series modelling, with the objective of overcoming key limitations of CNNs, particularly those related to temporal information leakage and the capacity to capture sequential context (Mo et al., 2023). A fundamental component of TCNs is their use of causal convolutions. In contrast to standard convolutions, causal convolutions ensure that the information from future time steps does not affect the previous ones (Liu et al., 2021). Figure 2.10 visually demonstrates this difference.

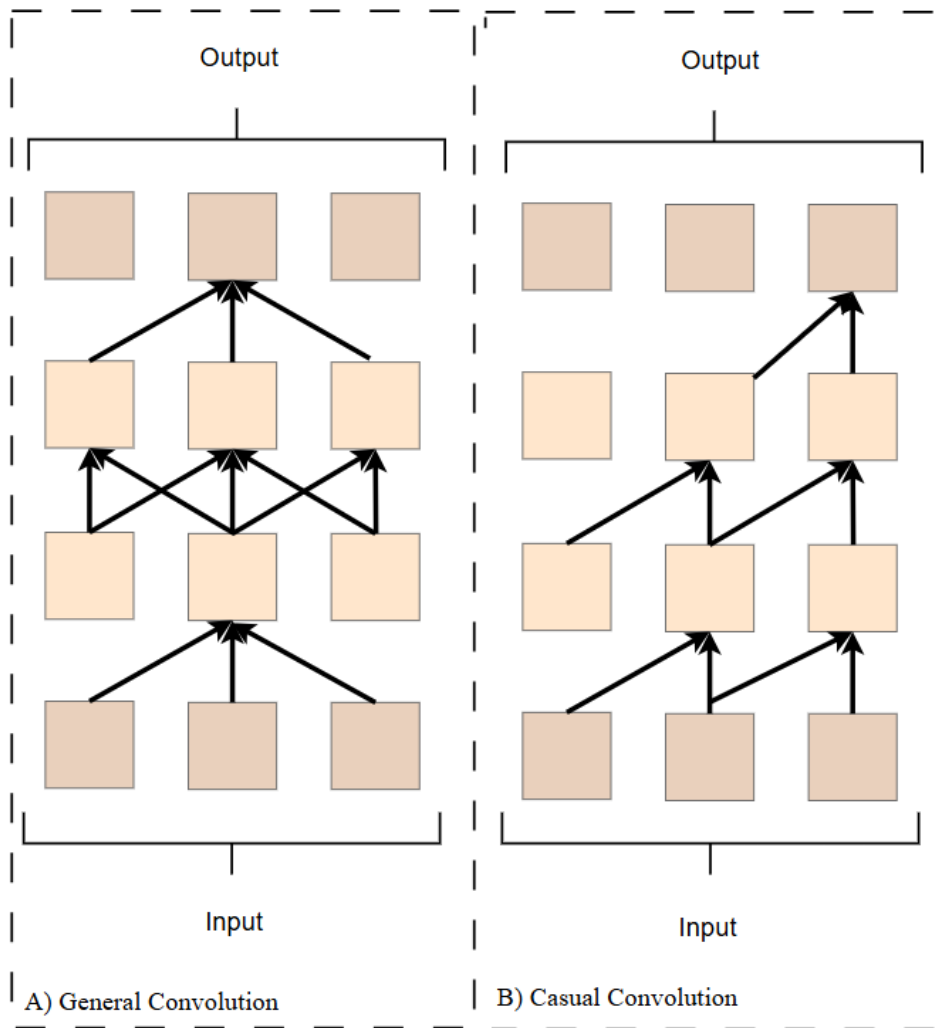


Figure 2.10: The general structure of convolution and causal convolution in TCNs, adapted from Hsu et al. (2022)

Additionally, because it uses dilated convolutions, its larger receptive fields enable it to capture information across longer ranges, whereas standard CNNs are limited to modelling local dependencies (Foroutan and Lahmiri, 2024). Figure 2.11 shows the general structure of a TCN, focusing on the dilution-causal operations.

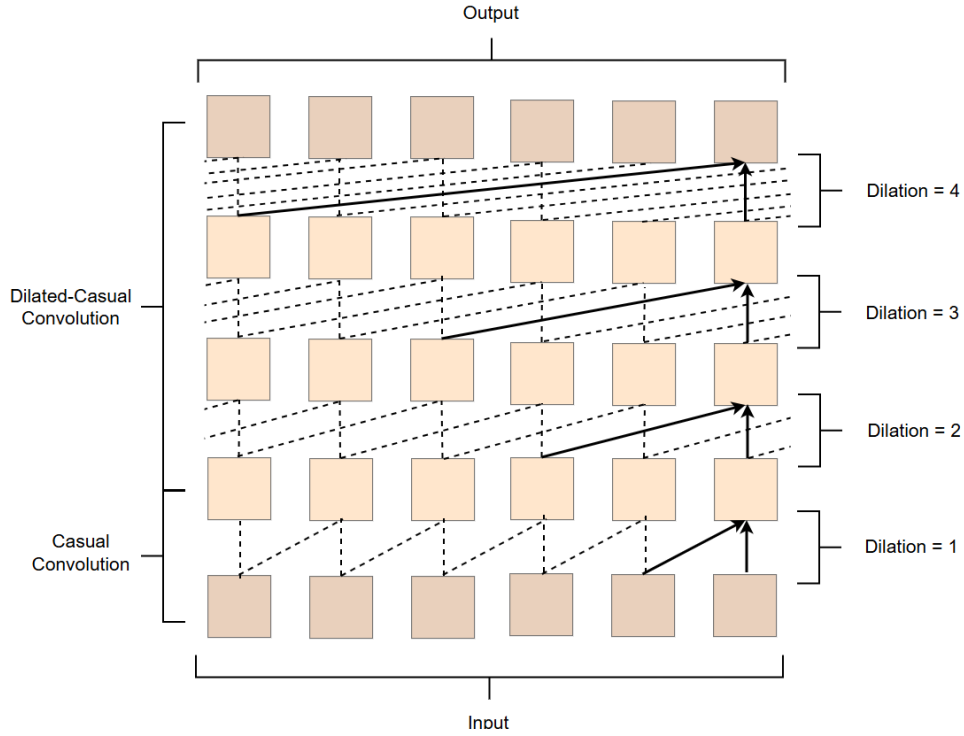


Figure 2.11: Dilated-causal convolution visualisation in TCNs, adapted from Duc et al. (2020)

Figure 2.11 shows how the dilation factor controls the spacing between the input positions considered by each convolutional layer. At a dilation factor of one, adjacent time steps are considered, while increasing the dilation factor progressively increases the distance between sampled inputs. This allows deeper layers to incorporate information from a wider temporal range without requiring a corresponding increase in the convolutional filter size.

The dilated convolution operation used by a TCN can be expressed as follows (adapted from Bai et al. (2018)):

$$F(s) = (\vec{x} *_d \vec{f})(s) = \sum_{i=0}^{k-1} f(i)x_{s-di}, \quad (2.19)$$

where $F(s)$ is the output of the dilated convolution at sequence position s , $\vec{x}$ is the input sequence, $\vec{f}$ is the convolutional filter, $f(i)$ is the i th filter coefficient, k is the filter size, and d is the dilation factor.

2.2.3 Autoencoders

AEs are types of neural-networks that contain an encoder that compresses information with the aim of maintaining a meaningful representation and then a decoder that reconstructs the input as closely as possible to the original (Bank et al., 2023). AEs are trained unsupervised (unlike CNN or TCN which uses labelled targets) by using the input itself as a reconstruction target to learn a compact latent representation (Berahmand et al., 2024). The general structure of AEs is shown in Figure 2.12.

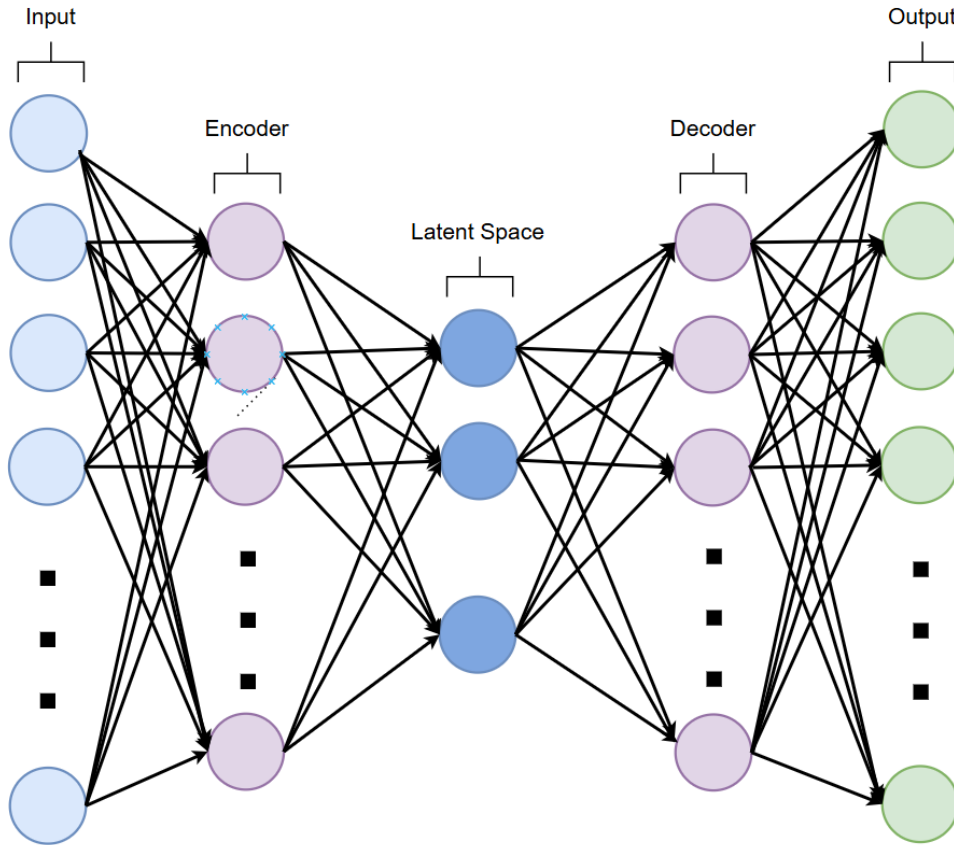


Figure 2.12: General structure of an AE, adapted from Lu et al. (2021)

An AE has three distinct parts, namely, the encoder, latent space, and decoder. At the encoder, the model compresses the input into a lower-dimensional representation. This representation is known as latent space. In the latent space, the compressed representation captures important features of the input data and then the decoder reconstructs the compressed representations of the features back to the original form. Aydemir and Paynabar (2020) describes AE as a non-linear extension of PCA. PCA is a linear technique that is used to reduce the dimensionality of data, but only effectively in linear relationships. On the other hand, AE approaches often use non-linear activation functions for both the encoder and decoder, which can capture more complex non-linear patterns. The

formulation of the decoder and encoder is described in Equation 2.20 and Equation 2.21 respectively (taken from Michelucci (2022)).

$$\vec{h}_i = g(\vec{x}_i) \quad (2.20)$$

$$\vec{\tilde{x}}_i = f(\vec{h}_i) = f(g(\vec{x}_i)) \quad (2.21)$$

where x_i is the sampled input, $g(\cdot)$ denotes the encoder function, h_i is the latent representation of the input, $f(\cdot)$ represents the decoder function, $\tilde{x}_i$ denotes the reconstructed version of the sampled input. Equation 2.20 and Equation 2.21 show that the encoder maps each input vector $\vec{x}_i$ to a latent representation $\vec{h}_i$, while the decoder maps this compressed representation back to the reconstructed input $\vec{\tilde{x}}_i$. The latent representation therefore retains the information considered most useful for reconstructing the original input while representing it in a lower-dimensional form

Training the AE implies finding that the functions $g(\cdot)$ and $f(\cdot)$ satisfy Equation 2.22

$$\arg \min_{f,g} \left\langle \Delta (\vec{x}_i, f(g(\vec{x}_i)))^2 \right\rangle \quad (2.22)$$

where Δ measures the squared difference between the input and its reconstruction (output), and $\langle \cdot \rangle$ denotes the average over all input samples in the dataset (Michelucci, 2022). For examples in the literature where AEs are implemented for RUL prediction, see subsection A.6.3.

The objective can be seen as minimising the reconstruction errors for the inputs and their reconstructions. Minimising these differences over the training set will optimise both the encoders and decoders such that the encoded features retain the important characteristics needed to reproduce the input. Therefore, since the number of dimensions in the latent space is less than or equal to the number of dimensions in the input, the Autoencoder learns a compact representation of the most significant patterns in the data as opposed to learning one independent representation for each of the original features.

2.3 Hyperparameter Tuning

ML models typically consist of weighted parameters that are optimised during training. These parameters fall within the actual architecture of the prediction algorithm. In contrast to the model parameters, hyperparameters are chosen before the start of training and control how the training process is carried out (Ilemobayo et al., 2024). Selecting

an appropriate combination of hyperparameters critically impacts how well ml models will perform and generalise on test data (Arnold et al., 2024). It is therefore vital to select a hyperparameter tuning technique that will effectively find an optimal combination. Typically, hyperparameter tuning is implemented by training models under different hyperparameter settings and selecting the combination that performs best on a validation data set.

As seen in [section A.9](#) there are several hyperparameter techniques that have been used to tune the hyperparameter. The most prominent are Grid Search (GS), Manual tuning, Genetic Algorithms (GA), Bayesian Optimisation Algorithm (BOA), and Random Search.

GS is one of the most effective methods of hyperparameter tuning, as it uses a brute force approach by testing every possible combination of hyperparameter. However, the trade-off of its effectiveness is its lack of efficiency, as it can be an impractical choice when dealing with ML algorithms that consist of a large set of hyperparameters. Bergstra and Bengio (2012) highlights this limitation by noting that GS constructs the Cartesian product of the potential value sets for each hyperparameter, thereby “suffering from the curse of dimensionality”, since the number of possible combinations grows exponentially with the number of hyperparameters.

Random search directly addresses this limitation. Instead of testing each possible combination, it uses random combinations of hyperparameters. This makes it much more efficient than GS while remaining relatively simple to implement (Bergstra and Bengio, 2012). However, in contrast, this method may overlook the best-performing combinations if they do not appear in the random sample runs.

More advanced techniques such as GA involve optimisation to converge to a near-optimal set of hyperparameters without the need to exhaustively test combinations. These algorithms are particularly suitable for high-dimensional hyperparameter spaces (Mehdary et al., 2024). GA is inspired by the evolutionary process of natural selection. Beg and Islam (2016) explains how this algorithm works in five steps.

Table 2.1: Main steps of the genetic algorithm for hyperparameter optimisation, adapted from Beg and Islam (2016).

Genetic Algorithm Step	Description
Step 1: Population Initialisation	The genetic algorithm randomly generates an initial population of candidate hyperparameter configurations.
Step 2: Selection	The fitness of each candidate hyperparameter configuration is evaluated, and the better-performing candidates are selected for reproduction.
Step 3: Crossover	Segments of selected hyperparameter configurations are combined to produce new offspring and generate new candidate configurations.
Step 4: Mutation	Random changes are introduced into certain hyperparameter configurations to maintain diversity in the search space.
Step 5: Elitist Operation	The best-performing hyperparameter configurations from the current generation are carried forward to the next generation.

It is acknowledged that, as discussed, there are many approaches to hyperparameter tuning. Within the scope of this study, GS is used for the simpler regression algorithms, while a BOA approach is used for the remaining models.

2.3.1 Bayesian Optimisation Algorithm

The BOA has become the leading approach for costly black-box optimisation problems due to its efficient use of data (Wang et al., 2023). Unlike classical optimisation methods, BOA typically does not rely on gradient information (Wu et al., 2017). Specifically, it applies the GPR model to build an approximation of the objective function and estimate the uncertainty involved with it. The next evaluation point is determined by selecting a point according to the acquisition function (Wu et al., 2017).

Optuna

This study implements BOA through the Optuna framework. Optuna is an automatic hyperparameter optimisation framework proposed by Akiba et al. (2019). Optuna applies BOA by dynamically narrowing down the range in which the best parameters lie based upon each trial’s results. Additionally, Optuna can limit the number of computationally expensive trials by pruning (preventing non-promising trails). Furthermore, because Op-

tuna uses a define-by-run API, it allows the search process to be customised to the specific requirements of the model (Al-Salih et al., 2026). The Optuna framework supports several sampling strategies, including independent methods such as TPE (Tree-Structured Parzen Estimator) and relational methods such as CMA-ES (Covariance Matrix Adaptation Evolution Strategy) (Akiba et al., 2019).

This study uses Optuna to implement BOA for hyperparameter tuning. Optuna is a framework, proposed by Akiba et al. (2019), created for automated hyperparameter tuning. Optuna applies BOA by dynamically narrowing down the range in which the best parameters lie based upon each trial’s results. In addition, it can limit the number of computationally expensive trials by pruning (stopping unpromising trails) (Al-Salih et al., 2026). Optuna supports several sampling strategies, including independent methods such as the Tree-Structured Parzen Estimator (TPE) and relational methods such as CMA-ES (Covariance Matrix Adaptation Evolution Strategy) (Akiba et al., 2019).

This study uses TPE as the sampling methodology. TPE is considered an exploitative method of BOA due to its acquisition function acting in a manner similar to probability of improvement (PI). Therefore, it favours candidate hyperparameter configurations that are more likely to produce an improved result over the current optimum solution (Watanabe, 2023).

Optuna uses the TPE as its sampling strategy. This strategy is a variant of BOA as it uses conditional densities of the hyperparameters rather than directly approximating the objective function. This can be formulated through Equation 2.23, taken from Watanabe (2023):

$$p(y \mid \vec{x}, D) = \begin{cases} l(\vec{x}) = p(\vec{x} \mid D^{(l)}) & \text{if } y \leq y^\gamma \\ g(\vec{x}) = p(\vec{x} \mid D^{(g)}) & \text{if } y > y^\gamma \end{cases} \quad (2.23)$$

2.4 Transfer Learning

The methods discussed thus far have focused on predictive model architecture, FE, and performance optimisation through hyperparameter tuning. However, when the amount of training data available is restricted, improvements obtained through model selection and tuning alone may remain limited. In such cases, transfer learning (TL) becomes a useful approach as it allows models to gain knowledge from other source domains. Weiss et al. (2016) define TL as a method that improves the performance of a model in one domain by using knowledge from another domain, thus meeting the need for high-performing learners that can be trained using more readily available data from various domains (Weiss et al., 2016).

There are several benefits in applying TL architectures, including improved accuracy and efficient use of scarce data (Hattula et al., 2023). The review by Iman et al. (2023) emphasises the increasing relevance of transfer learning within the field of ML, particularly due to its ability to reduce the reliance on large volumes of training data and the associated high costs of model training.

Transfer learning is commonly categorised as intra-domain or cross-domain TL. Intra-domain TL involves the transfer of knowledge between datasets from the same or closely related source domain, whereas cross-domain TL involves the transfer of knowledge between different domains (Otović et al., 2022). TL can also be described as homogeneous when the source and target domains share the same feature space and heterogeneous when their feature spaces differ.

Several studies have investigated the application of TL for RUL. For example, Sun et al. (2019) proposed a deep homogeneous TL method based on a sparse AE to predict the RUL of cutting tools operating under different conditions. Similarly, Xiang et al. (2025) introduced a micro transfer learning mechanism for the prediction of RUL and evaluated it on a turbo-engine and further validated it on data sets from wind turbine gearboxes. Na et al. (2025), proposed a heterogeneous TL approach in the cross-domain to transfer knowledge from the prediction of RUL of aircraft engines to the prediction of railway track circuit faults. The feasibility of transfer was assessed using MMD to compare the source and target data distributions. The target task was reformulated as a classification problem by discretising RUL values into fault classes, and knowledge transfer was achieved using a Data Distribution Fitting Network (DDFN).

2.4.1 Maximum Mean Discrepancy

TL involves a source dataset $D_s = \{x_i^s, y_i^s\}_{i=1}^{n_s}$ and a target dataset $D_t = \{x_i^t, y_i^t\}_{i=1}^{n_t}$. The greater the similarity between the source and target domains, the more accurate the forecasts for the target tend to be (Ehrig et al., 2024). The difference between them, often referred to as domain shift, is the discrepancy between the source and target data distributions (Li et al., 2025)..

MMD can be used as a metric to measure the discrepancy between the source and target distributions. Specifically, it is a kernel-based metric for quantifying how two probability distributions differ, obtained by computing the distance between their mean embeddings in a reproducing kernel Hilbert space (Gretton et al., 2012). Gretton et al. (2012) defines MMD as (Equation 2.24):

$$\text{MMD}[\mathcal{F}, p_s, p_t] = \sup_{f \in \mathcal{F}} (\mathbb{E}_{\vec{x} \sim p_s}[f(\vec{x})] - \mathbb{E}_{\vec{x} \sim p_t}[f(\vec{x})]) \quad (2.24)$$

where $\mathcal{F}$ is a class of functions $f : \mathcal{X} \rightarrow \mathbb{R}$, p_s and p_t are the probability distributions of the source and target, $\mathbb{E}_{\vec{x} \sim p_s}[f(\vec{x})]$ and $\mathbb{E}_{\vec{x} \sim p_t}[f(\vec{x})]$ are the expectations of $f(\vec{x})$ under the source and target distributions, respectively, and $\sup$ denotes the supremum over $\mathcal{F}$.

By choosing $\mathcal{F}$ as the unit ball of a universal reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ defined on a compact metric space $\mathcal{X}$, MMD can be expressed as the distance between the mean embeddings of the source and target distributions in $\mathcal{H}$:

$$\text{MMD}^2[\mathcal{F}, p_s, p_t] = \|\vec{\mu}_{p_s} - \vec{\mu}_{p_t}\|_{\mathcal{H}}^2 \quad (2.25)$$

Here, D_s and D_t denote samples drawn from the underlying source and target distributions p_s and p_t , respectively. $\mathcal{F}$ is the unit ball in $\mathcal{H}$, which is the set of functions in $\mathcal{H}$ with the RKHS norm less than or equal to 1. $\mathcal{H}$ is a reproducing kernel Hilbert space defined on the compact metric space $\mathcal{X}$, $\vec{\mu}_{p_s}$ and $\vec{\mu}_{p_t}$ are the mean embeddings of the source and target distributions in $\mathcal{H}$, respectively, and $\|\cdot\|_{\mathcal{H}}$ denotes the RKHS norm in $\mathcal{H}$ (Gretton et al., 2012).

Although MMD provides evidence of potential transferability, transferability itself must be demonstrated through empirical transfer learning (Na et al., 2025). This dissertation implements two transfer learning approaches for intra-domain, homogenous tasks, through fine-tuning and domain-adversarial neural-network (DANN)-based learning.

2.4.2 fine-tuning

Given that labelled target data are available in this study (refer to Chapter 3 for more information on the selected dataset), the fine-tuning approach considered here falls under inductive transfer learning. Inductive transfer learning corresponds to a setting in which labelled target-domain-data are available to learn a target predictive model while using knowledge from a related source domain/task (Pan and Yang, 2010). Fine-tuning is an inductive TL approach that starts with pre-trained parameters and continues training with gradient updates on a downstream task using labelled data (Devlin et al., 2019). In addition, the approaches discussed fall under parameter transfer, where knowledge is encoded in shared model parameters (or priors) transferred from the source and adapted using target-domain supervision (Pan and Yang, 2010). Since parameter-update strategies can differ, this study experiments with three alternative fine-tuning protocols. These include full transfer learning (FTL), feature freezing, and gradual unfreezing (GU) fine-tuning.

In FTL, all model parameters are updated during training on the new task. Specifically, it defines the target predictor as

$$p_t = h_t \circ f_L \circ \cdots \circ f_1 \quad (2.26)$$

where p_t denotes the target-domain predictor, $f_1, \dots, f_L$ are the layers of the pretrained feature extractor, L denotes the number of feature-extractor layers, and h_t is the task-specific prediction head for the target domain. The head h_t is randomly initialised, while $f_L \circ \dots \circ f_1$ is initialised from pretrained weights. Here, all parts of p_t are optimised jointly, so that both the target-specific head and the pretrained feature extractor are updated end-to-end to adapt the model to the target data (Peng et al., 2021). This approach works well when the target dataset is large, or where the model has relatively few parameters so that over fitting is common (Yosinski et al., 2014). Otherwise, if the target dataset is small while the model has many parameters, fine-tuning can cause over fitting so that the feature representations are frozen (Yosinski et al., 2014). Feature freezing, also called "selective fine-tuning", is based on the intuition that the lower layers in a neural-network learn general domain-independent features, while the higher layers learn more specific domain dependent features (Nguyen et al., 2022). Feature freezing then retains the target predictor (Equation 2.26) but freezes the pretrained feature extractor $f_L \circ \dots \circ f_1$ and optimises only the task-specific head h_t . Here, the head h_t is randomly initialised, while the weights of $f_L \circ \dots \circ f_1$ remain fixed at the pretrained values throughout training, so adaptation to the target task occurs only through the head (Peng et al., 2021).

Another alternative to FTL is the GU approach. Instead of fine-tuning all layers at once, which risks catastrophic forgetting, GU gradually unfreezes the model layers starting from the last layer since the last layer typically contains the most task-specific knowledge and the least general knowledge (Howard and Ruder, 2018). Thus, GU retains the target predictor $p_t = h_t \circ f_L \circ \dots \circ f_1$, but does not optimise all pretrained layers simultaneously. Instead, following the stage-wise formulation of Yang and Ma (2022), let $\theta^{(i)}$ denote the parameters of layer f_i . Then, at the training stage k , only the subset

$$R^{(k)} = \{\theta^{(i)} : L - k \leq i \leq L\} \quad (2.27)$$

is unfrozen and optimised, together with the task-specific head h_t , where L denotes the total number of pretrained layers in the model. Therefore, progressively lower layers are unfrozen in later stages until a larger portion of the pretrained feature extractor is adapted to the target task (Yang and Ma, 2022).

In addition to the fine-tuning learning approaches discussed, the study also investigates the performance of a source-only transfer learning approach. This involves using only the source dataset to pre-train the model and then using that model to make predictions on the target dataset (without fine-tuning). The advantage of this approach is that it is not dependent on labelled target data. The fine-tuning approaches discussed depend on the assumption that both the source and target data are labelled. This is because the model trained on the source domain is adapted to the target domain by using supervised target domain RUL labels. However, in real-world PdM applications, target labelled data is not always available or in very limited quantities. Furthermore, a source-only transfer learning approach may perform poorly on target data if the domain gap is very large. Therefore,

this study further investigates the use of domain-adaptation approaches that learn the target prediction task (RUL) from labelled source-domain data while using unlabelled target-domain input to encourage domain-invariant feature representations (Ganin et al., 2016).

2.4.3 Domain Adversarial neural-network

The Domain Adversarial neural-network (DANN), proposed by Ganin et al. (2016), is an ML approach specifically designed for domain adaptation. Their method derives from domain adaptation theory, which states that successful transfer occurs when predictions are based on features that cannot be distinguished between the source and target domains. They showed that this adaptive behaviour can be obtained in most neural-network-based models by adding standard layers together with a Gradient Reversal Layer (GRL). Figure 2.13 shows the general architecture of DANN.

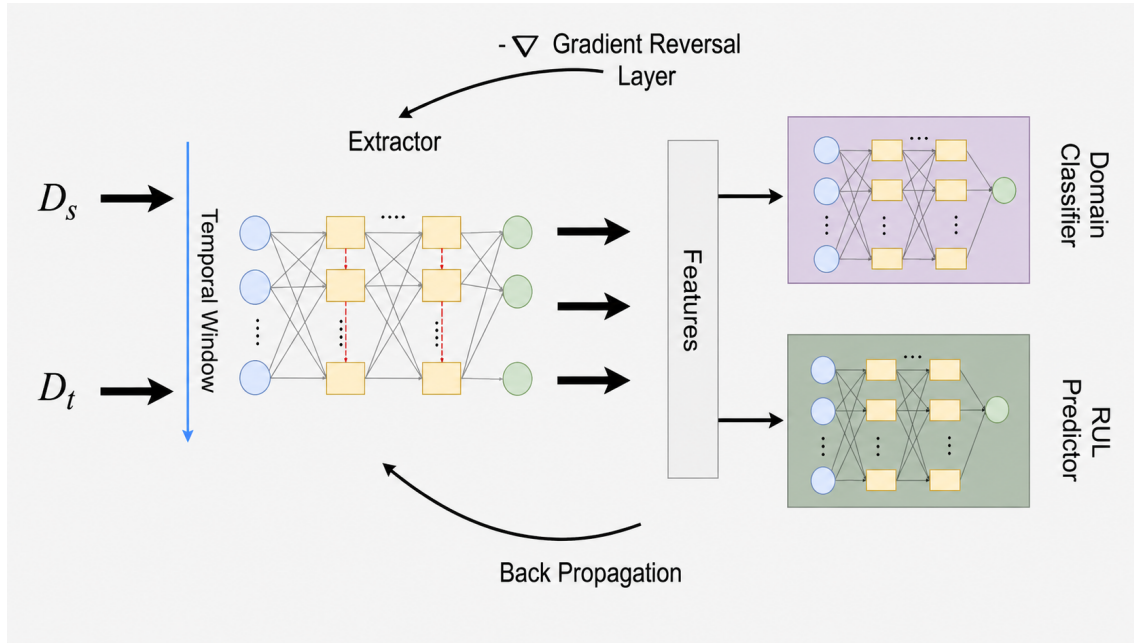


Figure 2.13: Architecture of DANN, adapted from Da Costa et al. (2020)

DANN includes a feature extractor and a feed-forward predictor. The domain classifier that performs the unsupervised domain adaptation then connects to the feature extractor using the GRL. This layer is used to multiply the gradients of the backpropagation process with a negative constant so that the backpropagation process can be performed in an ordinary way for all samples except for the source examples. For the source examples, it uses the standard backpropagation process to minimise the label prediction loss and the domain classification loss. Through this process, the domain distributions will be made similar to each other (Ganin et al., 2016).

As stated above, during training, the neural-network will have portions that are optimised based upon the labels provided for the source dataset and portions that will create similar representations despite having totally distinct datasets. These goals are achieved by defining an adversarial objective (Sicilia et al., 2023). Da Costa et al. (2020) used DANN for RUL prediction by combining a source-domain regression loss with an adversarial domain-classification loss. The resulting min-max objective trains the feature extractor and the RUL predictor to minimise task loss while encouraging domain-invariant features, with the domain discriminator optimised adversarially on the same representation. It is this competitive relationship between these two components of DANN that allows it to maintain effectiveness for the purpose of performing tasks while reducing the dependence on the domain (Sicilia et al., 2023; Da Costa et al., 2020).

2.5 Common Limitations and Challenges in PdM

2.5.1 Data Availability and Labelling Challenges

Despite the availability of IoT technologies, which enable the collection and processing of large volumes of data to improve prognostics, İnce and Genc (2023) note that a key challenge is the limited availability of labelled historical data associated with a system's health condition or RUL targets. They further note that even when a system's failure time is known, the onset of failure is often unavailable or not correctly labelled. Fathi et al. (2021) concur that performing PdM is particularly challenging when large datasets do not contain run-to-failure data. Without run-to-failure data, makes supervised RUL learning difficult. Kim et al. (2024) points out that RUL datasets often struggle with data imbalance. This is reportedly due to the fact that most time series samples correspond to healthy states, and very few show failure progression. He et al. (2024) note that, in many practical monitoring systems, it isn't feasible to fit equipment with multiple sensors because of where and how the devices operate. As a result, RUL prediction often has to rely on a single sensor's data, making it a key challenge to extract a sufficient set of degradation features from limited raw measurements to improve the RUL prediction accuracy. Gong et al. (2022) emphasise that datasets should include a sufficiently diverse range of operating states (and enough instances of each) across the full run-to-failure trajectory, or the RUL models may not learn the underlying degradation behaviour sufficiently.

2.5.2 Industrial Equipment Complexity and Variability

Industrial equipment often operates under complex and varying operating conditions which can lead to several challenges for PdM. Wang et al. (2020) emphasise that in practice, industrial assets operate in complex environments where degradation mechanisms often

deviate from generalised assumptions. Therefore, a generalised degradation model can often not be reliably applied to industrial systems without compromising accuracy. In addition, prognostic performance is affected by factors like the number of available sensors and unit-to-unit differences, and consequently, it is difficult to configure an individual ML model on one scenario that generalises well to other scenarios (Cheng et al., 2021). To address domain shift between training and test distributions, studies such as Cheng et al. (2021) highlight that a single model may perform poorly when operating conditions change. Furthermore, several of the studies note that real-world systems typically are not monotonic in degradation but can possess characteristics such as abrupt changes, reversible wear, or differing failure modes (Winkel et al., 2023). Tanwar and Raghavan (2020) add that discontinuities in wear-signal data can further complicate the PdM problem by subsequently increasing prediction error and confuse models trained on steady degradation. Furthermore, Gong et al. (2022) report that RUL models often fail to fully utilise degradation signals from correlated multi-sensor data, and Liu et al. (2024) add that interactions between spatially distributed sensors are often not effectively captured, leading to the loss of key degradation information during feature extraction.

2.5.3 Uncertainty and Prediction Reliability

Although machine learning (ML) algorithms (particularly those based on neural network architectures) have demonstrated superior performance compared with traditional prognostics methods for inferring degradation information, they are widely reported to lack the capability for quantifying predictive uncertainty and reliability (Rivas et al., 2022). He et al. (2024) adds that due to the noise in sensor signals and variability in model parameters, single-point RUL prediction does not often meet maintenance decision making requirements. Tanwar and Raghavan (2020) emphasise that providing only point estimates is a limitation for risk-informed predictive maintenance planning, since uncertainty information is needed beyond a single predicted RUL value.

2.6 Synthesis of Literature

This section synthesises the main findings in the SLR presented in [Appendix A](#). Specifically, it consolidates the findings for data used in literature for RUL prediction, predictive models used, feature extraction, optimisation and hyperparameter tuning, performance evaluation, and future research directions.

2.6.1 Data Used for RUL Prediction

The SLR reported that studies made use of both synthetic data as well as real world industrial equipment. As identified in [section A.5](#) 55% of the studies used synthetic data. Of this it was notably skewed to one 'bench mark' dataset referred to as C-MAPSS which is sensor data related to the simulation of turbofan engines (where 94% of the related studies made use of C-MAPSS). The real world data showed more variety and consisted from bearings, on Mill Etch Tooling, Lithium-Ion Batteries and several other various forms of industrial equipment.

The large number of researchers who have relied upon the C-MAPSS for RUL predictions has motivated its selection for the study as it has provided an industry-wide benchmark. The dataset was subsequently selected for this study. [Chapter 3](#) provides more detail on the C-MAPSS dataset.

2.6.2 Predictive Modelling Approaches for RUL Prediction

The results in [section A.7](#) indicated an notable preference for using neural-network-based approaches based predictive modelling methods for RUL predictions. More than two thirds of all studies relied upon an ANN or RNN. Of the studies utilising ANN models, 71% of them used Dense ANNs. Of those studies employing RNNs, 61% of them used LSTMs. A further 11% of the studies made use of the Gaussian Process Regression (GPR) model, 8% of the studies employed ESNs and another 6% each employed GRUs and Transformers. All other forms of prediction methodology combined represented only 9%. These studies demonstrated that a very high proportion of RUL research has been focused on developing neural-network based models of various types, and in particular, recurrent and dense network architectures.

This prevalence in literature motivated the use of Dense ANN, LSTM, GRU, and Transformer architectures among the neural-network baseline models investigated in this study. However, the SLR also showed that broad comparisons across different modelling families were less common, with studies more frequently comparing variants within a particular model family. In particular, simpler regression-based approaches such as MLR, regularised regression, and polynomial regression were not prominent among the RUL prediction approaches identified in the review. Moreover, tree-based approaches such as RF and XGBoost were also less frequently investigated as primary RUL predictors. This further motivates the investigation of this models alongside the neural-networks.

2.6.3 Feature Extraction

The SLR identified substantial use of feature extraction (FE). Of all identified FE approaches, CNN-based methodologies represented the majority and TCNs and AEs representing additional, but less utilised forms of FE. The majority of the remaining approaches were unique to the studied works. Additionally, the reviewed prediction models used in combination with other models often employed hybrid architectures where the input data underwent some form of feature extraction prior to processing via the RUL predictor. For example, there existed a notable number of combinations employing CNN-based FE with both LSTM-based and Dense-ANN based predictors. The prevalence of CNN, TCN, and AE-based FE in the reviewed literature provided the motivation for investigating these three approaches as feature extractors in the present study.

2.6.4 Optimisation and Hyperparameter Tuning

The SLR showed a strong preference for the Adam optimiser in RUL-related modelling, with Stochastic Gradient Descent (SGD) being the next most commonly applied approach (refer to [section A.8](#)). This indicates that adaptive gradient-based optimisation is widely used for training neural-network-based RUL models. The prevalence of Adam in the reviewed literature supported its use for the neural-network baseline and hybrid feature-extraction models investigated in this study.

A wider range of approaches was identified for hyperparameter tuning. Grid Search (GS) was the most frequently used method, appearing in 40% of the reviewed studies, while Genetic Algorithms (GA) and manual tuning each appeared in 10%. Bayesian optimisation and Random Search were also applied within the reviewed literature (refer to [subsection A.9.3](#)). The frequent use and simplicity of GS supported its use for the regression models investigated in this study, where only a small number of hyperparameters required optimisation. However, its exhaustive nature becomes increasingly computationally expensive as the number of hyperparameters increases. Subsequently, Bayesian optimisation through Optuna was selected for the tree-based and neural-network models with larger hyperparameter search spaces to provide a more computationally practical alternative to exhaustive GS for the models with high-dimension of hyperparameters.

2.6.5 Performance Evaluation

The results in [section A.10](#) indicated that the majority of RUL prediction research focused on performance evaluation in terms of accuracy (or error) based on predictive accuracy. Specifically, out of the 30 studies included in the SLR, accuracy/error-based metrics were used in all but two studies. Of these metrics, Root Mean Squared Error (RMSE) was the

most widely reported metric, and it was referenced in twenty of the thirty studies. The C-MAPSS scoring function was another widely used metric for measuring model performance. On the other hand, there was very little focus placed on assessing the overall performance of RUL models using other performance indicators. For example, only eleven of the studies reported training times, four studies assessed model reliability and fit and generalisation and interpretability was only evaluated in 2 or and fewer studies.

The above-mentioned results were an important factor in developing multiple complementary evaluation methods for assessing the performance of the developed models. As stated earlier, RMSE is used as the primary method for evaluating model prediction error, as well as for model selection. This is because RMSE has been established as a commonly used metric for evaluating prediction error within the RUL field. Additionally, MAE and the C-MAPSS scoring function are used as supplementary metrics to evaluate prediction error and assess the impact of both early and late RUL predictions. The coefficient of determination (R^2) is also used to evaluate model fit. Finally, training times are measured to evaluate model computational efficiency. Furthermore, the SLR indicated that the literature provided limited assessment of model robustness under reduced levels of training data, with only one of the reviewed studies explicitly evaluating performance across several reduced training-data levels. This provided further motivation to systematically evaluate the selected models across progressively smaller training-data scenarios, as well as to develop a metric for assessing prediction accuracy across the RUL lifecycle.

2.6.6 Future Research Directions

The SLR also identified recurring future research directions within the RUL prediction literature. Of the 30 reviewed studies, 26 proposed future work, resulting in 46 identified recommendations. The most frequently proposed directions were the application of models to new domains and subdomain adaptation, each accounting for 13% of the identified recommendations. Model uncertainty and further investigation of hyperparameter tuning each accounted for 9%, while practical application accounted for 7%. Other recurring recommendations included TL, anomaly detection, and equipment-specific investigations (refer to [section A.11](#)).

The recommendations indicate an emphasis on improving the generalisability and practical applicability of RUL prediction approaches. In particular, the prominence of new-domain application, subdomain adaptation, and transfer learning suggests that the ability of models to retain performance across different operating conditions, equipment, and data distributions remains an important research concern. Similarly, the recommendations relating to model uncertainty and practical application indicate a need to assess RUL models beyond predictive accuracy alone and to better establish their reliability under realistic operating conditions.

Chapter 3

Data Set Selection and Analysis

This chapter describes the dataset selected for the experiments conducted to address the research questions outlined in [subsection 1.1.5](#). First, the criteria that guided the selection of the dataset are presented. Next, the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset collection is introduced to discuss the key components and how its variants are defined by different combinations of operating conditions and fault modes. This is followed by an exploratory data analysis of the datasets.

3.1 Dataset Selection Criteria and Motivation

To ensure that the research questions can be answered ([subsection 1.1.5](#)), that the experiments are reproducible, and that the results can be interpreted relative to previous PdM/RUL work, the selected dataset should meet the following requirements:

- Req1** Publicly available and widely used as a benchmark in the literature.
- Req2** Supports supervised RUL learning from run-to-failure trajectories.
- Req3** Contains sufficient data to enable systematic training-size reduction for investigating data scarcity.
- Req4** Provides variants that differ in operating conditions and/or fault modes so that transfer learning can be investigated under significant domain shift between the source and target.

3.1.1 Dataset Identification and Motivation

The NASA Prognostics Centre of Excellence (PCoE) has developed a large collection of benchmark datasets accessible via its data repository (NASA, 2025). The repository spans diverse application areas, including biological cultivation, materials fatigue testing, and accelerated ageing of power-electronics, alongside other instrumented test-beds with multivariate sensor measurements and ground-truth condition indicators. Several of the datasets in this repository fall into specific categories that can be used for RUL prognostic experiments. As an example, the repository milling machine category contains a series of machining trials that are able to fail under different feed rates, speeds, and depths of cut, with tool degradation monitored through the measured VB (flank wear) in the inserts. Additionally, the battery category of the repository contains charge/discharge Li-ion experimental results from trials performed at a variety of temperatures. The degradation of the batteries was monitored by tracking the changes in impedance as a function of temperature. As another example, the repository contains a bearing category consisting of bearing run-to-failure tests, which will allow both condition monitoring and the study of degradation progression, and thus enable the modelling of RUL using vibration and other sensor measurement variables.

The selected dataset for this study is the National Aeronautics and Space Administration (NASA) Prognostics Centre of Excellence (PCoE) Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) turbofan engine dataset collection. C-MAPSS is widely used in PdM/RUL research (refer to the findings in [section A.5](#)) and is commonly treated as an established benchmark for RUL prediction studies (Ramasso and Saxena, 2014; Liu et al., 2025; Hatipoğlu and Yilmaz, 2025). Additionally, C-MAPSS was selected because it satisfies the requirement to support supervised RUL learning from run-to-failure degradation trajectories. In particular, the publication of the C-MAPSS data set notes that, in the training set, faults increase in magnitude until the system fails and that the test set is accompanied by a vector of RUL values (Saxena et al., 2008a). This subsequently enables an objective evaluation of the RUL prediction models. Furthermore, C-MAPSS is organised into numerous independent engine trajectories and includes significant trajectory counts for each subset (Mao et al., 2022). This makes controlled training-size reduction experiments possible by subsampling at a per-engine/trajectory level. In addition, C-MAPSS provides multiple dataset variants with different operating conditions and fault modes (refer to [section 3.2](#)) which enables TL-based experiments under significant domain shifts between source and target subsets.

3.2 Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)

C-MAPSS is a simulation software used by NASA PCoE to simulate large turbofan engines. MATLAB and Simulink (MathWorks) were used to create the environment. The simulation framework has several adjustable input parameters that allow for the input of varying values for (among others) the operational profile, closed-loop controllers, and environmental conditions (Saxena et al., 2008b). Figure 3.1 shows the main components of a turbofan engine.

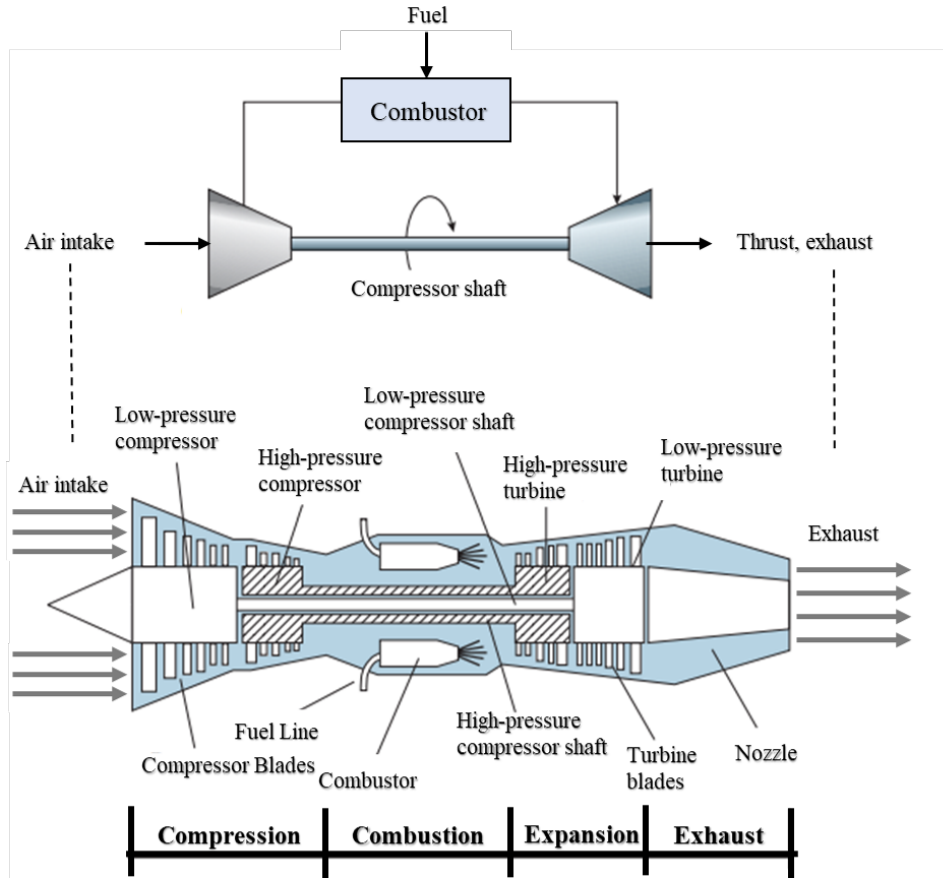


Figure 3.1: Simplified cross-sectional view of the internal structure and components of a typical commercial aircraft engine, adapted from Wickert and Lewis (2017).

A turbofan is built similarly to a turbojet engine around the core (where air is compressed, mixed with fuel, and combusted, then expanded through a rear nozzle to produce thrust) (Wickert and Lewis, 2017). However, a turbofan engine also incorporates large fan blades driven by the engine so that thrust is produced by both the exhaust jet and the fan. As seen in Figure 3.1, the core consists of multiple compressor stages driven by a shaft connected to the turbine along the centreline of the engine. The fuel–air mixture burns in

the combustion chamber, and the resulting high-pressure, high-temperature gas expands through the turbine, producing enough power to drive the compressor, with the remaining energy contributing to the thrust (Wickert and Lewis, 2017).

C-MAPSS simulates a turbofan engine plant with an integrated control system consisting of a fan speed controller and limit regulators that enforce operating constraints. The simulator provides 14 inputs, including fuel flow and 13 health-parameter inputs, which enable the simulation of fault and deterioration effects across the engine's five rotating components (the fan, low-pressure compressor, high-pressure compressor, high-pressure turbine, and low-pressure turbine) (Saxena et al., 2008b). Table 3.1 summarises these C-MAPSS inputs used to represent degradation scenarios for the simulated engine components.

Table 3.1: C-MAPSS inputs used to simulate degradation scenarios across rotating engine components, adapted from Saxena et al. (2008b).

Input (degradation modifier / control)
Fuel flow
Fan efficiency modifier
Fan flow modifier
Fan pressure-ratio modifier
Low-Pressure Compressor efficiency modifier
Low-Pressure Compressor flow modifier
Low-Pressure Compressor pressure-ratio modifier
High-Pressure Compressor efficiency modifier
High-Pressure Compressor flow modifier
High-Pressure Compressor pressure-ratio modifier
High-Pressure Turbine efficiency modifier
High-Pressure Turbine flow modifier
Low-Pressure Turbine efficiency modifier
High-Pressure Turbine flow modifier

The outputs that measure the system's response are comprised of 21 sensors (refer to Table 3.2). These sensor readings measure key engine operating conditions, including temperature, pressure, and rotational speed, across multiple internal components.

Table 3.2: C-MAPSS sensor parameters provided to participants.

Symbol	Description (Units)
s_1	Total temperature at fan inlet ($^{\circ}\text{R}$)
s_2	Total temperature at LPC outlet ($^{\circ}\text{R}$)
s_3	Total temperature at HPC outlet ($^{\circ}\text{R}$)
s_4	Total temperature at LPT outlet ($^{\circ}\text{R}$)
s_5	Pressure at fan inlet (psia)
s_6	Total pressure in bypass duct (psia)
s_7	Total pressure at HPC outlet (psia)
s_8	Physical fan speed (rpm)
s_9	Physical core speed (rpm)
s_{10}	Engine pressure ratio, P_{50}/P_2 (–)
s_{11}	Static pressure at HPC outlet (psia)
s_{12}	Ratio of fuel flow to P_{s30} (pps/psi)
s_{13}	Corrected fan speed (rpm)
s_{14}	Corrected core speed (rpm)
s_{15}	Bypass ratio
s_{16}	Burner fuel-air ratio
s_{17}	Bleed enthalpy
s_{18}	Demanded fan speed (rpm)
s_{19}	Demanded corrected fan speed (rpm)
s_{20}	HPT coolant bleed (lbm/s)
s_{21}	LPT coolant bleed (lbm/s)

The C-MAPSS collection has four publicly available datasets, each operating under different operating conditions and fault modes. [Table 3.3](#) shows the high-level difference between these datasets.

Table 3.3: Overview of C-MAPSS sub-datasets and experimental setup.

Dataset	Operating conditions	Fault modes	Train engines	Test engines
FD001	1	1	100	100
FD002	6	1	260	259
FD003	1	2	100	100
FD004	6	2	249	248

The most significant differences between these datasets relate to the number of faults that occur simultaneously and the number of operating conditions simulated. FD001 and FD003 both operate under a single operating condition, representing one combination of altitude, Mach number, and throttle settings, while FD002 and FD004 simulate engines under six distinct operating conditions. The Operational settings are presented in [Table 3.4](#) below.

Table 3.4: Operational settings in the C-MAPSS dataset.

Operational Setting	Notation	Physical Quantity
Operational Setting 1	os ₁	Altitude
Operational Setting 2	os ₂	Mach Number
Operational Setting 3	os ₃	Throttle Resolver Angle (TRA)

Each of the datasets has both a training dataset that can be used to develop models and a test dataset used to evaluate them. For the scope of this study, FD001 is used to conduct cross-model evaluations across different training-data fractions. Additionally, FD001 is used as the target dataset for transfer learning, with FD002 and FD003 serving as the source datasets. [Section 3.3](#) provides a more detailed view of the dataset through an exploratory data analysis of FD001. For a comparison of feature distributions across FD001, FD002, and FD003 to examine the potential domain shift, refer to [subsection 5.4.1](#). For more detailed information on the simulator and experimental setup used to generate C-MAPSS, the reader is referred to the publication by Saxena et al. ([2008b](#)).

3.3 Data Preparation and Exploratory Data Analysis.

This section discusses how the data was prepared to be used in the experiments. Moreover, it presents an exploratory data analysis (EDA) by visualising the maximum cycles (representing the start of each engine’s life), sensor feature trends, feature distributions, and the correlation between features and the target RUL. The raw data are stored in text files divided into training data, which consists of a matrix containing all sensor feature values and sequential life cycles, as well as the labelled RUL set.

Unlike the training trajectories, the C-MAPSS test trajectories are truncated before EOL. The accompanying RUL file provides one true RUL value for each test engine, corresponding to the final observed cycle of that engine (at any random point in time of the observed engine). To obtain a true RUL value for each preceding observed cycle, the RUL trajectory was reconstructed as

$$RUL_{i,j}^{\text{test}} = RUL_{i,\text{final}} + (C_{i,\text{max}}^{\text{test}} - C_{i,j}), \quad (3.1)$$

where $RUL_{i,\text{final}}$ is the RUL supplied by C-MAPSS for the final observed cycle of test engine i , $C_{i,\text{max}}^{\text{test}}$ is the final observed cycle of that engine, and $C_{i,j}$ is its cycle at index j . As with the training labels, the reconstructed test RUL values were subsequently capped at 125 cycles.

The raw C-MAPSS data are provided as separate training and test datasets containing the engine-cycle observations and corresponding sensor measurements. The procedures used to prepare these data for the ML models are summarised in [Figure 3.2](#).

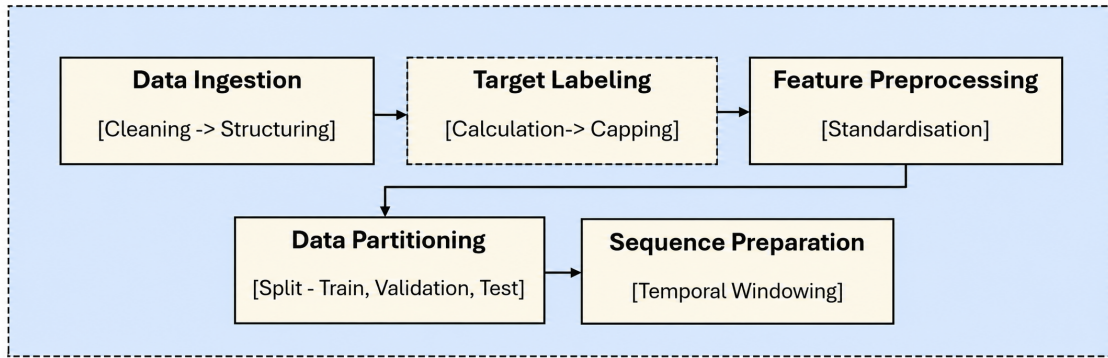


Figure 3.2: Preprocessing Steps for the C-MAPSS dataset.

Firstly, the data are read into usable data frames (2-dimensional mutable data structure), and the features are then labelled according to the convention shown in [Table 3.2](#). The training dataset does not contain the RUL labels for the engines; however, it does contain the sequential cycles until complete degradation has been achieved. Therefore, the RUL can be calculated through [Equation 3.2](#) and since the training set is full run-to-degradation data (the last point of the trajectory is defined as the EOL), it can be calculated through [Equation 3.3](#):

$$RUL_{ij} = \begin{cases} C_{i,\text{EOL}} - C_{i,j}, & C_{i,j} \leq C_{i,\text{EOL}}, \\ 0, & C_{i,j} > C_{i,\text{EOL}}. \end{cases} \quad (3.2)$$

$$RUL_{ij} = C_{i,\text{max}} - C_{i,j} \quad (3.3)$$

where RUL_{ij} is the RUL of the engine i at the cycle index j , $C_{i,\text{EOL}}$ EOL cycle threshold for the engine i ; used to define the point at which $RUL_{ij} = 0$, $C_{i,j}$ represents the time

step (cycle) of the engine i at the index j and, $C_{i,\max}$ represents the maximum observed cycle count (final) for the engine i on the run-to-failure trajectory.

RUL values early in an engine's operational life can be arbitrarily large. During this initial healthy operating stage, degradation is limited, and the system may exhibit little degradation-related information from which to reliably distinguish between high RUL values (Zhang et al., 2018; Zhao et al., 2020). Zhang et al. (Zhang et al., 2018) further note that estimating RUL before the onset of observable degradation has limited practical value. Therefore, piecewise-linear RUL representations are commonly used, in which the RUL is held at a maximum value during the healthy operating stage and decreases once degradation becomes evident (Zhang et al., 2018; Zhao et al., 2020). Following Jin et al. (2022), a piecewise linear representation is used in this study, with the maximum RUL capped at 125 cycles. Therefore, all training samples that have RUL values over 125 cycles are set with target labels equal to 125. This subsequently clips the labels of 30.3% of the FD001 training data which operate in a healthy state. To avoid data leakage, the effects on the number of RUL labels on the true labels on the test set are only discussed in the results chapter.

To avoid data leakage, the effects on the number of RUL labels on the true labels on the test set are only discussed in the results chapter. The training dataset consists of 100 engine unit degradation trajectories, which make up a total of 20,631 operational cycles for all engines. Figure 3.3 shows the maximum cycle for each engine.

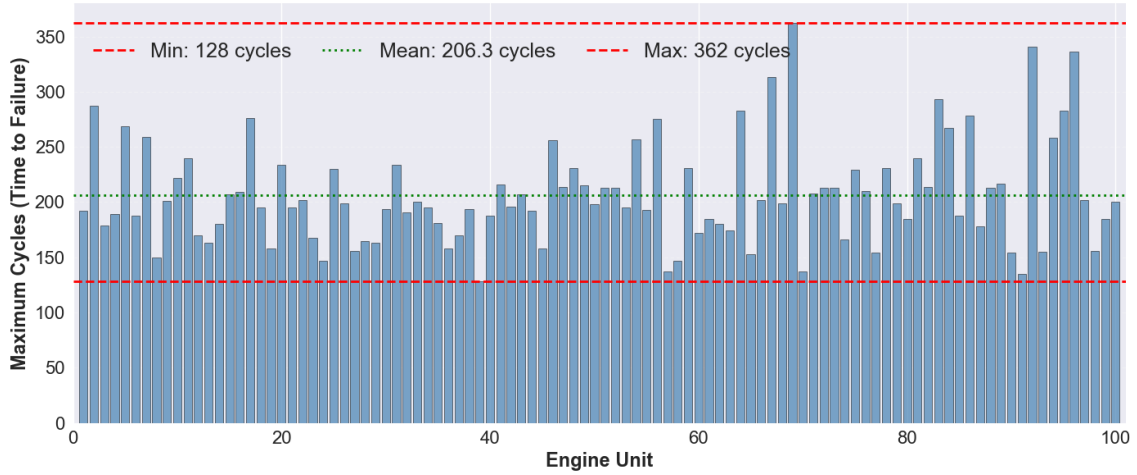


Figure 3.3: Maximum operational cycles per engine unit in the FD001 training dataset.

Figure 3.3 shows substantial variability in engine lifetimes, with maximum operational cycles ranging from approximately 100 to 350 cycles across the 100 engine units. The mean operational lifetime is 206.3 cycles.

The features are provided in a predefined order without column headers. For conven-

tion, sensors are denoted as s_n , where $n \in \{1, 2, \dots, 21\}$ denotes the n -th sensor feature, as shown in Table 3.3. Moreover, the operating settings are denoted as os_i (where $i \in \{1, 2, 3\}$) as shown in Table 3.4. Specifically, FD001 operates under a single operating condition representing a combination of altitude os_1 , Mach number os_2 , and throttle settings os_3 . The evolution of four representative sensor evolutions across their RUL is shown in Figure 3.4. To view all the sensors readings over RUL, refer to Appendix Section Figure B.1

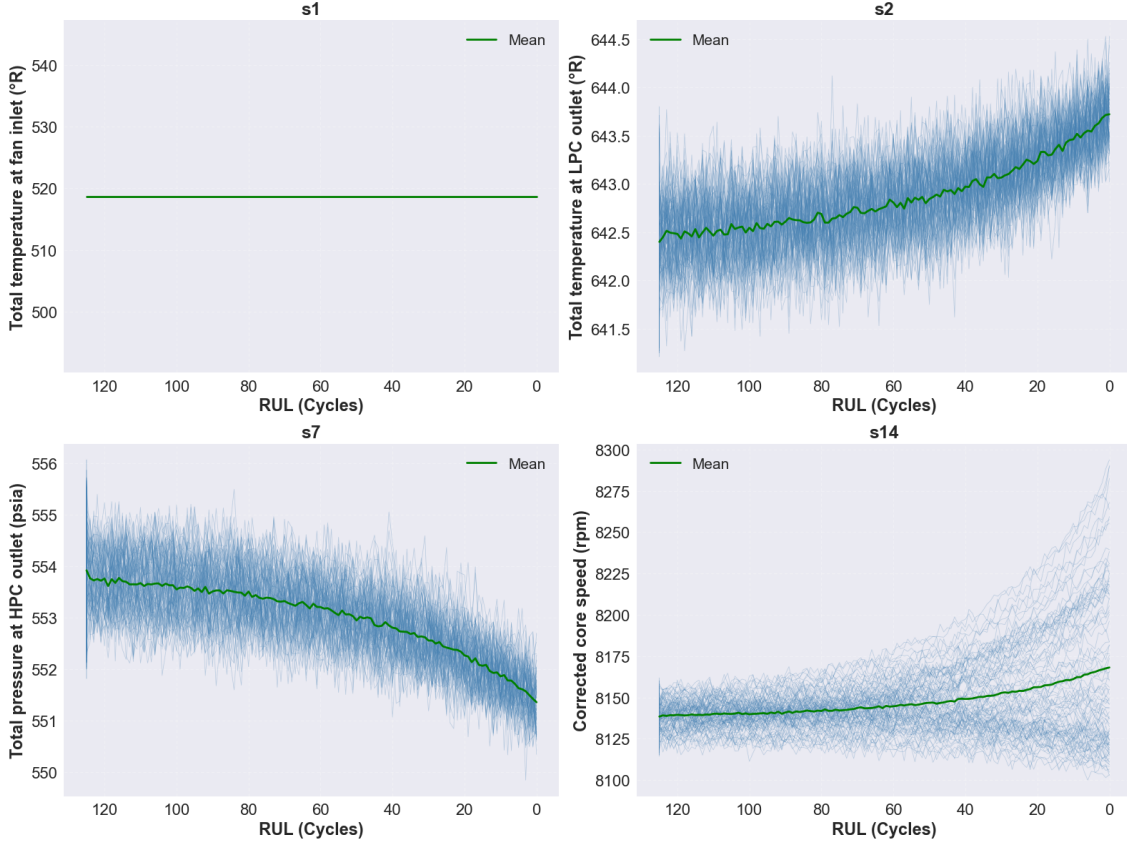


Figure 3.4: Evolution of four representative sensors (s_1 , s_2 , s_7 , s_{14}) over RUL, illustrating distinct degradation patterns.

Four degradation trends are observed across the sensors. Sensors s_1 , s_5 , s_6 , s_{10} , s_{16} , s_{18} and s_{19} show constant trends throughout their useful life. Therefore, these sensors will not provide degradation-related information and are unlikely to contribute to the prediction of the RUL. A second group of sensors s_2 , s_3 , s_4 , s_8 , s_{11} , s_{13} , s_{15} and s_{17} demonstrate a negative correlation with RUL, showing increasing values as RUL decreases (i.e., as the engine approaches failure). In contrast, sensors s_7 , s_{12} , s_{20} , and s_{21} show a positive correlation with RUL, showing decreasing values as RUL decreases. sensors s_9 and s_{14} show high variability and increasing spread as RUL approaches zero. This indicates a weaker correlation with RUL and potentially provides less information for degradation prediction compared to the more strongly correlated sensors. Figure 3.5 visualises the

correlation of these features through a correlation matrix.



Figure 3.5: Correlation matrix of C-MAPSS sensors and RUL for the FD001 training dataset.

It is observed that s_{11} (the HPC outlet) shows the strongest negative correlation with RUL (-0.78). s_{12} (Ratio of fuel flow to Ps_{30}) shows the strongest positive correlation with RUL (0.75). Several other sensors also display strong correlations with RUL, indicating that these sensors may contain useful degradation-related information. In contrast, sensors such as s_6 (Total pressure in bypass duct) and the constant features show little to no correlation with RUL and are therefore less likely to contribute useful degradation information.

To interpret the correlations relative to RUL it should be noted that C-MAPSS represents component deterioration through changes in component flow capacity and efficiency, which subsequently alter the thermodynamic behaviour of the engine system Saxena et al. (2008a). Subsequently, degradation can influence the measured quantities in the engine including, including compressor pressures (s_{s7} and s_{s11}), turbine temperatures (s_{s4}), fuel-related quantities (s_{s12}), bypass behaviour (s_{s15}), and turbine cooling flows (s_{s20} and s_{s21}).

The directions observed in Figure 3.5 are consistent with degradation trends previously reported for C-MAPSS ((Hong et al., 2020)). Specifically, the static pressure at the HPC outlet (s_{s11}), LPT outlet temperature (s_{s4}), and bypass ratio (s_{s15}) increase as degradation progresses, whereas total pressure at the HPC outlet (s_{s7}), the ratio of fuel flow to HPC outlet static pressure (s_{s12}), HPT coolant bleed (s_{s20}), and LPT coolant bleed (s_{s21}) decrease. Since RUL decreases as degradation progresses, sensors that increase towards failure will show negative correlations with RUL. On the other hand sensors that decrease towards failure shows positive correlations.

3.3.1 Data Preprocessing

As can be seen by Figure B.1, each sensor reading has varying levels of minimum and maximum ranges. To more easily visualise this, Figure 3.6 shows the distributions of the 4 representative features (refer to Figure B.2 to see all 21 sensor distributions).

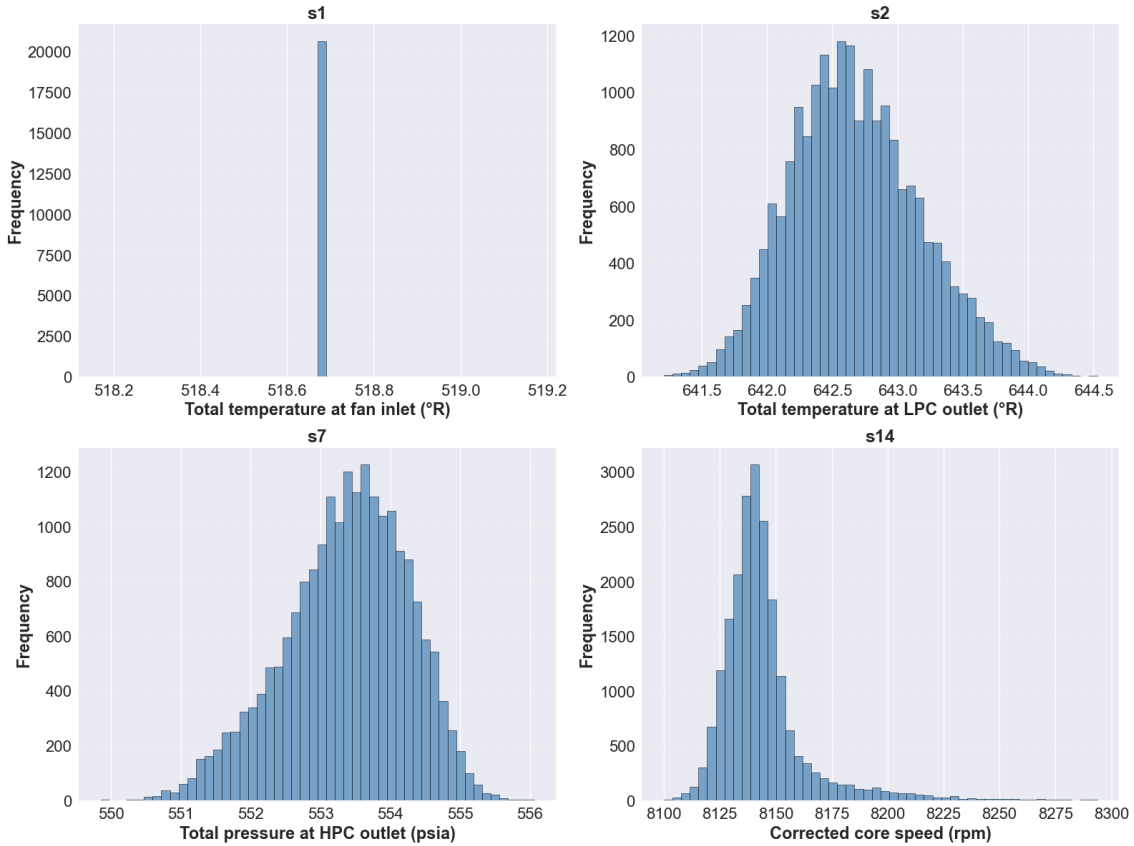


Figure 3.6: Distribution of the C-MAPSS FD001 Representative Sensors.

As can be observed, the ranges and magnitudes of these features can differ significantly. For example, s_7 records the total pressure at the HOC outlet (measured in psia) with readings ranging from 550 to 556 psia. In contrast, s_{14} records the corrected core speed,

measured in rotations per minute (rpm), which operates on a completely different scale, ranging approximately from 8,100 to 8,250 rpm. It should also be noted that sensors such as s_1 is constant in the FD001 dataset which is why there is no observed variance.

These differences in sensor units and magnitudes require the need for standardisation (Li et al., 2024). This is because when a feature has a variance and magnitude that are much larger than others, it can dominate the objective function, making it difficult for ML models to learn (scikit-learn, 2025). Standardisation was applied using the Standard Scaler function from the scikit-learn library. For the constant feature standardisation was still applied, however, since the feature has zero variance, the transformation only centres its constant value at zero and does not alter its variability.

For reference, the adapted formula is presented in Equation 3.4.

$$s_{k,ij}^{\text{std}} = \frac{s_{k,ij} - \mu_k}{\sigma_k} \quad (3.4)$$

where $s_{k,ij}$ is the sensor value k for the engine i in cycle j , $s_{k,ij}^{\text{std}}$ is the standardised sensor value, μ_k is the mean of the sensor k and σ_k is the SD of the sensor k

It should be noted that, to ensure that there is no data leakage, the standardisation parameters (μ_k and σ_k) were calculated from the training set only and applied to both training and test sets (Apicella et al., 2025).

After the data was standardised, the training set was then split into a training and validation set using 80/20 hold-out splits, respectively. The validation set will be used for model selection and hyperparameter tuning, while the training set is used for the actual model training responsible for weight updates during optimisation (Cho et al., 2024). To implement this, the scikit-learn `train_test_split` method was used with the `random_state` set to seed 42. It should be noted that Including observations on the same subject in both the training and test sets results in data leakage (Brookshire et al., 2024). Therefore, when splitting the training and validation set, the same principle is applied, and thus the implementation is done at the engine (subject) level as opposed to a time unit (cycle) level.

Since the engine degradation data are cyclic, an additional preprocessing step is required, namely temporal windowing of the data. This is specifically necessary for the sequential models that will be trained on the data. This method, also known as the sliding window, is necessary to construct sample data that meet the vector input requirement of sequential models such as LSTM (Yang et al., 2022). To implement this step, an overlapping window of 30 consecutive cycles with a stride of 1 was used. Each window captures sensor measurements over the temporal range, and the final time step serves as the target label. For the test set, each sliding window was therefore assigned the reconstructed true RUL

value corresponding to the final cycle of that window. The window size of 30 cycles was selected based on practices in previous research (Shi et al., 2023; Ensarioğlu et al., 2023) and is maintained as a fixed hyperparameter.

Chapter 4

Overview of the Methodology

This chapter presents the experimental methodology used to perform the RUL prediction experiments. The chapter discusses data preparation, baseline architecture development, and implementation of a four-phase experiment. The four phases are designed to progressively explore the RUL prediction performance under data limitations. The first three phases involve supervised learning models. In phase 1, 17 models from 3 different model families (classical regression, tree-based, and neural-network-based) are evaluated. Then, FS techniques are used to lower the dimensions of the training sets, and representative baseline models are trained on those sets. In phase 3, FE is then implemented on the selected feature sets. The final phase involves several learning techniques. Specifically, supervised fine-tuning techniques are evaluated. In addition, source-only transfer learning is investigated, as well as the unsupervised domain adaptation strategy DANN. All models are trained using the varying training samples with a constant validation set for evaluation. [Figure 4.1](#) presents an overview of the experimental design workflow.

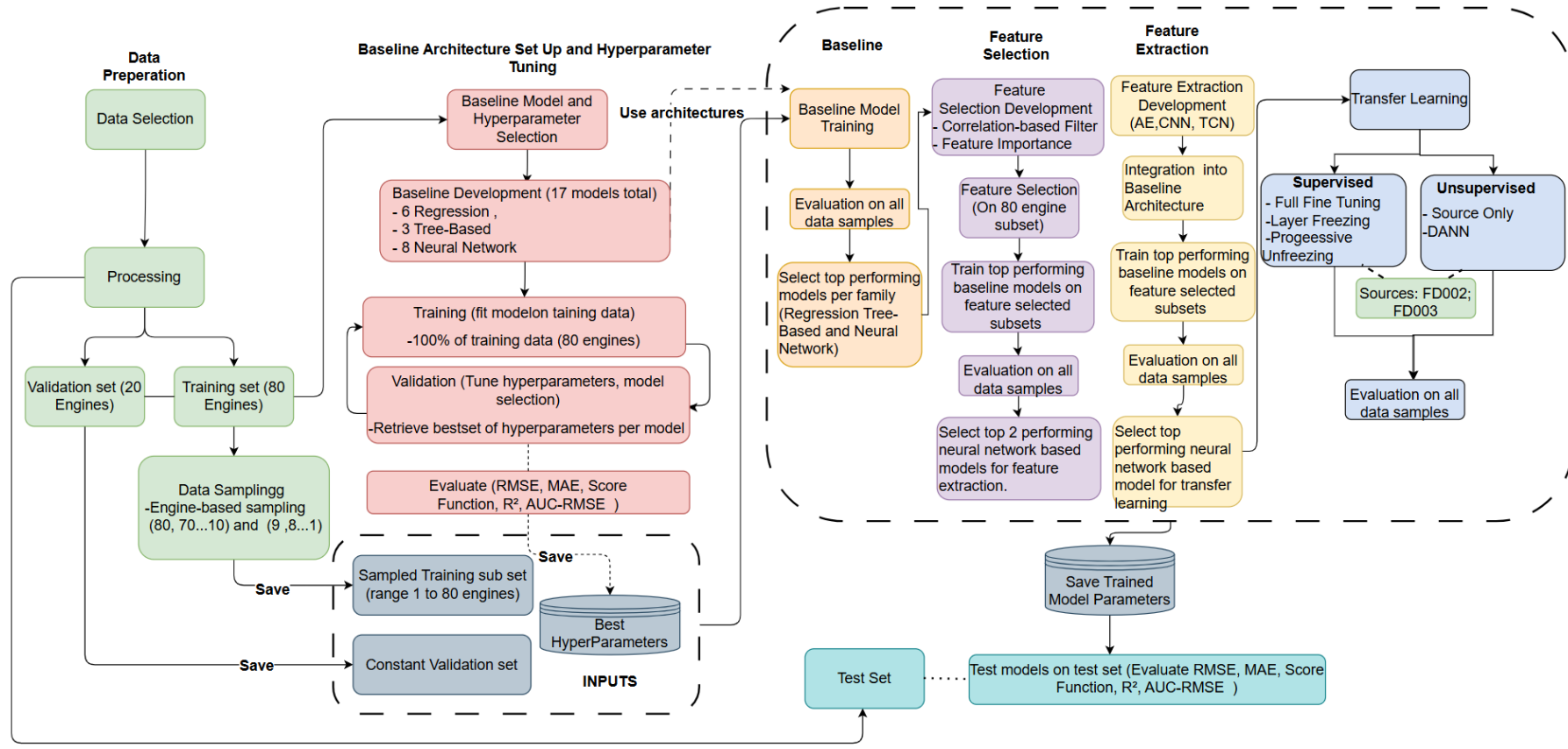


Figure 4.1: Overview of the experimental design.

4.1 Data Preparation

The first stage is data preparation. This includes the actual identification of a dataset that meets the criteria to achieve the research objectives. As discussed and motivated in [subsection 3.1.1](#) the C-MAPSS dataset has been identified as an appropriate dataset to be used for research. For a detailed overview of the data set, refer to [Chapter 3](#). The chapter also provides the steps used to prepare the data, including structuring, labelling, standardisation, temporal windowing, and splitting.

After the data-splitting step, a validation and training set are retrieved. The validation set specifically contains the degradation trajectories of 20 engines (3,490 observations), and the training set contains 80 (14,241 total observations). These trajectories have complete degradation (until the EOL is reached). However, the test set is provided separately. This set is put aside and will only be used after all models have been trained, validated, and selected. To prevent data leakage, no exploratory data analysis is done directly on the test set (to simulate the reality in which these data have never been seen prior to development). However, [section 3.3](#) provides an EDA of the training dataset; in addition, the metadata from the data repository (NASA, [2025](#)) where the test set was retrieved specifies that the test set consists of 100 other engine units that only experience partial trajectories (do not necessarily reach EOL). This simulates a real-world operation in which the currently observed data are previously unseen and the degradation trajectory constitutes a stochastic realisation within the unit's life-cycle process.

After preprocessing, the training set is incrementally reduced to allow systematic testing of the performance of the models under limited training data. Specifically, training is conducted at multiple engine counts differing by 10 units (80, 70, 60, ... , 10 engines), and to focus on limited-data scenarios, additional counts differing by one engine (9, 8, 7, ... , 1 engines).

The data subsets were selected at the engine-unit level using a fixed random seed of 42. The same randomly generated engine ordering was used for models. Moreover, the subsets are nested such that each smaller subset was contained within the preceding larger subset. For example the 9-engine subset was contained within the 10-engine subset. This is done to ensure that there is a controlled progression in which the amount of available training data scenarios was reduced. For reproducibility, the source code used for preprocessing can be accessed on GitHub (Maré, [2026](#)).

It is acknowledged that, at very small engine counts, the observed performance may depend on the characteristics of the particular engine units retained. Since only one nested sequence of engine subsets was evaluated, the variability associated with alternative engine selections was not quantified. Repeating experiments across multiple independently sampled engine subsets therefore is proposed for future work.

For the supervised experiments, namely the baseline, FS, FE, and fine-tuning transfer learning approaches, the training data were reduced together with the corresponding RUL labels. Therefore, each reduced training condition still contains both the input sequences and their associated target labels. The corresponding number of training observations available at each engine-count condition differs depending on whether the model uses the non-recurrent representation or the recurrent windowed representation. These counts are summarised in [Table 4.1](#).

Table 4.1: Number of observations available at each engine-count condition.

Engines	Non-windowed observations (for non-recurrent models)	Windowed observations (for recurrent models)
80	16,561	14,241
70	14,434	12,404
60	12,304	10,564
50	10,558	9,108
40	8,666	7,506
30	6,511	5,641
20	4,415	3,835
10	2,182	1,892
9	1,906	1,645
8	1,647	1,415
7	1,493	1,290
6	1,286	1,112
5	1,092	947
4	862	746
3	666	579
2	503	445
1	216	187
20 (Validation)	4,070	3,490
100 (Test)	13,096	10,196

The original test data (FD001) for the final model evaluation remained the same as the other training data scenarios. The non-windowed representation is evaluated based on 13,096 individual cycle observations. The sequential representation was evaluated on 10,196 overlapping 30-cycle observation windows with a stride of one. This means that there is one prediction per test window for the last predicted RUL. As such, all 100 trajec-

tories from the test engine were included in the two representations. Therefore, while the non-windowed and sequential representations used the same test engines as inputs, they had different number of observations due to their input representations.

For the unsupervised domain adaptation experiment using DANN, the target-domain-data is also reduced according to the same engine-count conditions. However, the corresponding target-domain labels are withheld during training and are only used for validation and final evaluation. Therefore, the DANN implementation reflects the unsupervised domain adaptation scenario, where labelled source-domain data are available, but target-domain adaptation is performed using unlabelled target-domain input.

To ensure fair comparison, every condition (80, 70...1 engines for training) is evaluated on a fixed validation set that consists of 20 engines (where each engine is observed through sequential life cycles until EOL). Therefore, the differences in validation metrics and their relationship to the reduction in training data are more interpretable since the validation set serves as a control.

4.2 Baseline Architecture and Hyperparameter Tuning

For the scope of this study, 17 algorithms from three different families (regression, tree-based, and deep learning) are selected to train on the data. For regression, MLR, Ridge, Lasso, ElasticNet, and polynomial (2nd and 3rd degrees). RF, XGB, and LGBM are selected to represent the tree-based family. Lastly, for the deep learning algorithms, namely ANN, RNN, LSTM, BiLSTM, GRU, CNN, TCN, and Transformer.

Before the phased experiments are executed, a baseline configuration phase is conducted. The goal of this phase is to determine a single fixed architecture for each algorithm (17 in total). These architectures are then kept constant for all subsequent conditions (i.e., across the different training data size ranges). In this way, the hyperparameters and model choice function as controls so that any performance differences can be interpreted more clearly as arising from the amount of training data and the type of algorithm. Thus, this phase sets a baseline set of architectures and hyperparameters that can be applied in later experimental phases and used as a baseline for evaluation.

The hyperparameter search was carried out using the GS or the Optuna optimisation framework. For all baseline models, 100% of the training data (80 engines) together with the same fixed validation set of 20 engines was used to evaluate the hyperparameter configurations, with validation RMSE used consistently as the model-selection criterion. To save computational costs and allocate more resources to the more promising combinations of hyperparameters, pruning was used (May et al., 2022). Since each algorithm differs in architecture, the search spaces were defined independently on each. For details of the

implementation of each model, see [subsection 5.1.1](#).

The output of the baseline architecture set-up, for each of the 17 algorithms, is a selected architecture baseline and one set of hyperparameters. These hyperparameters are selected using the full 80-engine training condition and are subsequently reused without retuning across the reduced training-data conditions. This was done deliberately to keep model configuration fixed so that changes in performance across the engine-count scenarios could be interpreted primarily in relation to the amount of available training data. However, subsequently it should be acknowledged that this also means that the selected hyperparameters are not necessarily optimal for each lower-data condition. These configurations are used and remain unchanged for the baseline and FS phases.

4.3 Feature Selection

The second experimental phase is FS. Its purpose is to evaluate whether reducing the dimension of the dataset will affect the performance of predicting degradation as training data becomes more scarce. The models used in this phase are a subset of the baseline models validated in the Baseline phase. For the scope of this study, a ranking procedure is used to select the best representative baseline models to be used for the FS phase.

Two FS approaches were applied. Firstly, a correlation-based filter method was used. This approach involves assessing the Pearson correlation between the features and the target (RUL). A threshold is then used to filter out all features whose correlation falls below that value. The second FS approach used is RF importance. Similarly for this approach, a threshold importance value is defined, and all features with an importance score below this threshold for the prediction of RUL are removed.

FS is performed on the full training dataset (80 engines) so that a single feature set per method is constant. The same baseline model architectures developed in [section 4.2](#) were subjected to the feature set.

4.4 Feature Extraction

The FE phase focuses on implementing different FE layers into the existing baseline deep-learning model architectures. The FE layers include AE, CNN, and TCN, three widely used algorithms for FE (refer to [section A.6](#)). The models in this phase subsequently have hybrid architectures in which AE, CNN, and TCN form the extractor layer and the baseline serves as the predictor. Since these models have a wider hyperparameter search space (extractor and predictor), the model is trained only on the dataset selected with the highest performing features. Moreover, the hyperparameter architecture of the baseline

model is conserved, whereas the extractor layer’s hyperparameters are tuned in this phase. Once tuned, the extractor hyperparameters are fixed, and the model is trained across the same engine counts (80, 70, ... 1) and evaluated with the fixed validation set.

4.5 Transfer Learning

The fourth experimental phase investigates whether TL can improve the RUL prediction performance on the FD001 dataset when the training data are limited. As discussed in [section 2.4](#), TL can be approached in several different ways. This study specifically investigates 5 different TL approaches. This includes three supervised fine-tuning approaches, a source-only transfer approach, and DANN training.

4.5.1 Supervised Transfer Learning

All three supervised fine-tuning techniques require that the target training dataset (FD001) has labelled RUL values. Among the three approaches, the first is full fine-tuning, where the entire pretrained model (on source data FD002 or FD003) is fine-tuned on the target dataset. The second approach involves an adaptive layer fine-tuning approach, where all layers are frozen after training, and only the output (dense layer) is trained on the target data. The third approach involves pretraining both the hidden and output layers on the source data set and then progressively unfreezing them to adapt to the target.

4.5.2 Unsupervised Domain Adaptation

Since real-world PdM scenarios do not always have labelled training data available, two additional transfer learning techniques are explored. This includes source-only transfer learning, which simply involves pretraining the model on the source dataset and then performing the inference on the target dataset. This approach relies on the assumption that the domains of the source and target datasets are similar since there are no adaptation layers.

Therefore, the DANN architecture is also implemented because it is designed to adapt to the target domain through unsupervised adversarial training. To implement DANN, this study follows the architecture proposed by Da Costa et al. (2020). In addition, the neural-network architecture selected during the validation phase is also implemented within the DANN framework to assess whether it improves performance relative to the LSTM-DANN architecture used in the original study. The hyperparameters for the DANN architecture are discussed in the following Implementation and Validation chapter ([Chapter 5](#))

4.5.3 Model Validation and Hyperparameter Tuning

A key step in selecting a model is the validation phase. Validation is the process of assessing whether the model reliably captures the relationship between measured data and predictive variables (Lopez et al., 2023). Lopez et al. (2023) notes that a common oversight in ML pipelines is that not enough importance is given to the "crucial stage of validation". They further emphasise that it can lead to models that appear promising but are actually the result of poorly designed cross-validation procedures.

Cross-validated predictive performance is frequently used to estimate or adjust the auxiliary parameters of ML models, often referred to as hyperparameters (Yates et al., 2023). In contrast to the model parameters, which are directly derived from the data during the training process, hyperparameters are parameters that are set manually outside of a learning algorithm (Pfob et al., 2022; Okuno et al., 2021). Unlike model parameters, which are directly learnt from the data during training, hyperparameters are set manually and control the learning process externally before training.

There are a variety of approaches to use to optimise these parameters through a process commonly referred to as hyperparameter tuning. As identified in the SLR the most common approach is the exhaustive GS method [section A.9](#). The benefit of using this approach is that it is simple in implementation and explores every possible combination of the hyperparameters. This is a practical fit for the models investigated in this study with a small set of hyperparameters, and was subsequently selected as the tuner for the regression models. However, the key limitation to this approach is that models with a high range of hyperparameters will take impractically long to exhaustively search each combination, and therefore a more efficient strategy is required.

As shown in [A.13](#), common alternative approaches for PdM include the GA ([subsection A.9.2](#)), BOA ([subsection A.9.3](#)), random search ([subsection A.9.4](#)) and manual tuning. The SLR discusses the application of each of these approaches. For the overview of theory, recall [section 2.3](#). As motivated in [section 2.3.1](#) an optimiser framework showing significant promise in studies called Optuna is also an approach to tune hyperparameters. Due to its advantage in implementation and efficiency, it has been selected as the hyperparameter tuning algorithm for models with many possible combinations.

For the supervised transfer learning and source-only experiments, the source-domain models use the same optimised hyperparameters selected during the Phase 1 baseline architecture development. This ensures that the transfer learning experiments are based on the already validated GRU architecture, rather than introducing a new independently tuned architecture for the source-domain training. The fine-tuning approaches then use a reduced learning rate, set to 10% of the original learning rate, to allow the pretrained weights to adapt more gradually and reduce the risk of catastrophic forgetting (Howard and Ruder,

2018). For DANN experiments, domain-adversarial hyperparameters are taken from Da Costa et al. (2020) rather than independently tuned. This is due to the complexity of tuning DANN architectures and ensures that the DANN implementation is reproducible and that the selected settings are based on an existing RUL domain-adaptation study rather than being chosen arbitrarily.

Typically, these approaches require a metric to assess if the hyperparameter selection is converging to an optimal combination (or, in the case of GS, to identify the best combination). The Root Mean Squared Error (RMSE) metric is selected as the metric used for the hyperparameter selection stage. Although validation results are computed for all evaluation metrics, the model-selection and hyperparameter-selection decisions are presented in the implementation and validation chapter (Chapter 5) are based on RMSE. Therefore, all models are selected according to the same error-based criterion, while the complete evaluation across all metrics is presented in the Results section (Chapter 6). The motivation for using RMSE is its tendency to penalise larger errors more heavily (Kollintza-Kyriakoulia et al., 2018). This is a desirable characteristic for RUL applications such as C-MAPSS in aircraft maintenance, where large prediction errors could lead to unexpected failures. For critical engineering systems such as aircraft, accurate prediction of RUL is important to reduce costs and ensure system reliability and prevent disasters (Tang et al., 2019). RMSE can be expressed as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (4.1)$$

where y_i is the true value, $\hat{y}_i$ is the predicted value, n represents the total number of observations used in the evaluation, and $(y_i - \hat{y}_i)^2$ is the squared prediction error. The reason why this metric penalises large errors more proportionally is because the error term is squared and therefore the errors grow quadratically as it increases (Chai and Draxler, 2014).

Since RUL is measured in engine cycles, RMSE is also expressed in cycles and provides a indication of the magnitude of the prediction error. For example, an RMSE of 10 indicates that the predicted RUL values typically deviate from the true RUL by approximately 10 cycles in magnitude, whereas an RMSE of 18 indicates considerably larger deviations of approximately 18 cycles. Therefore, lower RMSE values correspond to more precise estimates of the remaining operating life, although RMSE should not be interpreted as a strict error bound for every individual prediction.

4.5.4 Evaluation Metrics

RMSE also serves as a key metric to evaluate the prediction performance of the selected models. However, due to its characteristic, it has the limitation of biasing the results to outliers (Chai and Draxler, 2014). Moreover, simply assessing the performance of a model based on a single metric is not sufficient. Metrics for evaluating prognostic algorithms should also consider their impact on downstream PdM decisions, as these RUL predictions form the basis for optimising maintenance planning (Kamariotis et al., 2024).

Therefore, the evaluation methodology uses several additional metrics. As shown in [section A.10](#), a wide range of metrics used in PdM research have been identified. These cover accuracy/error metrics (refer to [subsection A.10.1](#)) and also assess computation efficiency [subsection A.10.1](#), reliability metrics ([subsection A.10.3](#)), fit ([subsection A.10.4](#)), generalisation ([subsection A.10.5](#)), interpretability and operational effectiveness ([subsection A.10.6](#)).

For the scope of this study, only the most commonly used metrics are used to cover the accuracy/error performance of the model, assess the fit, and consider the computational cost. Specifically, RMSE, mean absolute percentage error (MAE), and a scoring function commonly referred to as the C-MAPSS score. The C-MAPSS score proposed by NASA (2008) was used to evaluate RUL prediction performance. The score applies a larger penalty when the predicted RUL is higher than the true RUL than when it is lower. The scoring function is derived as follows (taken from Gao et al. (2023)):

$$\text{Score} = \begin{cases} \sum_{j=1}^n \left(e^{\frac{h}{a_1}} - 1 \right), & h \geq 0, \\ \sum_{j=1}^n \left(e^{\frac{h}{a_2}} - 1 \right), & h < 0, \end{cases} \quad h = \hat{y}_j - y_j \quad (4.2)$$

Where $\hat{y}_j$ is the predicted RUL, and what y_j is the actual RUL of the j th sample, and a_1 and a_2 are constants that control the sensitivity of the score depending on whether h is positive or negative. In other words, the scoring function (SF) weights the impact of under- and over-predictions, where smaller errors yield smaller score values, indicating better model performance.

Further metrics used in this study includes the coefficient of determination (R^2) is used to assess the fit. The training time (measured in seconds) quantifies the computation cost of each algorithm. For domain adaptation experiments using DANN, an additional metric is also reported, namely domain classifier accuracy. This metric measures how accurately the domain classifier can distinguish between the feature representations of the source-domain and the target-domain. Values closer to 0.5 indicate stronger domain confusion, since this suggests that the domain classifier is unable to reliably separate the two domains and that

the feature extractor has learnt more domain-invariant representations.

AUC-RMSE

This study also introduces an evaluation measure referred to as AUC-RMSE. To the best of the author’s knowledge, the specific metric has not previously been reported in the PdM literature. This metric was developed in response to the unequal distribution of observations across the RUL range, particularly the concentration of observations at the capped RUL value of 125 cycles. It was also motivated by Hyndman and Koehler (2006), who evaluated forecast accuracy separately at different forecast horizons before aggregating performance across the complete forecasting horizon. Whereas their proposed Mean Absolute Scaled Error (MASE) supports comparisons across time series with different scales, AUC-RMSE applies the related principle of evaluating error at different positions in the prediction domain to the RUL lifecycle.

AUC-RMSE is intended to be used alongside RMSE, to provide additional insight into model performance across the RUL range. Conventional RMSE gives each test observation equal influence. Therefore, if a large proportion of observations is concentrated within a particular region of the RUL range, prediction accuracy, then that region has a greater influence on the overall RMSE. In the present study, the RUL cap results in a comparatively high concentration of observations at 125 cycles (where the engines are considered healthy). A model that performs well within this densely populated early-life region may therefore obtain a favourable overall RMSE even when its performance is less consistent across the mid-life or near-end-of-life regions where RUL prediction becomes more important for PdM.

However, AUC-RMSE calculates prediction errors within predefined intervals and gives each interval equal influence on the final score. To interpret AUC-RMSE, as with conventional RMSE, lower values of AUC-RMSE indicate better predictive performance, while higher values indicate a greater prediction error across the RUL range.

Figure 4.2 shows a conceptual illustration of the AUC-RMSE metric.

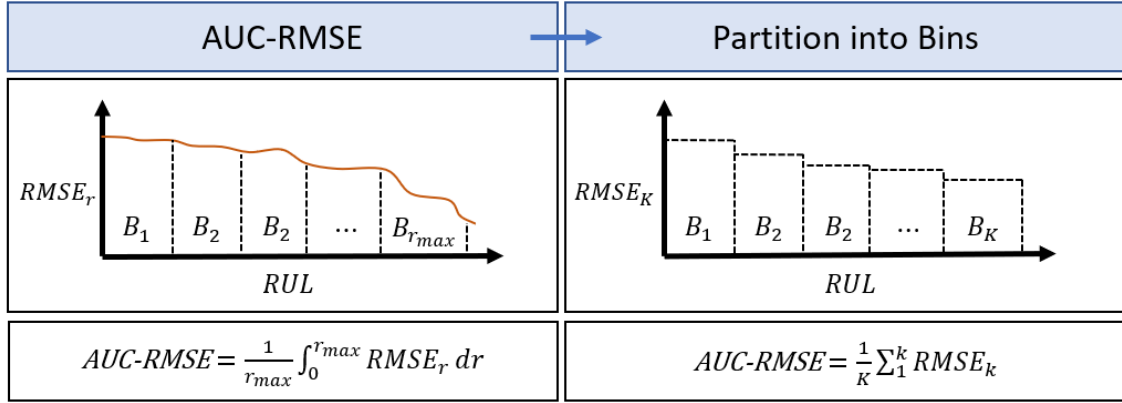


Figure 4.2: Conceptual illustration of AUC-RMSE

The AUC-RMSE is motivated by the idea of averaging the RMSE over the RUL range. In continuous form, this can be written as

$$AUC-RMSE = \frac{1}{r_{max}} \int_0^{r_{max}} RMSE(r) dr \quad (4.3)$$

In this study the lower limits $r_{min} = 0$ and $r_{max} = 125$ cycles. However, since the available RUL samples are finite, $RMSE(r)$ cannot be reliably estimated at every value of r cross the continuous RUL range. The RUL range is therefore divided into K bins ($B_1, \dots, B_K$), and an RMSE value, denoted by $RMSE_k$, is computed for each bin using all samples whose true RUL falls within that bin. The bin edges used in this study are

$$[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 125], \quad (4.4)$$

This includes 11 total bins used. The first 10 bins cover 10 cycles, while the final bin spans from 100 to 125 cycles. This wider final bin was chosen to limit the influence of the densely populated high-RUL region on the unweighted average. This design choice supports the use of AUC-RMSE alongside conventional RMSE by giving relatively greater emphasis to prediction performance in the lower RUL regions (which are less represented in the data).

Each bin is left-inclusive and right-exclusive, except for the final bin, which includes the upper boundary of 125 cycles. The bins are therefore defined as

$$[0, 10), [10, 20), \dots, [90, 100), [100, 125]. \quad (4.5)$$

For each bin B_k , an RMSE value is calculated using all test observations whose true RUL value falls within that bin:

$$\text{RMSE}_k = \sqrt{\frac{1}{n_k} \sum_{i: y_i \in B_k} (y_i - \hat{y}_i)^2}, \quad (4.6)$$

where n_k is the number of observations in bin B_k , y_i is the true RUL value, and $\hat{y}_i$ is the corresponding predicted RUL value. In the context of sequential models, the sliding windows are considered individually when evaluating both aggregate and lifecycle-specific performance using the metrics. Subsequently, the multiple overlapping sliding windows that originated from the same engine are evaluated independently and do not have average or weighting applied at the engine level. Each of the individual windows is then classified by lifecycle category based on the true RUL value at the last cycle number of the window in question. All such values with the same classification are then aggregated together to compute the corresponding bin specific RMSE.

The overall AUC-RMSE is then obtained by assigning equal weight to each bin and calculating the arithmetic mean of the bin-level RMSE values:

$$\text{AUC-RMSE} = \frac{1}{K} \sum_{k=1}^K \text{RMSE}_k \quad (4.7)$$

where $\mathcal{K}$ is the set of bins containing at least one test observation.

It should be noted that AUC-RMSE as a single metric value loses important information. Therefore, a complete picture of the distribution of the RMSE values across the bins is obtained by plotting the RMSE_k values against the RUL range. The plot reveals the shape of the curve, whether the error distribution is skewed, and where the accuracy is highest and lowest. This then more explicitly shows at which point in the engine lifecycle the RUL prediction models perform best and where they struggle.

However, due to the scope of this work, it is impractical to show and discuss the AUC-RMSE plots for every model and training-data condition. In total, the experimental design includes 31 model configurations: 17 baseline models, 6 feature-selection baseline models, 3 hybrid feature-extraction models evaluated on two feature-selected datasets, and 5 transfer learning models. Each of these configurations is also evaluated across multiple levels of training-data (see [Table 4.1](#)). Therefore, the AUC-RMSE plots are used selectively in the results chapter ([Chapter 6](#)), where representative models with the strongest test-set performance are analysed in more detail.

Moreover, to consolidate the findings across the different training-data levels, the results are grouped according to broad data-availability ranges. Specifically, performance is summarised across the 80 to 40 engine range, 40 to 20 engine range, 20 to 10 engine range, 10 to 5 engine range, and 5 engines and less than. Therefore, the results are presented in terms of high-data, moderate-data, low-data, very-low-data, and extremely limited-data

conditions, rather than discussing every individual engine-count condition separately for each model (recall [Table 4.1](#) for all data availability scenarios).

Furthermore, it should be noted that the interpretation of AUC-RMSE will depend on the width of the bins chosen for each lifecycle. Smaller bins provide a more detailed understanding of how prediction errors vary throughout the lifecycle, however, smaller bins can result in less stable RMSE values as there will generally be fewer data observations included in any one interval. Larger bins will provide a more general overview and may obscure small scale variations in performance. In particular, as the width of the bins increase, AUC-RMSE will begin to represent the overall view provided by traditional RMSE. For example, if only a single bin which spans the entire RUL range was used, then AUC-RMSE would have been equivalent to the traditional RMSE. Moreover, it is acknowledged that the exact AUC-RMSE values and the rankings of closely performing models could change under alternative bin widths. Therefore, a systematic empirical sensitivity analysis across multiple bin widths is identified as an area for future research.

Chapter 5

Implementation and Validation

The training and validation stages form a fundamental part of a ML pipeline. Using the same dataset for both model selection and performance evaluation can introduce optimistic bias in estimates of generalisation ability. During training, model parameters are learnt from the data by minimising a defined objective function, while hyperparameters control the learning process by controlling aspects such as model complexity and optimisation behaviour. These choices have a direct impact on the model’s ability to generalise to unseen data.

As described in [Chapter 4](#), the C-MAPSS dataset is split into three subsets, namely, a training set, a validation set, and a test set. The results presented in this chapter do not reflect the true generalisation performance of the models, as they are based on training and validation procedures used for model selection and hyperparameter tuning. The evaluation on the held-out test set is presented in the subsequent chapter ([Chapter 6](#)).

All experiments were conducted on a system equipped with a 12th Gen Intel Core i7-12700H processor, 16 GB of RAM, and an NVIDIA GeForce RTX 3060 GPU. The implementation was carried out using Python 3.10.16, with GPU-accelerated training used, where supported, through CUDA. The principal software packages included NumPy, pandas, scikit-learn, XGBoost, LightGBM, TensorFlow, and Optuna. Furthermore, the git repository at (Maré, 2026) contains the main experimental source code, data-processing scripts, configuration files, and software dependency requirements used in the experiments.

To control for stochastic variation and to be able to produce repeatable experimental results, a single fixed random seed (`RANDOM_SEED = 42`) was used throughout the implementation of the experiments. This fixed seed was used for Numpy, tf-kernels of tensorflow, Python’s random module, all `random_state` parameters of applicable scikit-learn classes, and Optuna TPE samplers. Engine-subset sampling and other stochastic preprocessing operations also used this fixed seed. Each neural network configuration at each experimen-

tal condition was trained once with evaluation using this fixed seed. Due to the wide scope of the experiments, which included over 280 model trainings/evaluations across multiple model configurations and data-availability conditions, repeating the complete experimental design over several random seeds was not computationally viable within the scope of the study. A supplementary sensitivity analysis was therefore conducted across five different random seeds for selected representative models and data-availability scenarios, as presented in [subsection 6.6.6](#). This analysis provides an indication of the variability associated with stochastic training.

5.1 Baseline Degradation Modelling

This section aims to establish a benchmark by evaluating models from three families (classical regression, tree-based, and neural-networks) in their baseline forms under progressive training data limitations. Therefore, to achieve the sixth objective (refer to [subsection 1.1.4](#)), this chapter examines how fundamentally different algorithms perform as data becomes scarce, without any additional feature-engineering steps. Subsequently, this enables a controlled baseline evaluation. Each baseline algorithm is trained using between 80 and 10 complete engine degradation trajectories, in steps of 10, and between 10 and 1 trajectory, in steps of 1 (17 data settings per model).

This phase requires 289 experiments (17 models are each evaluated under 17 training data availability configurations). In addition, the hyperparameters of each model are tuned once in the full training set and are kept fixed for the smaller training sets. Although this keeps the cross-comparison of the models more controlled by using a single tuned hyperparameter setting, it is acknowledged that the optimisation scope changes relative to the dataset's size. Settings where hyperparameters are retuned at smaller training sets can potentially make individual optimisation trials less computationally intensive and result in potentially wider exploration of parameters. Therefore, for future work, it could be interesting to separately optimise each training data configuration and investigate how optimisation behaviour changes as data becomes scarcer.

5.1.1 Baseline Model Implementation

The baseline models were developed in Python and several libraries were used. Scikit-learn was used for classical regression models and RF. For neural-network-based models, TensorFlow and Keras were used. [Table 5.1](#) shows the summary of the 17 models and corresponding libraries and main objects used for the implementation.

Table 5.1: Summary of baseline models, libraries and the main objects used.

Model	Library or API	Implementation Object
MLR	scikit-learn	<code>LinearRegression()</code>
Ridge	scikit-learn	<code>Ridge()</code>
Lasso	scikit-learn	<code>Lasso()</code>
ElasticNet	scikit-learn	<code>ElasticNet()</code>
Poly2	scikit-learn	<code>PolynomialFeatures(degree=2)</code>
Poly3	scikit-learn	<code>PolynomialFeatures(degree=3)</code>
RF	scikit-learn	<code>RandomForestRegressor()</code>
XGB	XGBoost	<code>XGBRegressor()</code>
LGBM	LightGBM	<code>LGBMRegressor()</code>
ANN	TensorFlow / Keras	<code>Sequential()</code> with <code>Dense()</code> layers
RNN	TensorFlow / Keras	<code>layers.SimpleRNN()</code>
LSTM	TensorFlow / Keras	<code>layers.LSTM()</code>
GRU	TensorFlow / Keras	<code>layers.GRU()</code>
BiLSTM	TensorFlow / Keras	<code>layers.Bidirectional(LSTM())</code>
CNN	TensorFlow / Keras	<code>layers.Conv1D()</code>
TCN	TensorFlow / Keras	<code>layers.Conv1D()</code> with dilated residual blocks
Transformer	TensorFlow / Keras	<code>layers.MultiHeadAttention()</code> within a custom Transformer block

The regression and tree-based models are trained on the non-windowed feature representation of the C-MAPSS dataset, while the neural-network-based models are trained on windowed sequential inputs. Moreover, the neural-network models require the input data to be represented as three-dimensional tensors. Tensors are data-storing objects that can be processed by models through multidimensional array operations (Chollet and Watson, 2025). The learning behaviour of these model families differs according to both their underlying architecture and their training procedure (see [section 2.1](#)). In particular, the models are trained by minimising their respective objective functions, while the neural-network-based models are optimised using the Adam optimisation algorithm during iterative weight updating. As highlighted in [section 2.3](#), model training is strongly influ-

enced by the hyperparameter settings selected before training. The following subsection discusses the implementation of the hyperparameter optimisation techniques implemented.

5.1.2 Hyperparameter Tuning Implementation

Hyperparameter tuning was performed using GS and BOA via Optuna. The tuning strategy was selected based on the dimensionality of the hyperparameter search space for each algorithm. As discussed in [section 2.3](#), exhaustive GS becomes increasingly impractical as the dimensionality of the hyperparameter search space increases because the number of possible combinations grows rapidly. Models with two or fewer searched hyperparameters were therefore tuned using GS, since exhaustive evaluation remained computationally practical for these lower-dimensional search spaces. Optuna was applied to models with higher-dimensional search spaces, where exhaustive GS would result in a substantially larger number of hyperparameter combinations and therefore become computationally impractical. The use of different optimisation approaches was therefore based on the size of the respective search spaces rather than the model family itself. All algorithms optimised through Optuna were subjected to 50 trials with an early stopping patience of 15 trials, and for neural-networks each trial had a patience of 10 epochs. A summary of the number of hyperparameters explored for each model and the subsequent tuning approach is presented in [Table 5.2](#).

Table 5.2: The number of explored hyperparameters per model.

Family	Models	Number of Searched Hyperparameters	Tuning Method
Regression	MLR, Poly2, Poly3	0	None
Regression	Ridge, Lasso	1	Grid search
Regression	ElasticNet	2	Grid search
Tree-based	RF	5	Optuna
Tree-based	XGB	7	Optuna
Tree-based	LGBM	6	Optuna
Neural-Network	ANN, RNN, LSTM, GRU, BiLSTM, CNN	6	Optuna
Neural-Network	TCN	5	Optuna
Neural-Network	Transformer	7	Optuna

The classical regression models involved minimal tuning. MLR does not require hyperparameter optimisation. For polynomial regression, second- and third-order polynomial

basis functions were evaluated as separate predefined baseline model configurations (Poly2 and Poly3). The regularised linear model variants (Ridge, Lasso, and ElasticNet) were tuned with respect to their regularisation hyperparameters. Specifically, the regularisation strength α was treated as a hyperparameter for both Lasso and Ridge, while ElasticNet also included the L1-ratio. These hyperparameters were selected through GS based on minimum validation RMSE.

The hyperparameter search spaces used for these models were guided by the documentation of the respective libraries, computational constraints, and preliminary training runs conducted using only the training set. These preliminary runs were used to identify ranges that were computationally feasible and to exclude settings that resulted in clearly unstable training. Their purpose was therefore to establish practical search-space bounds (not final configurations). The final search ranges were selected to provide sufficiently broad coverage of plausible configurations while remaining computationally practical for the number of model evaluations required in this study. Final hyperparameter selection was subsequently performed using the GS or Optuna procedures based on minimum validation RMSE. The complete hyperparameter search spaces are provided in. ([section C.1](#))

5.1.3 Grid Search

For the regularised linear models (Lasso, Ridge, and ElasticNet), coefficient estimation follows the regularised objectives shown in [Equation 2.3](#), [Equation 2.4](#), and [Equation 2.5](#). Therefore, once a regularisation setting is specified, model training proceeds by minimising the corresponding objective functions.

The hyperparameter tuning is then applied as an external selection step using an exhaustive GS. For each model, predefined candidate values were specified for the regularisation hyperparameters, and each configuration was trained on the training set and assessed on the validation set. The selection was based on the minimum RMSE validation (see [Equation 4.1](#)), and the selected hyperparameter setting is fixed for subsequent engine-count experiments. The tuning visualisations for these models can be seen in [Figure 5.1](#).

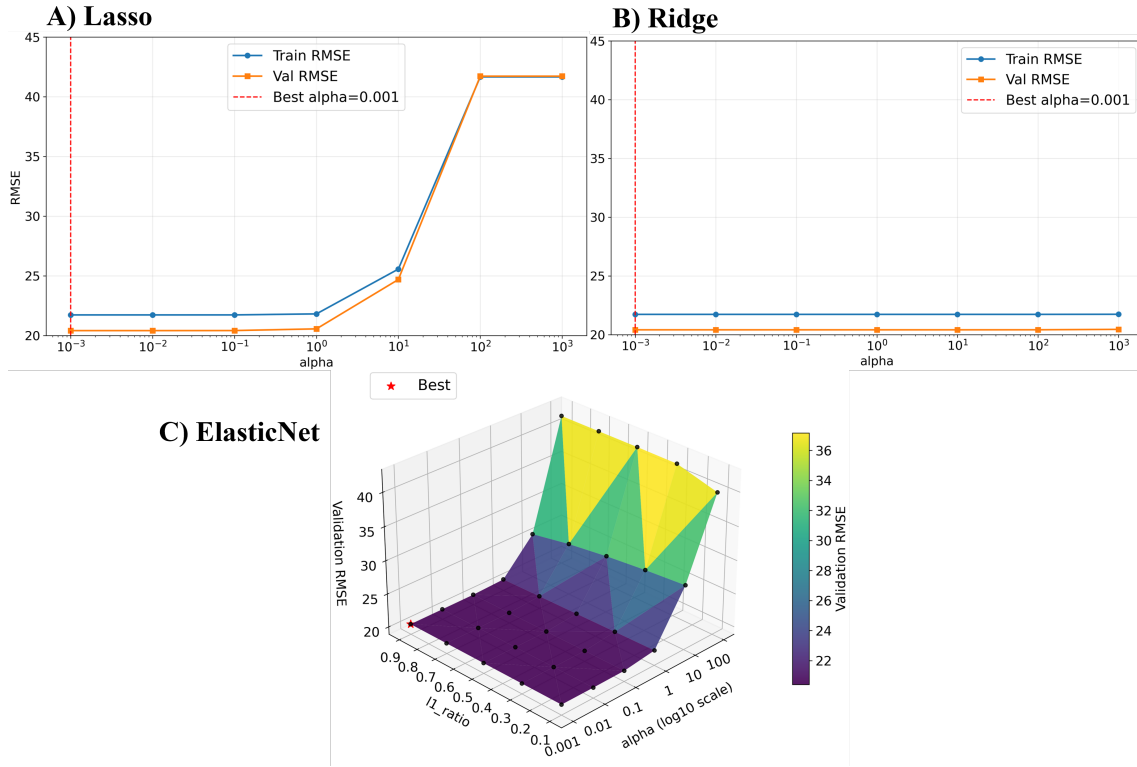


Figure 5.1: Hyperparameter Optimisation Curve RMSE for A)Lasso, B)Ridge and C) ElasticNet

Figure 5.1 shows the training and validation RMSE across the evaluated regularisation strengths for Lasso and Ridge, while the ElasticNet surface shows validation RMSE across the combined α and L1-ratio search space. Hyperparameter selection was based on the minimum validation RMSE rather than the training RMSE, since the validation set was used for model selection. For Lasso and Ridge, the validation RMSE is marginally lower than the training RMSE at lower values of α . This small difference may be attributed to differences in the characteristics of the training and validation engine trajectories, which may result in slight differences in prediction difficulty between the two subsets. The close correspondence between the training and validation RMSE, however, indicates that no substantial generalisation gap is evident within this region.

Although the minimum validation RMSE occurs at the lowest evaluated regularisation strength ($\alpha = 0.001$), Figure 5.1 shows that the improvement in validation RMSE diminishes as α decreases, with performance becoming relatively stable for $\alpha < 1$. For larger values of α , performance degrades significantly, particularly for Lasso, where the L1 penalty drives the coefficients towards zero and leads to underfitting. Ridge shows more stable behaviour across larger α values, likely due to the smoother shrinkage characteristic of L2 regularisation. ElasticNet introduces an additional hyperparameter, the L1 ratio, and is therefore visualised using a 3D surface. The results indicate that variations in the

L1 ratio have minimal impact on validation performance at lower values of α , while performance similarly deteriorates for larger α values. The small differences between training and validation RMSE at lower regularisation strengths provide no indication of substantial overfitting. However, the limited improvement obtained through regularisation may indicate that the linear models are unable to fully capture the underlying relationships in the data.

5.1.4 Optuna

Optuna was used as the Bayesian hyperparameter optimisation framework for all tree-based and neural-network models (see [section 2.3.1](#) for the theoretical background). For each model, tuning was implemented as a study that minimised validation RMSE over the predefined search space for each model. Candidate configurations were generated using Optuna’s Tree-structured Parzen Estimator (TPE) sampler.

Each trial consisted of model training on the training set followed by evaluation on the validation set. The best hyperparameter configuration was selected as the trial with the minimum validation RMSE. The high level Optuna-based hyperparameter tuning procedure used is shown in [algorithm 1](#).

Algorithm 1: Optuna-based hyperparameter optimisation procedure

Input: Model M , search space Λ_M , training data $\mathcal{D}_{train}$, validation data $\mathcal{D}_{val}$, trial budget T **Output:** Best hyperparameter configuration λ^* **Define objective function** $f(\text{trial})$;Sample $\lambda \in \Lambda_M$ using `trial.suggest_*` methods;Instantiate model $M(\lambda)$;**if** M is a neural-network **then** Attach `EarlyStopping(monitor=val_loss, patience=10, restore_best_weights=True)`; Attach `TFKerasPruningCallback(trial, monitor=val_loss)`;**end**Train $M(\lambda)$ on $\mathcal{D}_{train}$;Compute validation RMSE on $\mathcal{D}_{val}$;

Return validation RMSE;

Initialise Optuna study;

Set optimisation direction to minimise;

Set sampler to `TPESampler(seed=42)`;**if** M is a neural-network **then** Enable pruning using `MedianPruner(n_startup_trials=10, n_warmup_steps=5)`;

Use study-level early stopping callback with patience = 15 non-improving completed trials;

else Disable pruning (`pruner=None`);**end**Execute `study.optimize(objective, n_trials = T)`; $\lambda^* \leftarrow \text{study.best_params}$;

[algorithm 1](#) defines an objective function in which each trial is one hyperparameter configuration. λ represents the individual configuration, while Λ represents all possible configurations. The maximum number of trials T was set at 50. For neural-network-based algorithms, early stopping and pruning was used. The early stopping was set to 10. Therefore, if the validation RMSE does not improve after 10 epochs, then the training stops and the best weights is restored.

Pruning was implemented using Optuna’s median-based strategy to terminate unpromising neural-network trials based on intermediate validation performance. The starting trails were set to 10. This prevents pruning before 10 trials have been completed. The Warm up steps were set to 5 to ensure that each trail is trained for at least five epochs before considering pruning. The patience for pruning was set to 15. To ensure that the trials are

reproducible, the TPE sampler was configured with a random seed set to 42.

The complete hyperparameter search spaces for the baseline models are provided in [section C.1](#). Furthermore, the optimal hyperparameters found for the classical regression models are shown in [section C.2](#). Additionally, the hyperparameters found for the models tuned through Optuna (including tree-based and neural-network-based models) are presented in [section C.3](#).

The complete hyperparameter search spaces for the baseline models are provided in [section C.1](#). The final hyperparameter configurations selected for the baseline models are summarised in Table 5.3.

Table 5.3: Selected hyperparameter configurations for the baseline models.

Model	Selected Hyperparameters
Lasso	$\alpha = 0.001$
Ridge	$\alpha = 0.001$
ElasticNet	$\alpha = 0.001$; L1-ratio = 0.9
RF	n_estimators = 500; max_depth = 20; min_samples_split = 14; min_samples_leaf = 3; max_features = sqrt
XGB	n_estimators = 966; max_depth = 5; learning_rate = 0.010171; subsample = 0.890795; colsample_bytree = 0.823410; min_child_weight = 1; gamma = 1.633698
LGBM	n_estimators = 409; num_leaves = 61; learning_rate = 0.016729; subsample = 0.778232; colsample_bytree = 0.628778; min_child_samples = 12
ANN	n_hidden_layers = 2; units_layer1 = 256; units_layer2 = 32; dropout_rate = 0.2; learning_rate = 0.01; batch_size = 128
RNN	n_recurrent_layers = 1; units_layer1 = 256; units_layer2 = 128; dropout_rate = 0.5; learning_rate = 0.0001; batch_size = 32
LSTM	n_lstm_layers = 1; units_layer1 = 128; units_layer2 = 64; dropout_rate = 0.1; learning_rate = 0.001; batch_size = 32
BiLSTM	n_bilstm_layers = 1; units_layer1 = 64; units_layer2 = 32; dropout_rate = 0.5; learning_rate = 0.001; batch_size = 64
GRU	n_gru_layers = 1; units_layer1 = 256; units_layer2 = 64; dropout_rate = 0.3; learning_rate = 0.0005; batch_size = 32
CNN	n_conv_layers = 2; n_filters = 64; kernel_size = 5; dropout_rate = 0.1; learning_rate = 0.0005; batch_size = 128
TCN	n_filters = 32; n_blocks = 3; dropout_rate = 0.4; learning_rate = 0.01; batch_size = 32
Transformer	num_transformer_blocks = 2; d_model = 128; num_heads = 2; ff_dim = 256; dropout_rate = 0.1; learning_rate = 0.001; batch_size = 256

For Lasso and Ridge regression, the regularisation strength is controlled by α . The larger the values the greater the impact of regularisation. Lasso applies an L1 penalty, which can shrink some coefficients to exactly zero and therefore perform implicit feature selection, while Ridge applies an L2 penalty that generally shrinks coefficients without eliminating them. Notably, both models selected value of $\alpha = 0.001$ represents relatively weak regularisation implying that stronger regularisation was not beneficial for validation. Furthermore, Elastic net is also seen to select the same value for $\alpha = 0.001$ (L2 penalty) and selected L1 ratio of 0.9 places substantially greater emphasis on the L1 component (although the overall regularisation remains weak).

The tree-based models have higher-dimensional hyperparameter spaces, with RF, XGB, and LGBM having 5, 7, and 6 tuned hyperparameters, respectively. A common parameter across all three models is `n_estimators`, which controls the number of trees included in the ensemble. The RF is configured to grow trees to a maximum depth of 20 levels through `max_depth=20`, while `min_samples_split=14` further constrains tree growth by requiring at least 14 samples in a node before it can be split. Similarly, `min_samples_leaf=3` requires each resulting leaf to contain at least three samples which will prevent the formation of leaves based on very small numbers of observations. The `max_features=sqrt` configures RF to calculate the square root of the available features. XGB selected 966 boosting trees with a maximum depth of 5 levels. The relatively low `learning_rate=0.0102` means that each successive tree makes only a small adjustment to the prediction that accumulated. The `subsample=0.8908` configuration means that each tree is trained using approximately 89% of the available training samples. In addition the hyperparameter `colsample_bytree=0.8234` limits each tree to approximately 82% of the available features. Then the `min_child_weight=1` sets the minimum weight required in a child node. The `gamma` specifies the minimum reduction in the XGB objective required before an additional tree split is accepted. Therefore, a candidate split is only created if it improves the objective by at least 1.6337. LGBM selected 409 boosting trees with a maximum of 61 leaves per tree. The relatively low `learning_rate=0.0167` means that each successive tree makes a comparatively small contribution to the accumulated prediction (similar to XGboost). Furthermore, the `subsample=0.7782` setting uses approximately 78% of the training observations for sampling, while `colsample_bytree=0.6288` limits each tree to approximately 63% of the available features. Then the `min_child_samples=12` requires each leaf to contain at least 12 observations.

For LSTM, BiLSTM, and GRU, the same six hyperparameters were tuned. This includes the number of recurrent layers, number of units, dropout rate, learning rate, and batch size. All three selected a single recurrent layer, although they differed in the number of units used. The LSTM selected 128 units, while the BiLSTM selected 64 units and the GRU selected 256 units. A larger number of recurrent units generally increases the representational capacity of the model, but also increases the number of trainable parameters and therefore the computational cost of training. Therefore, when interpreting the

training time metrics of these models this should be taken into consideration. Moreover, although the BiLSTM appears comparatively small, it processes the input sequence in both the forward and backward directions, meaning that its computational cost is higher than the number of units alone may suggest. The LSTM is configured with the lowest drop out rate of 0.1, meaning that 10% of the relevant activations are randomly dropped (in order to prevent overfitting). BiLSTM notably has the highest drop out rate of 0.5. The learning rates for both LSTM and BiLSTM were configured at 0.001 with batch sizes (the number of observations processed before each weight update) of 32 and 64 respectively.

CNN and TCN had 6 and 5 hyperparameters tuned respectively. The CNN selected two convolutional layers with 64 filters and a kernel size of 5. This means that each convolutional layer learns 64 filters, with each filter examining a local window of five time steps at a time. The selected dropout rate of 0.1 provides relatively limited regularisation. The of CNN was tuned to a learning rate of 0.0005 with batch size of 128. The TCN selected 32 filters across three TCN blocks. Increasing the number of blocks increases the depth of the network and allows the dilated convolutions to capture temporal relationships over a wider receptive field. The selected dropout rate of 0.4 applies stronger regularisation than that selected for the CNN. The TCN also selected a substantially larger learning rate of 0.01 and a smaller batch size of 32 which means its weights will be updated more frequently using smaller groups compared to the CNN configurations.

For the Transformer, the selected configuration consisted of two Transformer blocks with a model dimension of 128, two attention heads, and a feed-forward dimension of 256. The two attention heads will allow the model to learn different relationships between positions in the input sequence and the model dimension of 128 controls the size of the internal feature representation processed by the Transformer. A dropout rate of 0.1 provides relatively limited regularisation. The selected learning rate was 0.001, while the comparatively large batch size of 256 means that each weight update is calculated using 256 observations.

Figure 5.2 shows the total optimisation time taken for each algorithm.

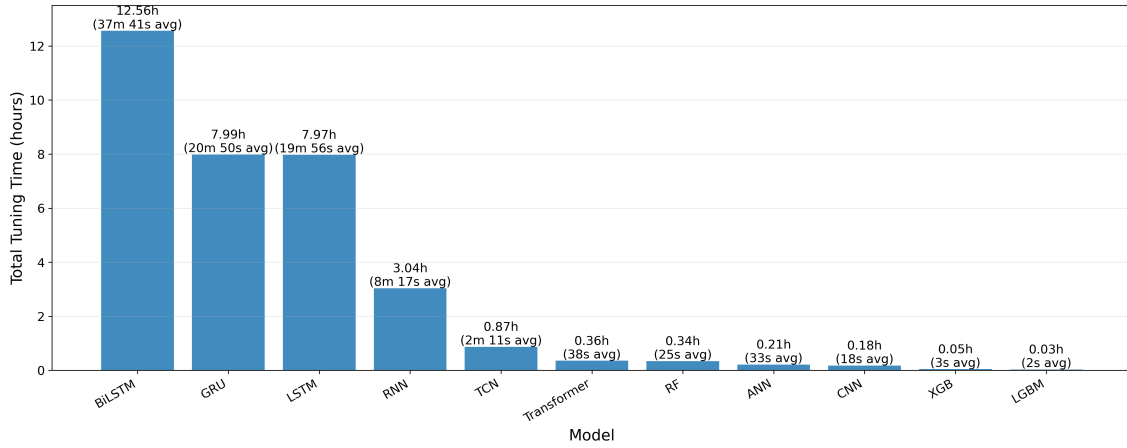


Figure 5.2: Total hyperparameter tuning time per algorithm using Optuna

From [Figure 5.2](#), it is evident that neural-network-based models generally require substantially longer hyperparameter optimisation times compared to tree-based models. This is expected, as neural-networks are trained iteratively over multiple epochs using gradient-based optimisation, whereas tree-based models rely on decision tree construction and ensemble methods such as bagging and boosting.

BiLSTM took the longest time of 12.56 hours to optimise followed by GRU and LSTM taking just under 8 Hours. In particular, despite GRU being considered an efficient variant of LSTM (recall [section 2.1.3](#)), it took roughly the same amount of time. However, this is largely due to the best hyperparameters selected for GRU compared to LSTM, where LSTM used 128 units in the first hidden layer, while GRU used 256, making it a larger model. Furthermore, a longer time for BiLSTM is expected due to the process sequences in both forward and backward directions, which effectively doubles the computational requirements, although its first and second layers are only 64 and 32 units. ANN and CNN models demonstrate significantly lower optimisation times relative to the recurrent architectures. This is consistent with their structural properties, as feed-forward networks are simpler in which they do not maintain temporal state, and convolutional architectures involve parameter sharing and local connectivity, which results in more efficient computation (recall [section 2.1.3](#) and [subsection 2.2.1](#), which discusses the theory of ANN and CNN, respectively). Moreover, TCN and Transformer models also show comparatively efficient optimisation behaviour. Tree-based boosting models, particularly XGBoost and LightGBM, showed the lowest optimisation times, with average trial durations of only a few seconds.

The hyperparameter optimisation trajectories for the neural-network and tree-based algorithms are shown in [Figure 5.3](#). The x-axis represents the trail number and the y-axis shows the RMSE validation score, where each plot represents the 'best-so-far' validation score. The faint scatter plots show the validation RMSE values of all individual trails.

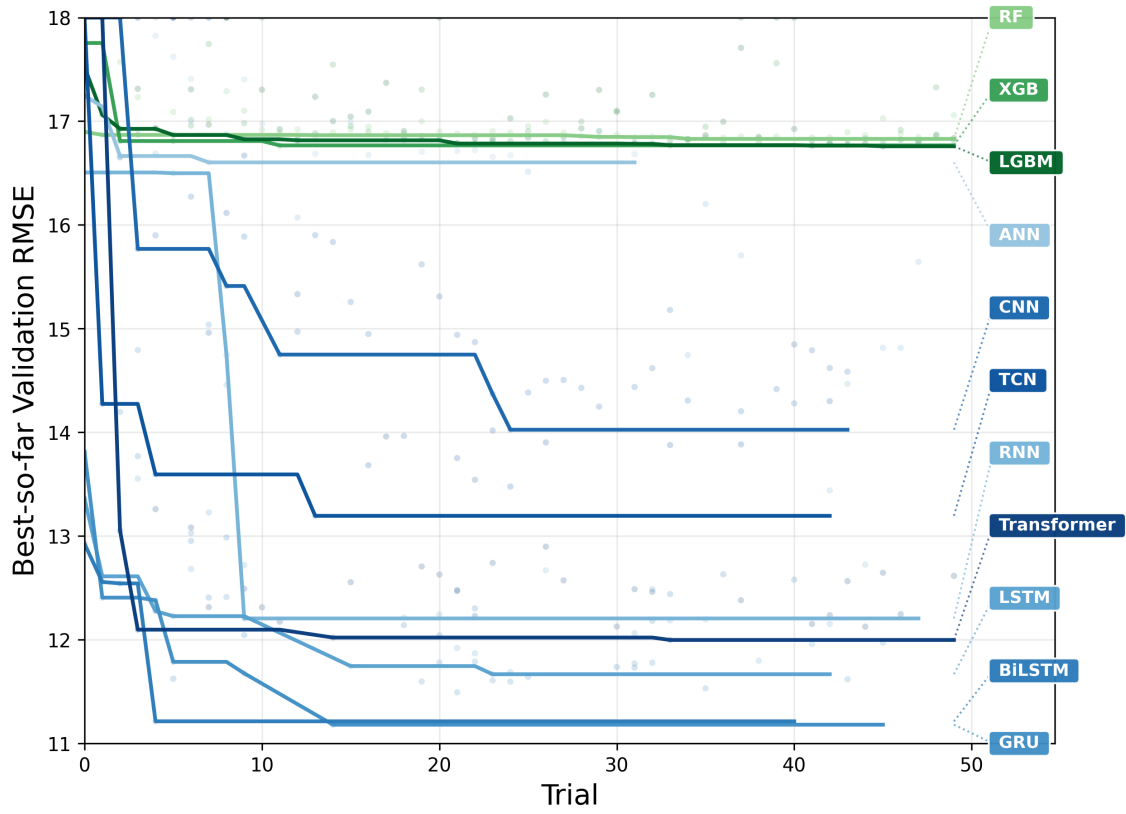


Figure 5.3: Best-so-far validation RMSE versus Optuna trial for tree-based and neural-network models

As seen in Figure 5.3, most models converge within the first 5 to 15 trials. The tree-based models plateau quickly, with all three models reaching a final best-so-far validation RMSE just below 17. In contrast, neural-network models show stronger overall validation performance. The best-performing models, including Transformer, LSTM, BiLSTM, and GRU, achieve validation RMSE values below 12. Most curves plateau before approximately 40 trials, and thus the use of a $T = 50$ trial budget for the Optuna search is validated as sufficient. Although the validation curves for the tree-based models are less competitive than those of the neural-networks, they still considerably outperform the linear models in terms of RMSE validation (Figure 5.1). This provides further evidence that the linear models struggle to capture the non-linear relationships in the data compared with tree-based and neural-network models. Notably, the ANN converges quickly but performs more comparably to the tree-based models than to the recurrent and sequence-based neural-networks. This suggests that modelling the temporal structure is important for the C-MAPSS degradation task.

5.2 Feature Selection

For the FS phase, a representative group of baseline models is used. The selection is based on the RMSE validation ranking in the full 80-engine training environment (14241 total observations), while also retaining at least one model from each model family. [Table 5.4](#) shows the selected models implemented in this phase.

Table 5.4: Baseline validation RMSE results at 80 engines with 14 241 total observations.

Validation Rank	Model	Model Family	RMSE	Selected for Phase 2
1	GRU	neural-network	11.2926	Selected
2	Transformer	neural-network	12.1872	Selected
3	BiLSTM	neural-network	12.1931	Selected
4	LSTM	neural-network	12.4308	Selected
5	RNN	neural-network	12.7659	Not selected
6	CNN	neural-network	14.8829	Not selected
7	TCN	neural-network	15.1109	Not selected
8	ANN	neural-network	16.6305	Not selected
9	XGB	Tree-based	16.7183	Selected
10	LGBM	Tree-based	16.7487	Not selected
11	RF	Tree-based	16.8257	Not selected
12	Poly2	Regression	18.5711	Selected
13	Poly3	Regression	18.6070	Not selected
14	Lasso	Regression (linear)	20.4109	Selected
15	MLR	Regression (linear)	20.4109	Not selected
16	Ridge	Regression (linear)	20.4109	Not selected
17	ElasticNet	Regression (linear)	20.4110	Not selected

The RMSE validation results show evidence that the RNNs respond best on the C-MAPSS dataset, with GRU performing the best with a validation RMSE of 11.29. All neural-network-based models outperformed tree-based models (with ANN slightly behind XG-Boost). Linear regression models performed the weakest of all families. In particular, Lasso, MLR, and Ridge had the same score, indicating that regularisation was not necessary. Polynomial regression models outperformed linear models, but increasing the degree of polynomial did not improve performance and led to poorer results compared to tree-based models.

The neural-network models selected for this phase were therefore GRU, Transformer, BiL-

STM and LSTM. To assess how other model families respond to feature reduction, XGBoost, Poly2 and Lasso were also included as representative tree-based and regression models for further experimentation.

5.2.1 Correlation-based Feature Selection

The exploratory analysis in [section 3.3](#) specifically [Figure 3.5](#) (Correlation matrix of C-MAPSS sensors and RUL for the FD001 training dataset) showed that several C-MAPSS sensor measurements have stronger linear associations with RUL than others. After ranking the features, each absolute correlation value was divided by the sum of all absolute correlations to obtain its proportional contribution. The cumulative sum of these proportions was then used to retain the highest-ranked features until 99% of the total absolute correlation strength was captured.

This resulted in the selection of 14 of the original 24 input sensors and operational setting features (refer to [Table 3.2](#) for all sensor parameters and [Table 3.4](#) for operational settings). The selected features include:

$$\{s11, s4, s12, s7, s15, s21, s20, s17, s2, s3, s8, s13, s9, s14\}.$$

This subset contains the features that are most strongly correlated with RUL and excludes those with weak correlations. The resulting lower-dimensional dataset is then used to train the chosen baseline models.

5.2.2 Tree-based feature selection

Pearson correlation captures only linear relationships and, therefore, may exclude features with non-linear relationships with RUL. Therefore, the use of RF as a FS method is also explored.

To implement this, the Sklearn RF Regressor object is used. The parameters were set at `n_estimators = 100`, `max_depth = 10` and `Random seed = 42`. [Figure 5.4](#) shows the importance scores for each feature.

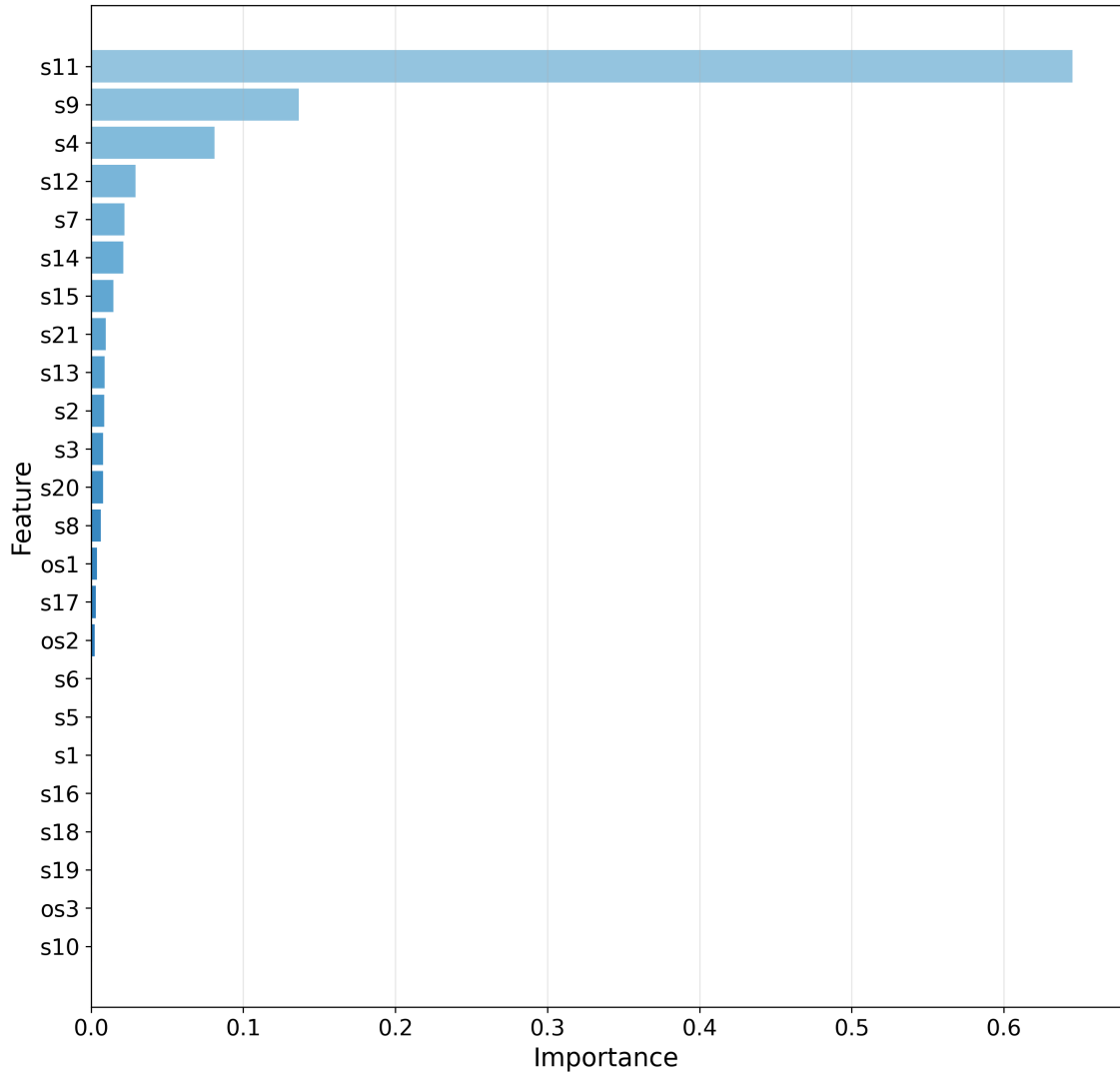


Figure 5.4: Ranked Random Forest feature importance scores for the FD001 training dataset

From Figure 5.4 it is evident that s_{11} (Static pressure at the HPC outlet) is the feature that contributes significantly more importance than the other characteristic relative to the target RUL. To determine which other features contribute enough to be selected, a cumulative importance threshold of 99% was used. This resulted in 12 selected features:

$$\{s_{11}, s_9, s_4, s_{12}, s_7, s_{14}, s_{15}, s_{21}, s_{13}, s_2, s_3, s_{20}\}$$

In particular, the tree-based approach selected the same features as the correlation-based method but excluded s_{17} and s_8 . The exclusion of s_8 and s_{17} from the RF-selected subset does not imply that these measurements contain no degradation information, as both were retained by the correlation-based approach and showed degradation-related trends

during exploratory analysis. Rather, they provided less additional importance within the fitted RF model. Specifically, s_8 represents the regulated physical fan speed, while s_{17} represents bleed enthalpy and provides a more indirect indication of HPC degradation than the strongly selected compressor-pressure measurements.

The selected features showing particular importance to the s_{11} (HPC outlet) makes sense due to FD001 containing a single simulated fault mode of high-pressure-compressor (HPC) degradation. Since C-MAPSS represents component deterioration through changes in component flow capacity and efficiency, which then alter measurable engine quantities, it is physically reasonable that an HPC-outlet pressure measurement is highly sensitive to degradation occurring in the HPC itself. Moreover, since deterioration of compressor flow capacity and efficiency affects the pressure and thermodynamic response of the engine, the strong importance assigned to this measurement is physically plausible. The selection of other measurements related to HPC pressure (s_7), turbine temperature (s_4), fuel-related behaviour (s_{12}), bypass behaviour (s_{15}), and turbine cooling flows (s_{20} and s_{21}) is also consistent with the response of the engine to progressive component degradation.

5.2.3 Feature Selection Validation

The selected models from Table 5.4 were trained on the two subset of features. Table 5.5 shows the RMSE validation results.

Table 5.5: Validation RMSE at 80 engines across feature selection methods.

Model	Baseline	Correlation_FS	Tree_FS
GRU	11.293	11.802	11.542
Transformer	12.187	11.760	12.006
BiLSTM	12.193	12.087	11.923
LSTM	12.431	12.020	11.925
XGB	16.718	16.751	16.862
RF	16.826	16.822	16.894
Poly2	18.571	18.543	18.615
Lasso	20.411	20.479	20.526

In all models, the differences in RMSE are small (generally within 0.1–0.4), suggesting that FS methods have a limited impact on predictive performance for this data set. No method consistently outperformed all models. Correlation FS performs best for Transformer, RF, and Poly2, while Tree based FS slightly outperforms for LSTM and BiLSTM. Despite the insignificance in RMSE the FS does add the benefit of reducing the dimension of the dataset which, in turn, speeds up the training time for the models. Figure 5.5 shows the

training time for each of the selected models.

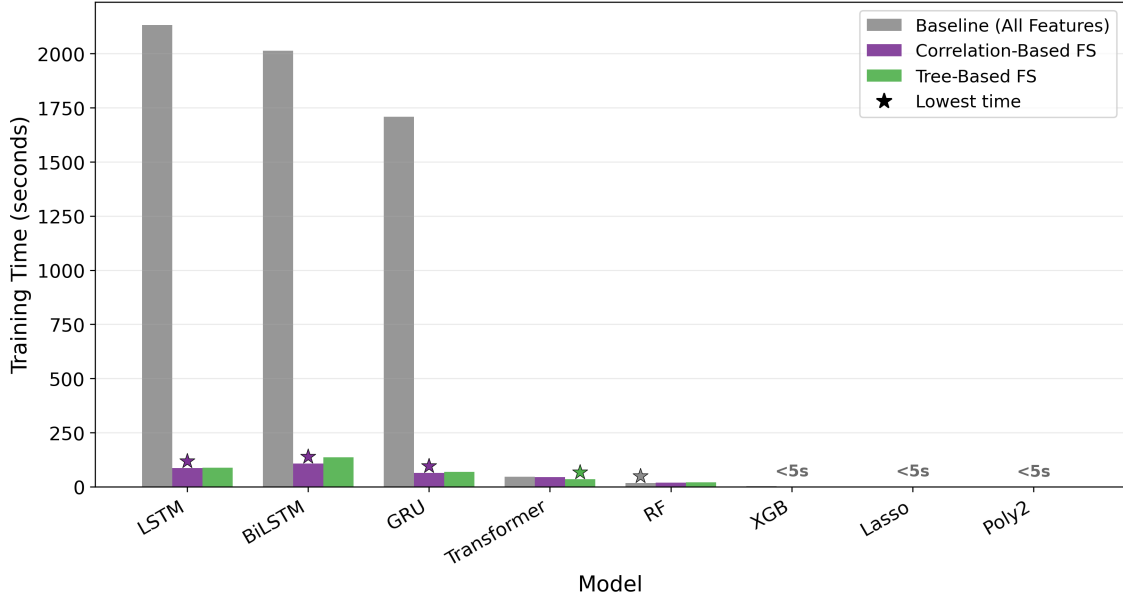


Figure 5.5: Training time comparison for selected models at 80 engines using the full 24-feature baseline, the 14-feature correlation-based subset, and the 12-feature Random Forest importance subset.

From Figure 5.5 it is evident that FS significantly reduced training time for the neural-network models. Although the validation RMSE differences between the two FS methods were small, the correlation-based subset was selected for subsequent experiments. This subset retains 14 features versus 12 for the tree-based method, which preserves two additional sensor signals that may capture additional degradation information but still achieves substantial reductions in training time.

5.3 Feature Extraction

Based on the RMSE validation results, it is evident that in terms of accuracy, the neural-network-based models have considerably outperformed the tree-based and classical regression models. In addition, through FS, the dimension of the dataset could be reduced, resulting in a significant reduction in training time. A further advantage of neural-network models is that FE can be directly integrated into the architecture. These FE layers are jointly trained with the prediction layers through backpropagation.

Of the neural-network models, GRU shows the most promising validation results (as it obtained the lowest validation RMSE of all models and the lowest training time of the neural-network-based models). Therefore, the influence of feature extractors (FE) on the

GRU predictor is investigated. Specifically, CNN-GRU, TCN-GRU, and AE-GRU hybrid architectures are implemented.

The hyperparameters found in the baseline model phase are kept fixed for the GRU predictor. Only the FE layer was tuned using Optuna over 30 trials, with validation RMSE as the optimisation objective. The selected extractor configurations were then fixed, and hybrid models are evaluated across all subsets of training engine-count.

5.3.1 CNN-GRU

Convolution-based extractors are designed to detect short-term structures in the sensor signals. Convolutions provide a shift-invariant extraction of features, which means that these local degradation patterns can be detected regardless of the exact position within the input sequence (Krč ek et al., 2024). This is an advantage, as degradation does not always occur at the same time-step for every engine. However, the convolution-based extractor learns long-term dependencies due to its limited receptive field (Zeng et al., 2023). The GRU component is a suitable prediction layer as it focuses on modelling the temporal dependencies as well as the sequence evolution. Due to its gate components, it has the ability to use long-term memory to learn how information accumulates over time and, subsequently, how the earlier degradation signals relate to failure (recall section 2.1.3, for an overview of the GRU architecture). Figure 5.6 shows the hybrid architecture flow diagram implemented for CNN-GRU.

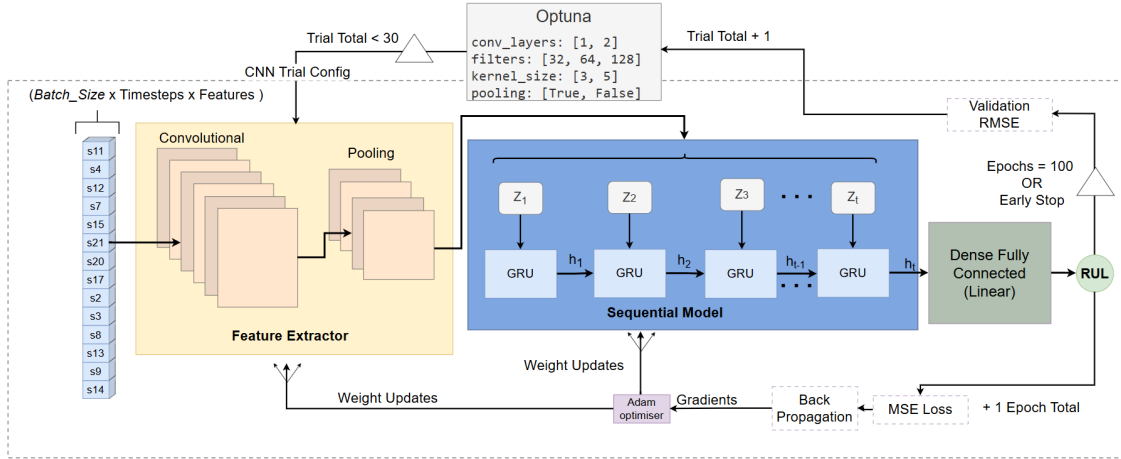


Figure 5.6: CNN-GRU architecture, training and hyperparameter tuning flow.

Similarly to the baseline GRU model, the hybrid CNN-GRU architecture receives a 3D tensor of dimension batch size $\times$ time steps $\times$ features. The batch size is a hyperparameter taken from the baseline GRU and fixed at 32. The time steps correspond to the global window length used when the series was sequenced, 30, and the feature dimension is the 14

sensors kept after correlation-based FS. A full mini-batch therefore has the shape $32 \times 30 \times 14$. Sliding the window over the training trajectories produces 14,241 training sequences, each of shape 30×14 ; during optimisation these are grouped into mini-batches of size 32, giving $14,241 / 32 = 446$ parameter-update steps per epoch (the last step uses whatever sequences remain after the full batches).

The batches are first inputted into the FE, which consists of convolutional layers and a pooling layer. The configuration of the feature extractor is tuned by Optuna through four hyperparameters, namely, the number of convolutional layers, filters, kernel size, and if pooling should take place. The number of filters is the number of convolution kernels used in a CNN layer. The kernel size determines the window length that the filter looks at one time. If pooling is selected, then that layer reduces the dimensionality before passing it to the GRU.

The hyperparameters obtained from the tuning of the baseline GRU are used. The convolutional FE outputs a sequence of feature vectors Z_t , where each vector represents local temporal patterns at time step t . These vectors are fed sequentially into the GRU, which updates its hidden state h_t by combining the current input Z_t with the previous hidden state h_{t-1} . The learnt feature representation is then mapped to a single continuous RUL prediction value through a linear activation.

Similarly to the GRU baseline experiments, each Optuna trial is trained for a maximum of 100 epochs with early stopping (patience set to 15). Mean Squared Error (MSE) is used as the loss function, and the gradients are computed through backpropagation, with the Adam optimiser jointly updating the weights of both the FE (CNN) and the sequential model (GRU). After either 100 epochs are reached or early stopping is triggered, the next Optuna trial is performed. The subsequent trial uses the validation RMSE with Optuna's Bayesian-based optimisation to determine the next CNN hyperparameter configuration to evaluate within the specified search space.

5.3.2 TCN-GRU

The TCN-GRU architecture follows a similar design; however, the FE is replaced by dilated CNNs, which is designed to capture longer-range temporal dependencies through dilations (recall [subsection 2.2.2](#), for the conceptual discussion of the TCN architecture) before the GRU receives the sequence. [Figure 5.7](#) shows the following.

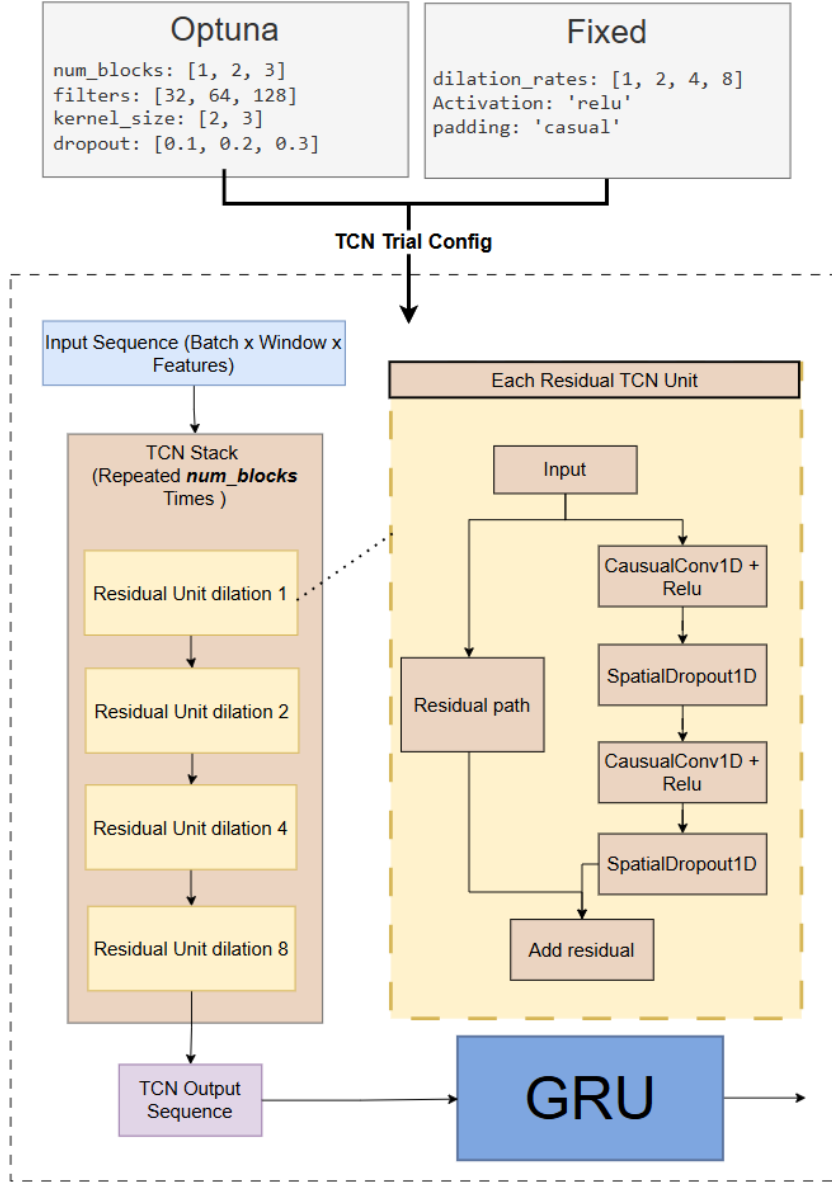


Figure 5.7: TCN-GRU FE architecture.

The TCN layer consists of a TCN stack that performs the forward pass through four dilation units. The dilation rates are set as fixed parameters of 1, 2, 4, and 8, respectively, and this entire dilation stack is repeated according to the tuned hyperparameter `num_blocks`. Each residual dilation unit functions to sequentially increase the receptive field and consists of two layers of causal convolutions and `SpatialDropout1D` layers. In addition, the units contain a residual shortcut connection that adds the block input back to the transformed output. Since activations within convolutional feature maps are often strongly correlated, standard element-wise dropout may regularise them ineffectively (Tompson et al., 2015). Therefore, `SpatialDropout1D` is used in TCN blocks to drop entire feature maps rather than individual activations, which promotes greater independence between learnt feature maps and reduces co-adaptation (TensorFlow, 2024). Causal convolution is implemented

by setting the padding configuration to causal. This prevents the leakage of information from the future to the past (Goay et al., 2023). The resulting TCN output sequence is then passed to the GRU.

5.3.3 AE-GRU

An AE is a neural-network used in unsupervised learning that first transforms the input into a compact representation and then reconstructs it with the aim of matching the original input as closely as possible (recall, subsection 2.2.3). The compact representation (referred to as the latent representation) can help capture non-linear embeddings while preserving the underlying structure of the data (Joas et al., 2025). Figure 5.8 shows the AE-GRU architecture, training, and hyperparameter tuning flow.

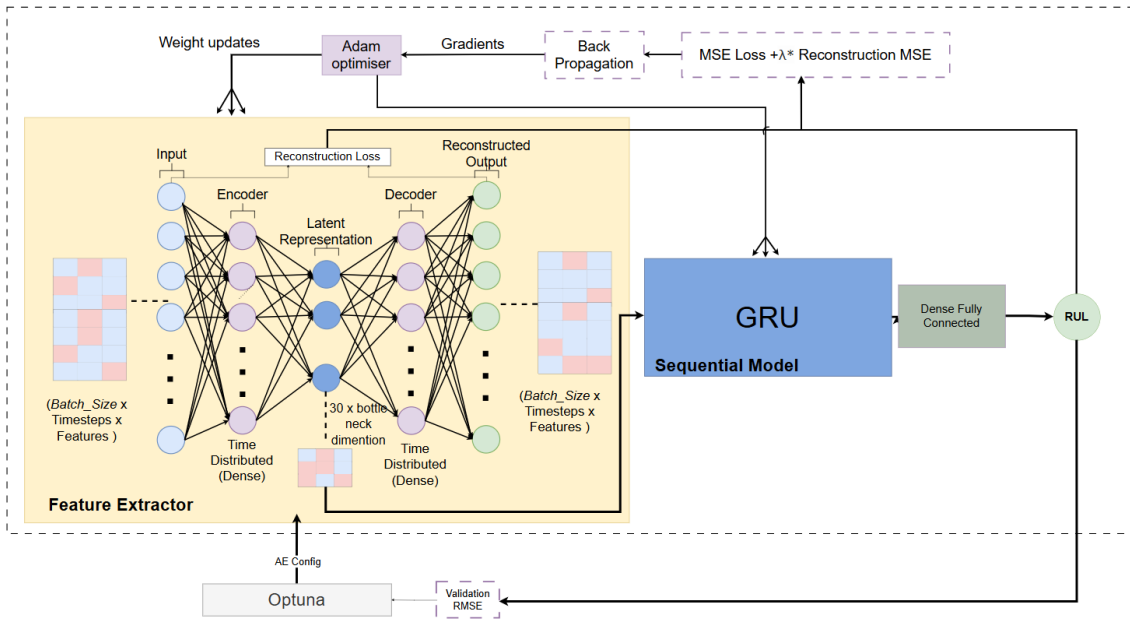


Figure 5.8: AE-GRU architecture, training and hyperparameter tuning flow.

The implemented AE-GRU architecture contains a time-distributed AE (which independently encodes and decodes each time step from the input sequence) FE. The encoder applies a dense feed transformation to each time step. The latent representation is then passed through to the GRU model, as well as a time-distributed decoder to reconstruct the original features.

The difference in the training loss between the CNN-GRU and AE-GRU arises from the different roles of their feature-extraction components. The CNN-GRU and TCN-GRU perform supervised RUL prediction, with their feature extractors trained jointly with the GRU using the RUL prediction MSE. In contrast, the AE-GRU includes an additional reconstruction objective because the AE is required to learn a latent representation that

preserves sufficient information to reconstruct the original sensor inputs. As discussed in Equation 2.22, AE training therefore involves minimising the reconstruction difference between the input and its reconstructed output. In the AE-GRU implementation, this reconstruction objective is retained alongside the supervised RUL prediction objective. Therefore, the total loss is defined as the sum of the RUL prediction MSE and a weighted AE reconstruction MSE, following Ragab et al. (2020):

$$L(\theta) = L_{\text{rul}}(\theta) + \lambda L_{\text{rec}}(\theta) \quad (5.1)$$

where $L_{\text{rul}}(\theta)$ represents the RUL MSE loss and $\lambda L_{\text{rec}}(\theta)$ represents the reconstruction loss weighted by the hyperparameter λ . Moreover, since the input data are three dimensions (batch size, time window, and number of feature), the implementation is done as follows:

$$L_{\text{total}} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 + \lambda \cdot \frac{1}{NTF} \sum_{i=1}^N \sum_{t=1}^T \sum_{f=1}^F (x_{i,t,f} - \hat{x}_{i,t,f})^2 \quad (5.2)$$

Where, L_{total} is the joint objective used to train the AE-GRU model. Backpropagation computes the gradients of L_{total} with respect to all parameters of the AE and GRU, which are then updated using the Adam optimiser. The L_{total} is the addition of the RUL related MSE and is summed by a weighted reconstruction MSE loss where N , T and F represent the batch size, time steps and number of features, respectively. λ is the hyperparameter multiplied by the reconstruction loss, which controls the contribution of the reconstruction loss to the overall loss function during training. For hyperparameter selection, like CNN-GRU and TCN-GRU, Optuna still selects the best configuration using validation RUL RMSE.

5.3.4 Feature Extraction Validation

The full search ranges for each of the hybrid models can be seen in section C.4 and the selected hyperparameters are shown in Table 5.6 and can also be referred to in section C.5). After running 30 Optuna trails on each model with each trail capped at 100 epochs with 15 patience early stopping, the following hyperparameter settings were selected (Table 5.6)

Table 5.6: Optuna-selected configurations for hybrid GRU models.

Model	Selected hyperparameters
CNN-GRU	conv_layers = 1; filters = 32; kernel_size = 5; pooling = False; batch_size = 256.
TCN-GRU	num_blocks = 3; filters = 32; kernel_size = 5; dropout = 0.3; batch_size = 32.
AE-GRU	encoder_layers = 1; encoder_units = 32; bottleneck_dim = 4; λ_{recon} = 0.1; batch_size = 32.
Shared settings	
Global settings	Window size = 30; feature selection method = Correlation_FS; selected features = 14; Optuna trials = 30; maximum epochs = 100; early stopping patience = 15; activation = ReLU; optimiser = Adam.
GRU fixed settings	Units = 256; layers = 1; dropout = 0.3; learning rate = 0.0005.

It can be observed that relatively compact architectures were selected for all three models. CNN-GRU is configured with a single convolutional layer with a larger batch size, whereas both TCN-GRU and AE-GRU used smaller batch sizes. In particular, Optuna selected the lowest bottleneck dimension (latent space) of 4 for the AE FE. Figure 5.9 shows the original 14 feature correlation matrix compared to the compressed latent space generated by the AE during training.

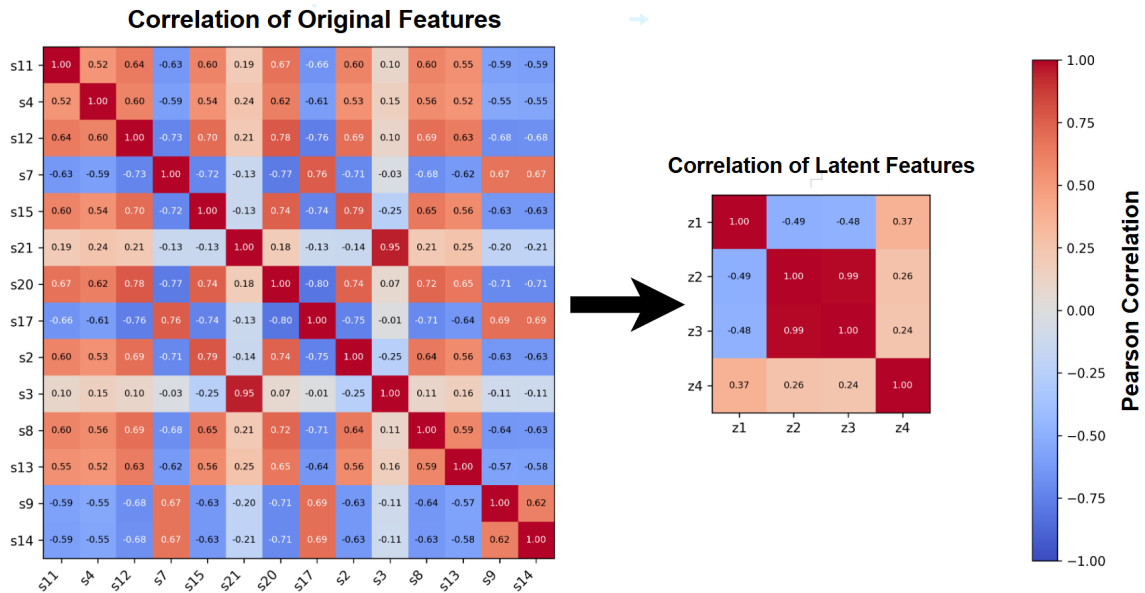


Figure 5.9: AE-GRU correlation of 14 originally selected features compared with the correlation of the latent features generated by the AE.

The AE was able to find a compact latent representation of 4 features. Notably, two of the latent features (z_2 and z_3) were highly correlated implying that an even further reduction of the dimension of the latent space could be possible. However, z_1 , which was less correlated, and z_4 shows moderate independence.

Although the AE reduced the 14 selected input features to a four-dimensional latent representation before being supplied to the GRU, the reported AE-GRU training time represents the total training time of the combined framework, including the additional training of the AE. Consequently, reducing the input dimensionality of the GRU does not necessarily reduce the overall training time. The total AE-GRU training time was therefore substantially longer (3m 3s) than that of the stand-alone GRU (1m 4s), as the additional computational cost of training the AE offsets any potential reduction in the GRU component. Of all hybrid models, the TCN-GRU had the longest total training time at 8 minutes and 26 seconds, while the CNN-GRU reduced the training time by approximately 20 seconds relative to the stand-alone GRU. It should further be noted that the training time of the GRU component within the AE-GRU framework was not measured separately; therefore, the reported values should be interpreted as total framework training times. Figure 5.10 shows the final model training time for the pure GRU and hybrid models.

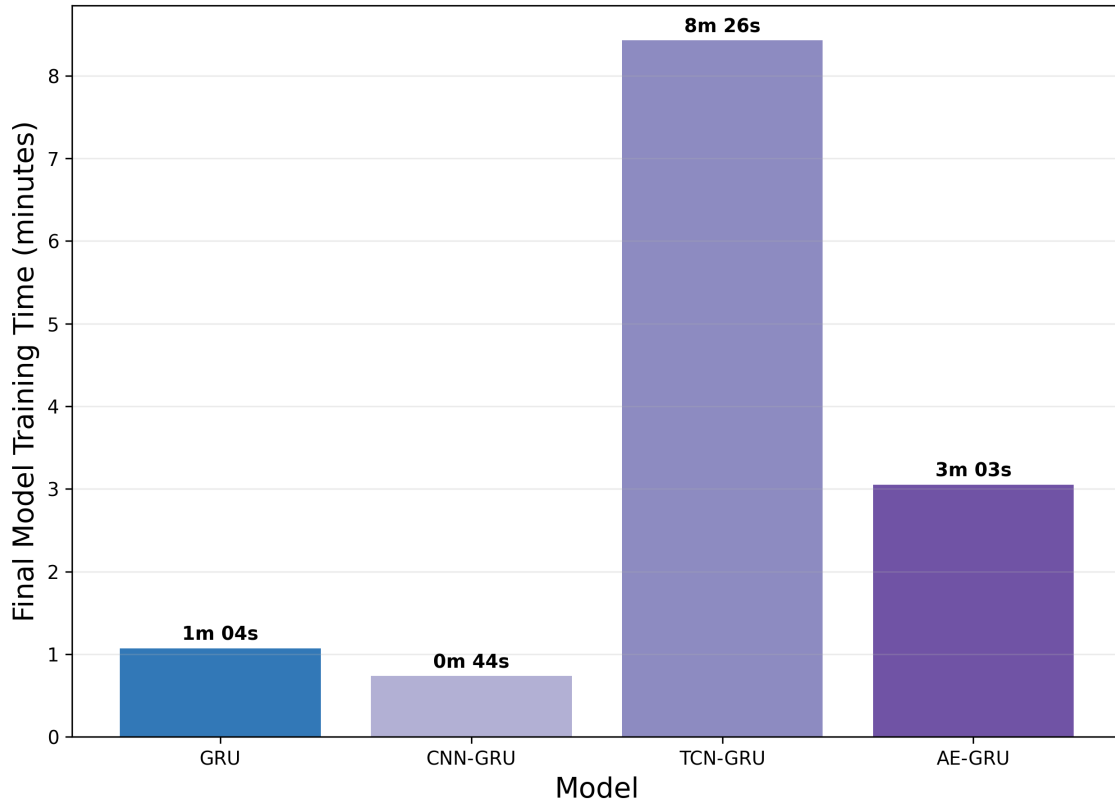


Figure 5.10: Training durations of the selected models (80 engines) for GRU CNN-GRU, TCN-GRU, and AE-GRU.

Figure 5.11 shows the Optuna hyperparameter optimisation history for the CNN-GRU, TCN-GRU, and AE-GRU models. Notably, despite the integration of FE layers, the GRU baseline (trained on all 24 features) still achieved the lowest 'best-so-far' validation RMSE score during Optuna tuning. This suggests that the selected feature set may already have been sufficiently informative for the GRU to model degradation relationships directly. Moreover, the additional FE layers potentially only increased the model complexity without yielding a corresponding validation RMSE improvement.

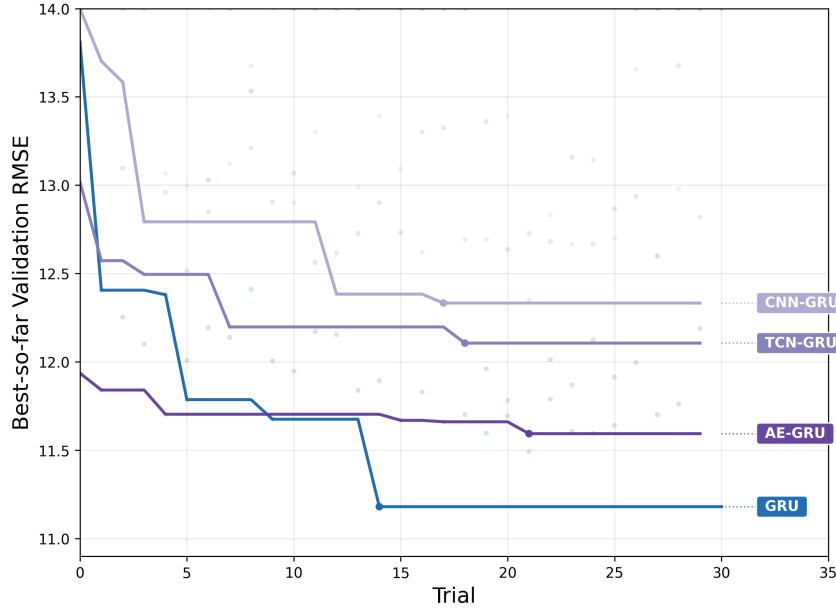


Figure 5.11: Optuna hyperparameter optimisation history for the CNN-GRU, TCN-GRU, and AE-GRU models.

5.4 Fine-tuning Transfer Learning Implementation

This section presents the implementation of transfer learning (TL) models based on fine-tuning as part of the final experimental phase. Specifically, the best-performing baseline model(selected based on validation RMSE), is pretrained on the source domains FD002 and FD003 and subsequently adapted to the target domain FD001. Validation is used for model selection and training convergence within this phase.

Since the GRU model achieved the best RMSE validation score, it is selected as the base architecture for TL. The GRU model is pretrained separately on FD002 and FD003 and then each pretrained model is fine-tuned on the FD001 target dataset. The input data used to pretrain the models is the same set of features selected by correlation FS (14 sensors). Therefore, homogeneous TL can be carried out as the models use the same feature space during pretraining and fine-tuning.

5.4.1 Maximum Mean Discrepancy Analysis

As discussed in [Chapter 3](#) (which provides an overview of C-MAPSS), FD001, FD002, FD003 and FD004 are datasets collected under different operating conditions and fault modes. These differences can result in a domain shift in the input distribution, which can affect how suitable a source dataset is for TL. To quantify the distributional similarity between FD001, FD002, FD003, and FD004, MMD was computed on the same windowed representation and the same 14 selected features used in the prior experiments.

For each dataset, the windowed training array was loaded in the form (N, T, F) , where N denotes the number of windows, T denotes the window size, and F denotes the number of selected features. The 3D training windows were then flattened into a two-dimensional representation by multiplying the window size and the number of features, so that each window was treated as one observation in the MMD calculation. Moreover, since the datasets differ in size and MMD requires computationally intensive pairwise kernel matrix calculations, the datasets were randomly sampled. Specifically, a maximum of 10,000 training windows was sampled without replacement from each dataset, using a random seed equal to 42.

To compute MMD, the biased empirical estimate of squared MMD was used. This can be formulated as follows, adapted from Gretton et al. (2012):

$$\widehat{\text{MMD}}_b^2(X, Y) = \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m k(x_i, x_j) + \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n k(y_i, y_j) - \frac{2}{mn} \sum_{i=1}^m \sum_{j=1}^n k(x_i, y_j). \quad (5.3)$$

where $\widehat{\text{MMD}}_b^2(X, Y)$ denotes the biased empirical estimate of squared MMD between samples X and Y , and $k(\cdot, \cdot)$ denotes the Gaussian RBF kernel, defined as:

$$k(a, b) = \exp(-\gamma \|a - b\|^2) \quad (5.4)$$

where γ controls the kernel width and was determined separately for each dataset pair using the median-distance heuristic described by Gretton et al. (2012). For each dataset pair, the windowed observations from the two datasets were combined and the positive off-diagonal pairwise squared Euclidean distances were calculated. Letting d_{med}^2 denote the median of these squared distances, the kernel parameter was calculated as:

$$\gamma = \frac{1}{2d_{\text{med}}^2}. \quad (5.5)$$

This represents the median heuristic in the γ -parametrisation of the Gaussian RBF kernel and with the kernel width adapted separately to the scale of each dataset pair.

Furthermore, x_i and y_j denote individual window observations sampled from the two datasets, and m and n are the corresponding sample sizes. Therefore, MMD is computed by substituting the population expectations with empirical averages over the sampled observations (Gretton et al., 2012). The estimated MMD values between the C-MAPSS dataset domains can be seen in Table 5.7

Table 5.7: Pairwise Maximum Mean Discrepancy (MMD^2) values between C-MAPSS domains using an RBF kernel.

Domain A	Domain B	MMD^2
FD002	FD004	0.000054
FD001	FD003	0.012435
FD002	FD003	0.020084
FD003	FD004	0.020116
FD001	FD002	0.022136
FD001	FD004	0.022159

FD002 and FD004 show a MMD^2 value close to zero, indicating that their feature representations are highly similar. In contrast, FD001 shows a larger domain gap to FD002 and FD004, with MMD^2 values of 0.022136 and 0.022159, respectively. This large discrepancy is expected since FD001 contains data from a single operating condition, whereas FD002 and FD004 contain data from six operating conditions. FD001 and FD003 also show a discrepancy, with a MMD^2 value of 0.012435. This is notably larger than the discrepancy between FD002 and FD004, even though both domain pairs differ in terms of fault-mode only. However, relative to the other cross-domain comparisons, the FD001-FD003 discrepancy is moderate.

Since FD003 shows only a moderate discrepancy from FD001, it is expected that TL from FD003 to FD001 may be more effective, since the source and target feature representations are relatively more similar. In contrast, TL from FD002 to FD001 is expected to be more challenging since this pair shows one of the largest observed domain discrepancies. The following section therefore evaluates fine-tuning-based TL from both FD002 and FD003 source domains to the FD001 target domain.

5.4.2 Fine-tuning Implementation

The fine-tuning approaches implemented in this section follow the TL strategies discussed in subsection 2.4.2. Specifically, full fine-tuning, layer freezing, and gradual unfreezing were implemented. All other hyperparameters in the previously optimised GRU configuration were retained. The three strategies are first trained on the FD002 and FD003 datasets

before fine-tuning on the target FD001 dataset. Figure 5.12 shows the implementation of the three fine-tuning strategies

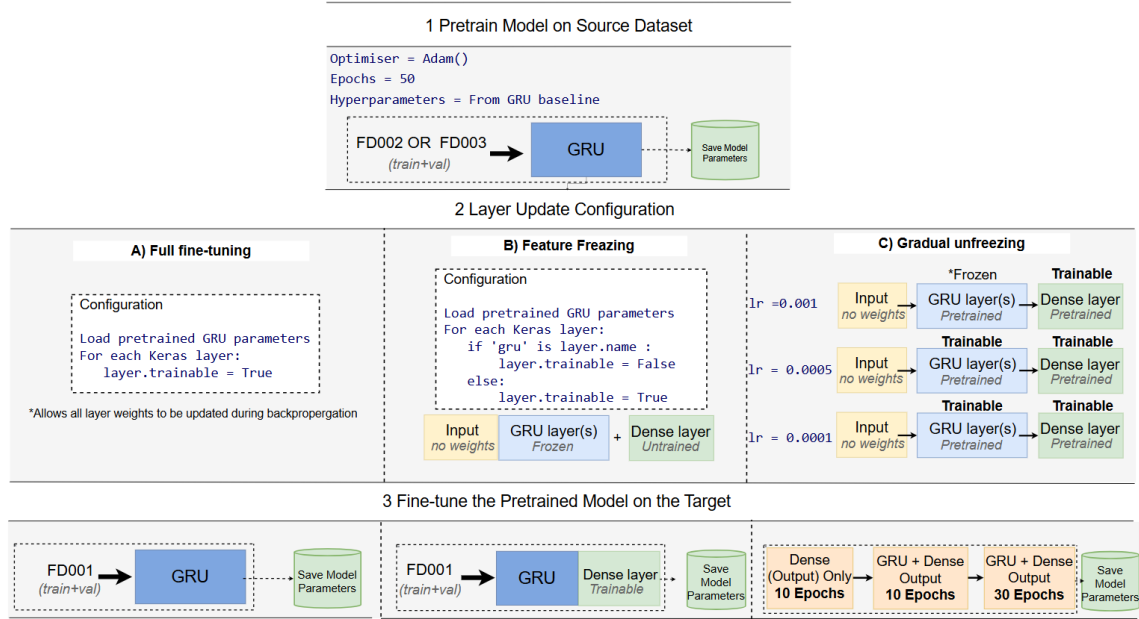


Figure 5.12: Overview of the supervised transfer-learning workflow used for the GRU model. In Step 1, the GRU model is pretrained on a source dataset (FD002 or FD003). In Step 2, one of three fine-tuning strategies is configured: (A) full fine-tuning, where all pretrained GRU layers remain trainable; (B) layer freezing, where the pretrained GRU layers are frozen and only the output dense layer remains trainable; or (C) gradual unfreezing, where training begins with the pretrained GRU layers frozen and these layers are progressively unfrozen during fine-tuning. In Step 3, the configured model is fine-tuned using labelled target-domain data from FD001. Here, lr denotes the learning rate, while “Frozen” and “Trainable” indicate whether the parameters of the corresponding layer are updated during fine-tuning.

Full fine-tuning

All three fine-tuning strategies involved pretraining the source datasets (FD002 or FD003) with the GRU baseline hyperparameter configuration. The fine-tuning layers were all configured to train with a maximum of 50 epochs with a total of 100 epochs for training the full fine-tuned models.

Full fine-tuning involves using a pretrained network and adapting it to the target domain. To implement this, all pretrained layers were kept trainable. This means that during backpropagation, the loss calculated on the target-domain-data was used to update the weights in both the recurrent GRU layers and the final prediction layer. The model was then recompiled using the Adam optimiser with a reduced learning rate of 0.0001. The

feature freezing strategy involved freezing the GRU layer and adding a dense feed-forward output layer. This dense layer is initially untrained and while the pretrained GRU is kept frozen, the dense layer is set to trainable to adapt to FD001. The gradual unfreezing strategy, on the other hand, splits the 50 epochs into 3 stages. It starts with a pretrained dense output layer and initially is trained with a frozen GRU and a trainable output layer for 10 epochs and a learning rate of 0.001. The GRU layer is then unfrozen with a more gradual learning rate of 0.0005 for 10 epochs. In the last stage, the learning rate is set to 0.0001 for a more gradual fine-tuning to FD001. The training times for each of the fine-tuning strategies are shown in Figure 5.13.

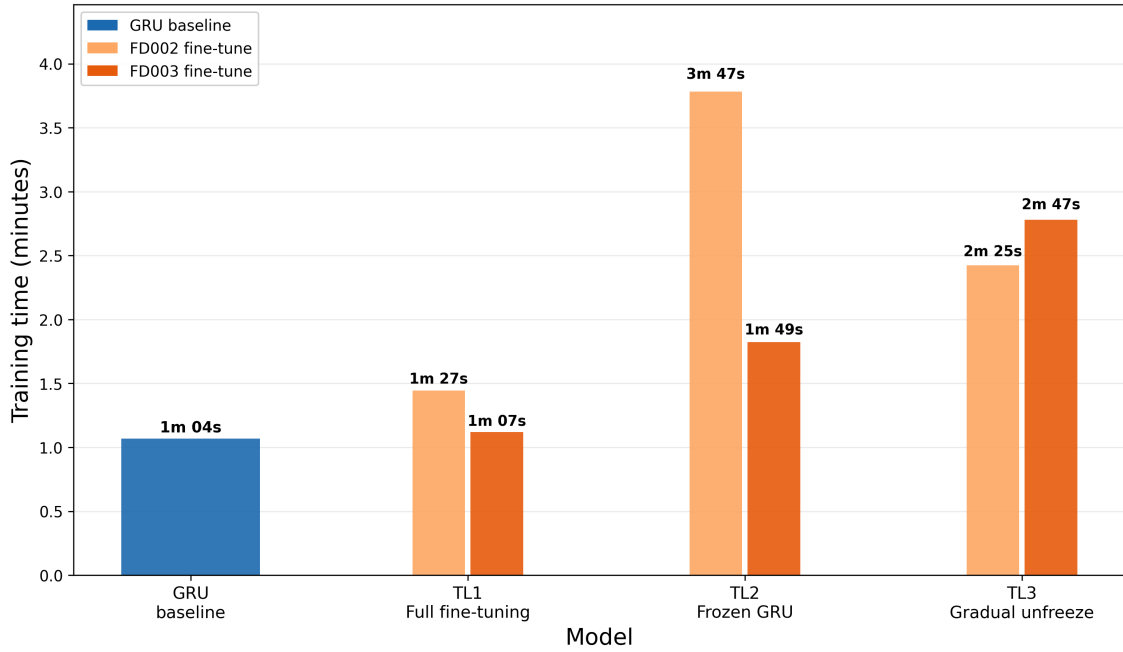


Figure 5.13: Training times of the fine-tuning strategies VS the GRU baseline.

The validation RMSE values of the resulting full-data models are shown in Table 5.8.

Table 5.8: Validation RMSE of GRU fine-tuning strategies using the full FD001 training set.

Model	Val RMSE FD002 to FD001	Val RMSE FD003 to FD001
TL1 full fine-tuning	13.58	12.97
TL2 frozen GRU	19.58	14.99
TL3 gradual unfreezing	13.54	13.06

It is evident from Table 5.8 that neither of the different fine-tuning strategies is able to improve the validation RMSE of the non-transfer GRU model under the full-data setting.

The GRU model that is trained directly on FD001 (using the correlation-selected feature set) scored a validation RMSE of 11.802, whereas all validation errors of the fine-tuned models exceeded that. Therefore, if all 80 FD001 training engines are available, then there does not appear to be any direct validation advantage provided by the transferred source-domain models, and they could potentially have caused negative transfer. This is most apparent with FD002 to FD001, especially in the TL2 frozen-feature strategy. As previously noted through the use of the MMD analysis, there exists a greater domain gap between FD002 and FD001 than exists between FD003 and FD001. In this scenario, freezing the recurrent representation within the model appears to prevent the model from adapting its source-domain temporal features to the target-distribution of FD001. On the other hand, both TL1 and TL3 performed relatively better since the GRU layer is permitted to be updated during fine-tuning. However, all three validation RMSE values exceed those of the non-transfer GRU baseline when considering the full-data condition.

Although the validation results based on 80 engines did not demonstrate any evidence of improvement when fine-tuning TL is applied, subsequent validation experimentation with decreasing amounts of target-data availability shows that the relative behaviour of how the models' performances change as the amount of FD001 training data decreases. For example, [Figure 5.14](#) illustrates examples of validation RMSE values for representative target engine settings.

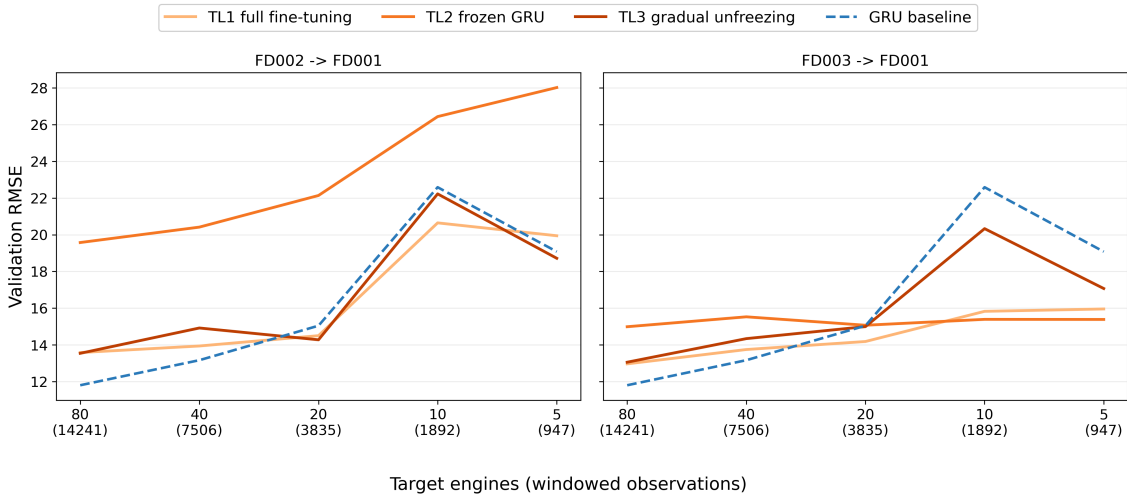


Figure 5.14: The Validation RMSE results for TL1, TL2 and TL3 and GRU baselines trained independently on engine counts of 80, 40, 20 10 and 5 scenarios

In the validation results shown by [Figure 5.14](#), it is evident that the benefit of fine-tuning strongly depends on both the quantity of available FD001 target data and the selected source domain. With 80 engines total, the GRU model trained only on FD001 target data had the lowest validation RMSE, suggesting that there is little, if any, improvement to be gained from using TL when sufficient labelled target domain data are available. However,

as the number of target domain engines decreases, the behaviour of the models relative to each other begins to change.

In the validation results shown by Figure 5.14, it is evident that the benefit of fine-tuning depends on both the quantity of available FD001 target data and the selected source domain. With 80 engines total, the GRU model trained only on FD001 target data had the lowest validation RMSE, suggesting that there is little, if any, improvement to be gained from using TL when sufficient labelled target-domain data are available. However, as the number of target-domain engines decreases, the behaviour of the models relative to each other begins to change. Since these validation results originate from the fixed-seed primary experiments, small differences and changes in the relative ranking of closely performing fine-tuning strategies should not be overinterpreted. The stability of these trends across different random seeds is assessed further in subsection 6.6.6.

This trend is evident in the FD003 to FD001 case. Although none of the three fine-tuning methods was able to perform better than the baseline at 80 or 40 engines, each method produced a lower validation RMSE than the baseline when there were 20 or fewer target-domain engines available. At ten engines, the baseline produces a validation RMSE of 22.587, whereas TL1 and TL2 show substantially lower values of 15.8280 and 15.392, respectively. A similar pattern is also present in five engine trajectories, where TL1 and TL2 remained relatively close to being competitive with the validation RMSE values of 15.957 and 15.386, respectively, as opposed to 19.0782 for the baseline. These results indicate that the benefit of TL from the FD003 domain increases as the number of labelled FD001 training data becomes more limited..

This trend is evident in the FD003 to FD001 case. Although none of the three fine-tuning methods was able to perform better than the baseline at 80 or 40 engines, each method produced a lower validation RMSE than the baseline when there were 20 or fewer target-domain engines available. At ten engines, the baseline produces a validation RMSE of 22.587, whereas TL1 and TL2 show significantly lower values of 15.8280 and 15.392, respectively. A similar pattern is also present in five engine trajectories, where TL1 and TL2 remained relatively close to being competitive with the validation RMSE values of 15.957 and 15.386, respectively, as opposed to 19.0782 for the baseline. These results indicate that the benefit of TL from the FD003 domain increases as the number of labelled FD001 training data becomes more limited.

TL benefits were not achieved with less consistency in the FD002 to FD001 setting. Both TL1 and TL3 provide an improvement over the baseline at twenty and ten engines, but these improvements are smaller and less stable than what was found in the FD003 to FD001 setting. Additionally, TL2 performs poorly throughout all FD002 to FD001 engine settings, with the validation RMSE values substantially higher than the baseline. This is consistent with the previous MMD analysis in which FD002 was shown to have

a greater difference from FD001 than FD003 did. Since TL2 fixes the pre-trained GRU layer and only adapts the prediction head, it is more dependent upon the source domain representation. Therefore, when the source-target domain gap is larger, TL2’s fixed representation appears to be less suitable for adapting to FD001.

5.5 Source-Only Transfer and Adversarial Domain Adaptation Implementation

The previous sections have covered supervised learning techniques, which are designed under the assumption that there are available lab data in the target domain. However, as discussed ([subsection 1.1.2](#)), practitioners do not always have the ability to use labelled data. This poses a significant challenge if the goal is to perform PdM but the only data that is available is the sensor data with no run-to-failure history.

In cases such as this, it is necessary to use domain adaptation techniques, which involve finding a labelled data set in a similar domain and applying unsupervised learning techniques to map the source labels (RUL) to the target. The simplest approach to do this is to simply train a model in the source domain and then use the pretrained model to perform inference on the target dataset (source-only TL). Therefore, to implement the source-only TL strategy, the same baseline GRU architecture is used, and the source datasets are pretrained separately (FD002 and FD003). The pretrained model is then subsequently used to perform RUL predictions on FD001.

However, this approach is not always sufficient, especially if the domain distribution gaps between the datasets are large. Therefore, more advanced techniques such as DANN can be required to utilise the available unlabelled target sensor data for training and to jointly train the model on labelled source domain data. The implementation of DANN is more complex than the source-only TL, as it requires multiple components including a shared FE, a label prediction branch, and a domain classifier branch with a GRL (as discussed in [subsection 2.4.3](#)). To implement DANN, this work uses the LSTM-DANN architecture proposed by Da Costa et al. (2020). Additionally, since the GRU-based models have shown promising validation performance in supervised experiments, a GRU-DANN variant is also implemented with the same architectural configuration.

It should further be noted that DANN requires a differentiable feature extractor that can be jointly optimised for both RUL prediction and domain discrimination through adversarial training. Therefore, the previously evaluated linear and tree-based models were not directly suited to the standard DANN framework. Among the neural-network models, the LSTM-DANN architecture of (Da Costa et al., 2020) was adopted to provide a literature-established implementation specifically developed for C-MAPSS RUL prediction. In addition, the GRU, which performed strongly in the preceding supervised investigations, was

incorporated as an alternative feature extractor within the same DANN framework.

The implementation of the model architecture is summarised in Figure 5.15.

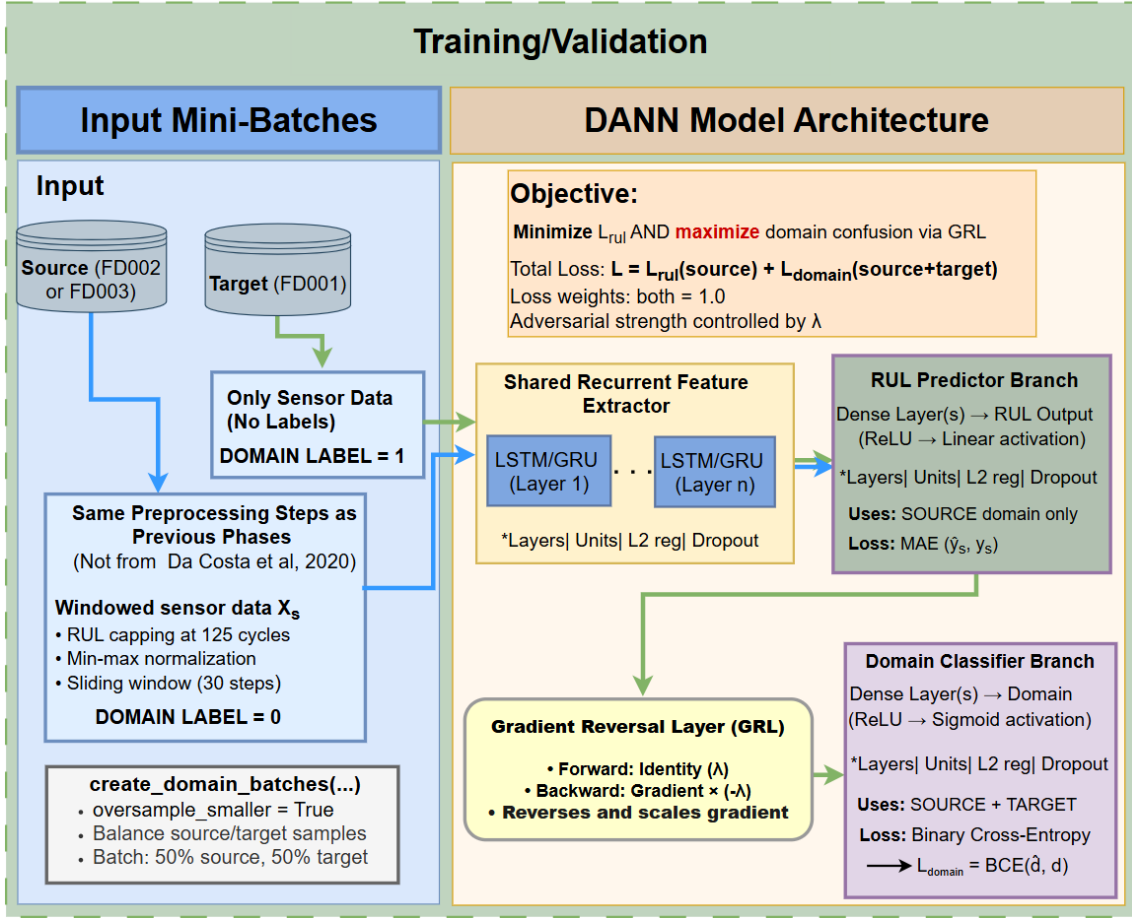


Figure 5.15: Implementation of DANN.

5.5.1 Input Mini-Batch Preparation

The procedure used to preprocess the data is the same as in the previous phases. This includes the RUL capping at 125 cycles and standardisation. In addition, since DANN consists of a neural-network-based FE, RUL predictor, and domain classifier branch, the sequenced datasets (with a sliding window of 30) are used. The target domain training data are subsamples of varying engine counts (1 to 80 engines) in order to evaluate the performance across different levels of target domain representation.

Following the unsupervised domain adaptation procedure established by Da Costa et al. (2020), the training data consists of a fully labelled source domain (FD002 or FD003) and an unlabelled target domain (FD001). During training, mixed-domain mini-batches are constructed using a function for the code implementation, which randomly samples

from the source and target training sets (refer to [section D.2](#) for the batch generator code implementation). This is done with oversampling applied to the smaller dataset to balance batch composition. Each sample is assigned a domain label (0 for source, 1 for target) for domain classification. The source samples (FD002 or FD003) retain the RUL labels for supervised learning. Since Keras expects each model output to receive a target array with the same batch dimension as the input batch, the target training samples are assigned dummy RUL labels with zero sample weights to ensure that they do not contribute.

5.5.2 DANN Model Architecture

The architecture consists of three main components, namely, the shared recurrent (LSTM or GRU) FE, a RUL predictor branch, and a domain classifier branch. In [section D.3](#) the code implementation segment for these three components is provided for reference. The shared FE consists of LSTM units that process the windowed sensor data from both domains. This component aims to learn invariant representations through adversarial training. In line with Da Costa et al. (2020), the hyperparameter configuration of the recurrent layer depends on the source data set used (refer to the Recurrent Layer in [Table 5.9](#)). If the source is FD002, then the FE consists of a recurrent layer of 64 units with a dropout of 0.1. However, if source FD003 is used, then the model becomes two layered with 64 and 32 units, respectively, and a dropout of 0.3. This is followed by a dense layer of 64 (for FD002) or 128 units (for FD003). The RUL predictor branch consists of one or two fully connected layers (depending on the configuration and the source dataset used) and performs regression using only sample samples from the source domain. The domain classifier branch of GRL attempts to distinguish between the source and target domains.

The GRL has a hyperparameter λ that determines the strength of contribution of the domain classifier during adversarial training. The implementation of GRL is also added in the appendix ([section D.4](#)). To implement it, TensorFlow’s custom gradient function is used, which enables the customisation of how the gradients are computed during backpropagation. Gradients are defined by multiplying $-\lambda$ so that in backpropagation the gradient is flipped.

The total loss function combines two components with equal weighting (refer to [Equation 5.6](#)).

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{RUL}}(\text{source}) + \mathcal{L}_{\text{domain}}(\text{source} + \text{target}) \quad (5.6)$$

Where $\mathcal{L}_{\text{RUL}}(\text{source})$ is the MAE computed only on source samples with non-zero sample weights, and $\mathcal{L}_{\text{domain}}$ is the binary cross-entropy computed on all samples. The adversarial strength is controlled by λ (the GRL multiplier), where λ is set to 1.0 for FD002 to

FD001 and to 2.0 for FD003 to FD001 (Da Costa et al., 2020). MAE was retained as the RUL regression loss for the DANN implementation because (Da Costa et al., 2020) explicitly evaluated both MAE- and MSE-based regression losses and reported that MAE produced the best-performing DANN results. RMSE is nevertheless retained as the principal evaluation metric in the present study to maintain consistency with the preceding analyses

5.5.3 DANN Training Implementation

As shown in Figure 5.16, the training procedure follows the typical ML pipeline. First, mixed-domain mini-batches are sampled from the source and target training sets. A forward pass is then executed through the DANN model, which computes the RUL prediction loss (source samples only) and the domain classification loss (all samples). Backpropagation is then performed (with the GRL reversing gradients from the domain classifier to the FE). After each epoch, the model is validated using the source validation set to monitor the RUL prediction performance. Training stops when early stopping is activated (patience = 15 epochs without improvement) or the maximum epoch limit (200 epochs) is reached. Once training has ended, the final evaluation is performed on the target test set.

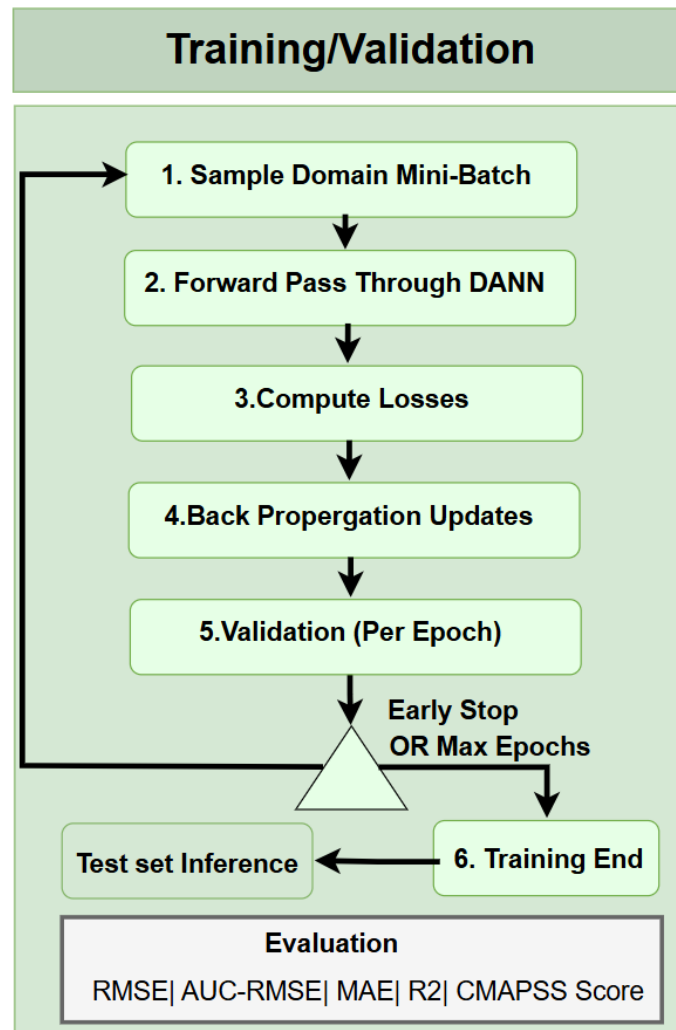


Figure 5.16: Training procedure implementation of DANN.

Table 5.9: DANN hyperparameter configurations for the FD002 to FD001 and FD003 to FD001 transfer settings based on Da Costa et al. (2020).

Hyperparameter	FD002 to FD001	FD003 to FD001	Reference
Feature Extractor			
LSTM layers	1	2	Table 3
LSTM units	64	64, 32	Table 3
LSTM dropout	0.1	0.3	Table 3
Dense units	64	128	Table 3
RUL predictor			
RUL layers	1	2	Table 3
RUL units	32	32, 32	Table 3
RUL dropout	0.0	0.1	Table 3
Domain classifier			
Domain layers	2	2	Table 3
Domain units	16, 16	32, 32	Table 3
Domain dropout	0.1	0.1	Table 3
Training			
Adversarial weighting factor (λ)	1.0	2.0	Table 3
Batch size	512	256	Table 3
Learning rate	0.01	0.01	Table 3
Maximum epochs	200	200	Table 3
L2 regularisation	0.01	0.01	Table 3
Gradient clipping	1.0	1.0	Section 5.1
Optimiser	SGD	SGD	Table 3
Momentum	0.0	0.0	Assumed
Early-stopping patience	15	15	Assumed

5.5.4 Validation of the DANN model

The training times for the DANN variants under these configurations, as well as the baseline source-only GRU, are presented in Figure 5.17. Since all models were trained with early stopping (patience equal to 15), the computational efficiency is also shown through the time per epoch, which normalises for the different convergence points across models for comparison.

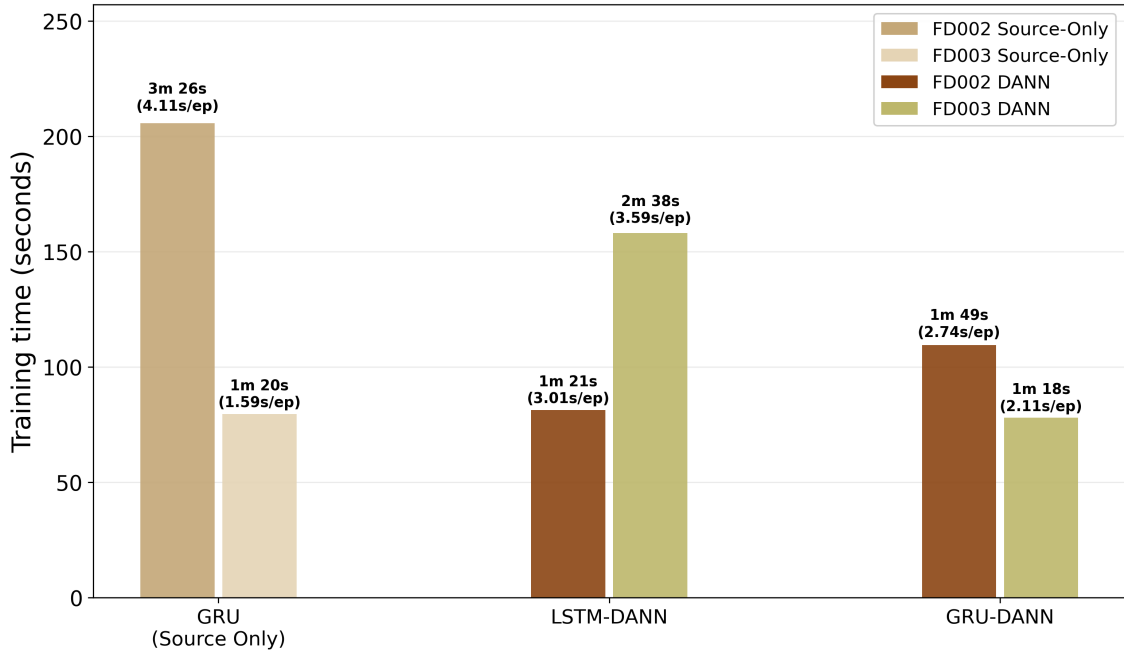


Figure 5.17: The training times of LSTM-DANN, GRU-DANN as well as the source-only GRU baseline.

All models were trained relatively quickly, with total training times ranging from 1 m 18 to 3 m 26 s. The GRU source-only model trained on FD002 took the longest at 3 m 26 s with an efficiency of 4.11 s per epoch. Moreover, the FD003 source-only model was significantly faster at 1.59 s per epoch. Furthermore, it is observed that GRU-DANN demonstrates the best computational efficiency, with training times ranging from 1 m 18 s to 1 m 49 s and per-epoch times between 2.11 and 2.74 s for both source datasets.

The extent to which adversarial training was able to trick the classifier (to find domain-invariant representations) is shown in [Figure 5.18](#). This is specifically shown through the domain training classifier accuracy (represented by the red line and scaled by the left Y-axis). The grey stippled line represents the chance-level domain classification with an accuracy of 0.5. It should be noted that this level is used as a diagnostic indication of domain confusion and not as the stopping criterion, as the domain-classifier accuracy may fluctuate around this value during adversarial training. The plot further shows the validation RUL error (orange line scaled by the right Y-axis) and the source training RUL error (blue line also scaled by the right Y-axis). The decreasing validation RUL error indicates an improvement in generalisation to the target FD001. Early stopping was based on the validation RUL error, where training stops if no improvement was observed for 15 consecutive epochs. The source training RUL error is observed to increase during the later training epochs, reflecting the trade-off between fitting the source-domain RUL task and learning domain-invariant feature representations through adversarial training.

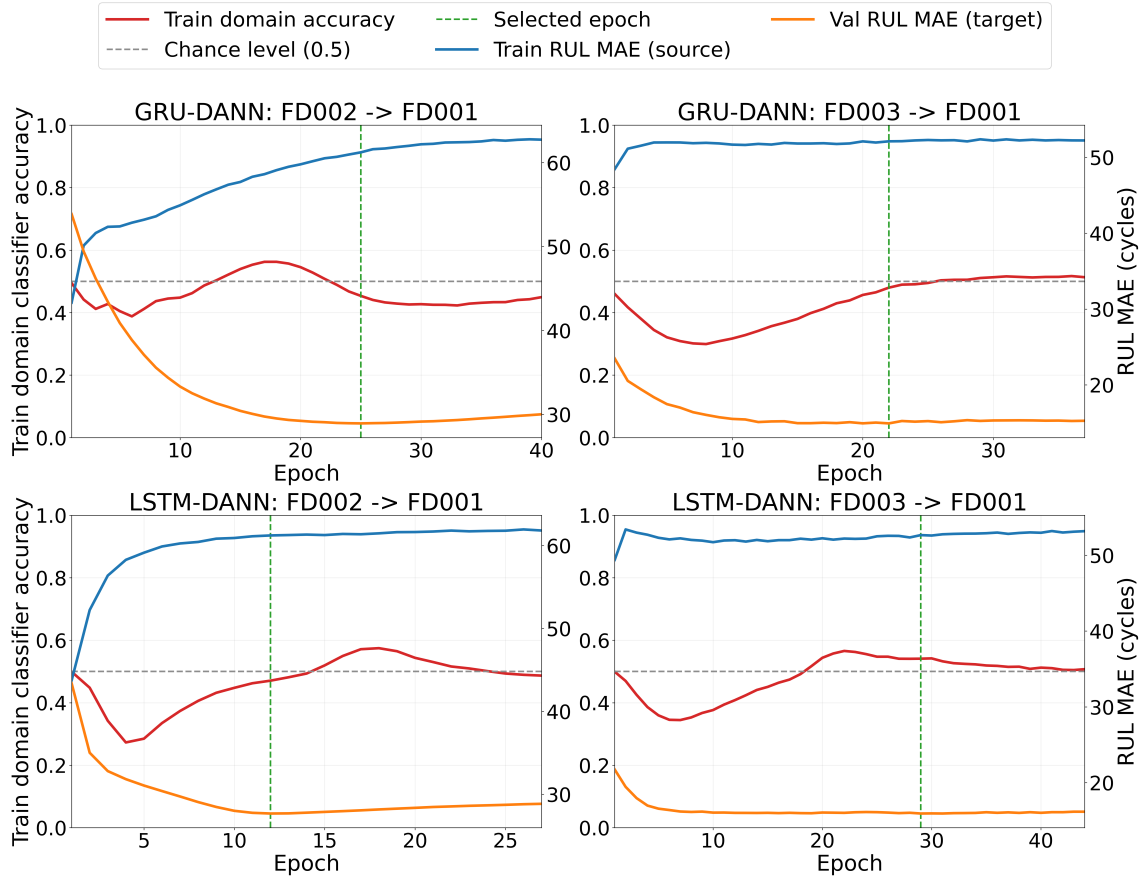


Figure 5.18: Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 80 unlabelled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).

For further reference, similar training curve plots can be referred to in [section D.1](#) for the scenarios where trajectories of 80, 40, 20 and 10 unlabelled target engines are used. To present the validation scores, [Figure 5.19](#) compares the validation RMSE values of the DANN variants against the supervised and source-only baselines using 80, 40, 20, 10 and 5 target engines.

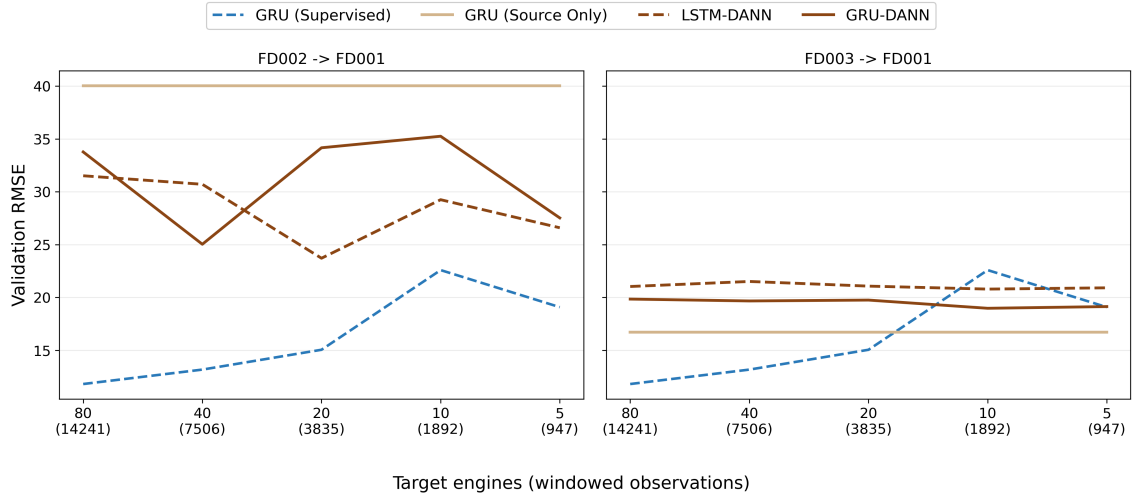


Figure 5.19: Validation RMSE performance across varying target engine counts for domain adaptation from FD002 and FD003 to FD001, comparing supervised learning, source-only transfer, and DANN variants.

For the FD002 to FD001 transfer case, the source-only GRU has a high validation RMSE. This line remains constant due to the fact that the model is trained only on the source domain and no target-domain engines. The source-only model performed poorly compared to the DANN models with a validation RMSE of 40. This provides evidence that due to the large domain gap between the source and the target, a domain adaptation strategy is required. The two DANN variants of LSTM-DANN and GRU-DANN significantly decrease the RMSE for validation compared to the source-only model, indicating that the training objectives are generally beneficial in this scenario. In a larger number of target engines, the two DANN variants have similar performance. However, GRU-DANN outperforms at medium engine counts of 20 and 10, and at 5 engines, each variant performs similarly. Notably, decreasing the number of target training data does not cause substantial performance degradation for the DANN models as the two variants are fairly consistent over the range evaluated, while it appears that the performance of the LSTM-DANN variant is slightly better at the low end (less than 20 engines).

For the FD003 to FD001 case, the source-only GRU achieves the best validation performance among the unsupervised transfer methods. This is likely due to the small domain gap between the two datasets, making direct source-only transfer effective without adversarial adaptation. It is also notable that the source-only and GRU-DANN methods remain competitive with the supervised GRU baseline across most target engine counts and even outperform it at lower data scenarios (5-10 engines). This provides evidence that simple source-only transfer from a closely related domain may be competitive with supervised learning when labelled target data is scarce.

These validation results provide evidence that the effectiveness of the transfer depends

on the selected source domain and the extent of the source-target domain gap. Since the results presented here are based on the validation RMSE, they are used only to assess the behaviour of the implemented source-only and DANN models. The final comparison of model performance is therefore presented in the next chapter using the held-out test set.

Chapter 6

Results and Discussion

This chapter presents the final evaluation of the selected models on the held-out test set, which consists of 100 engine units with partial degradation trajectories. Unlike the training set, these engine trajectories experience only partial degradation (thus not necessarily reaching EOL). This subsequently simulates the real-world scenario in which predictions must be made at arbitrary points in their life cycles without the knowledge of when the failure will occur. Through an analysis of the test results for the implementations of baseline, feature engineering (FS and extraction), and transfer learning (supervised and unsupervised), this chapter aims to address the research questions stated in [subsection 1.1.5](#).

The chapter subsequently consists of eight sections. [section 6.1](#) evaluates the performance of a variety of ML models trained on the FD001 full feature dataset. [section 6.2](#) examines how the models perform after the application of feature engineering methods, including both FS and FE. [section 6.4](#) presents the results of the supervised transfer learning strategies. [section 6.5](#) evaluates the performance of the source-only TL model as well as the unsupervised DANN models. Finally, in [section 6.6](#) the research questions stated in [subsection 1.1.5](#) are discussed.

Moreover, as discussed in [Chapter 4](#), all the models were trained on reduced subsets of the FD001 training data (from 80 engines down to 1 engine) with a validation set of 20 engines. Each of these experiments was treated as an independent scenario of training data availability, and the models were trained from scratch. To consolidate findings, test results are grouped and averaged according to five data availability ranges: high-data (80-40 engines), moderate-data (40-20 engines), low-data (20-10 engines), very-low-data (10-5 engines), and extremely limited-data (5 engines). This table presents the averaged RMSE, AUC-RMSE, MAE, C-MAPSS score, and R^2 . In addition, in each section further analysis is then made on representative models using the AUC-RMSE metric to investigate the models' prediction reliability and performance throughout the engines life cycles at the varying data availability scenarios.

It should also be noted that the results presented in this chapter are based on the fixed random seed used. Since model performance can vary as a result of training-data selection and stochastic model training, small differences between closely performing models should be interpreted with the consideration that there is inherent variability between seeded runs. A supplementary random-seed sensitivity analysis using five independent seeds is presented in [subsection 6.6.6](#) to assess the stability of the main observed trends.

6.1 Baseline Model Performance Under Data Scarcity

The baseline model phase involves seventeen models from three different ML families. The classical regression models include four linear models, namely MLR, Lasso, Ridge regression, and elastic net, as well as the non-linear polynomial regression models of the second and third degree. The tree-based models included RF, XGboost, and LightGBM. The neural-network-based models include ANN (dense feed-forward), RNN, LSTM, BiLSTM, GRU, CNN, TCN, and Transformer. [6.1](#) presents the RMSE, AUC-RMSE, MAE, C-MAPSS score, R^2 and training time. In addition, it includes an R-Rank and A-Rank column which represents the relevant rank the models place in terms of RMSE and AUC-RMSE. In response to [R3](#), the difference column (Diff) captures how each model’s ranking position changes when evaluated by AUC-RMSE instead of RMSE. Negative values (green shading) indicate models that rank better under AUC-RMSE, while positive values (red shading) indicate models that rank worse, and yellow represents no position change. In addition, the table is organised by five data ranges and is ordered by RMSE rank within each range.

Table 6.1: Baseline model test performance averaged across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the model rankings according to RMSE and AUC-RMSE, respectively, while Diff indicates the change in ranking when evaluated using AUC-RMSE instead of RMSE (negative values indicate an improved rank, positive values a decreased rank, and zero no change). Time(s) denotes model training time in seconds.

80 to 40 Engines (High Data)									
Model	RMSE	R-Rank	AUC-RMSE	A-Rank	Diff	MAE	C-MAPSS	R^2	Time(s)
GRU	13.68	1	13.29	1	0	9.93	3.48e+04	0.79	1522.2
Transformer	14.34	2	14.21	4	+2	9.43	4.61e+04	0.77	38.3
LSTM	14.55	3	13.56	2	-1	10.65	4.22e+04	0.76	1474.6
BiLSTM	14.85	4	13.87	3	-1	11.30	4.37e+04	0.75	1639.0
CNN	16.33	5	15.85	5	0	12.79	5.53e+04	0.70	27.1
XGB	17.13	6	22.91	10	+4	11.86	2.47e+05	0.61	2.3
RF	17.15	7	22.44	9	+2	12.49	1.79e+05	0.61	13.3
LGBM	17.16	8	22.92	11	+3	11.94	2.45e+05	0.61	1.3

ANN	17.18	9	22.10	8	-1	12.34	2.13e+05	0.61	22.3
RNN	17.26	10	20.48	7	-3	12.90	1.42e+05	0.66	1314.6
Poly2	19.04	11	24.18	12	+1	14.60	2.37e+05	0.52	0.3
Poly3	19.53	12	24.94	17	+5	14.45	4.28e+05	0.50	9.2
TCN	19.90	13	20.31	6	-7	15.45	1.57e+05	0.55	331.1
MLR	20.67	14	24.50	14	0	16.56	1.35e+05	0.44	0.0
Ridge	20.67	15	24.50	13	-2	16.56	1.35e+05	0.44	0.0
ElasticNet	20.67	16	24.50	15	-1	16.56	1.35e+05	0.44	1.4
Lasso	20.67	17	24.50	16	-1	16.56	1.35e+05	0.44	1.4
40 to 20 Engines (Moderate Data)									
GRU	15.08	1	14.81	2	+1	11.09	5.03e+04	0.74	1191.7
Transformer	15.74	2	15.63	4	+2	11.43	5.70e+04	0.72	26.7
LSTM	16.50	3	15.20	3	0	12.14	6.78e+04	0.69	862.7
ANN	17.46	4	22.11	6	+2	12.58	2.08e+05	0.60	13.7
RF	17.47	5	22.59	7	+2	12.72	1.85e+05	0.60	6.9
XGB	17.54	6	23.36	10	+4	12.06	2.95e+05	0.60	1.6
LGBM	17.65	7	23.35	9	+2	12.20	2.83e+05	0.59	1.0
CNN	18.16	8	18.82	5	-3	14.22	1.09e+05	0.62	25.4
BiLSTM	18.30	9	14.60	1	-8	14.27	6.32e+04	0.62	1091.5
Poly2	19.15	10	24.99	15	+5	14.57	2.62e+05	0.52	0.2
MLR	20.69	11	24.70	12	+1	16.58	1.35e+05	0.44	0.0
Ridge	20.69	12	24.70	11	-1	16.58	1.35e+05	0.44	0.0
Lasso	20.69	13	24.70	14	+1	16.58	1.35e+05	0.44	0.0
ElasticNet	20.69	14	24.70	13	-1	16.58	1.35e+05	0.44	0.0
Poly3	21.18	15	27.38	16	+1	15.25	4.64e+11	0.41	7.9
TCN	21.93	16	23.18	8	-8	17.26	2.83e+05	0.45	333.1
RNN	38.32	17	32.10	17	0	33.28	3.10e+07	-1.67	802.3
20 to 10 Engines (Low Data)									
Transformer	17.56	1	19.42	4	+3	13.04	9.70e+04	0.65	23.8
GRU	18.17	2	18.64	3	+1	13.48	9.16e+04	0.62	696.4
ANN	19.04	3	21.30	6	+3	14.28	1.89e+05	0.52	10.6
LSTM	19.20	4	16.86	1	-3	13.63	9.87e+04	0.58	720.1
RF	19.58	5	21.23	5	0	14.45	1.58e+05	0.49	3.5
XGB	19.72	6	21.97	7	+1	13.95	2.56e+05	0.49	1.2
LGBM	19.89	7	22.00	8	+1	14.10	2.42e+05	0.48	0.9
Poly2	20.64	8	22.92	14	+6	16.14	1.79e+05	0.44	0.1
BiLSTM	21.33	9	17.90	2	-7	17.70	1.21e+05	0.48	751.2
Ridge	22.20	10	22.07	9	-1	17.82	1.24e+05	0.35	0.0
MLR	22.20	11	22.07	10	-1	17.82	1.24e+05	0.35	0.0
ElasticNet	22.20	12	22.07	11	-1	17.82	1.24e+05	0.35	0.0
Lasso	22.20	13	22.07	12	-1	17.82	1.24e+05	0.35	0.0
CNN	22.63	14	22.67	13	-1	17.63	2.19e+05	0.41	18.3
Poly3	24.24	15	29.35	16	+1	18.16	2.25e+06	0.22	6.2
TCN	28.82	16	23.69	15	-1	23.59	3.80e+05	0.04	73.8
RNN	79.87	17	57.37	17	0	73.48	1.48e+08	-6.22	151.5
10 to 5 Engines (Very Low Data)									
Transformer	18.83	1	20.72	2	+1	13.13	1.70e+05	0.60	28.5
RF	20.02	2	21.46	5	+3	14.44	1.68e+05	0.47	1.8

ANN	20.06	3	21.79	6	+3	15.08	2.24e+05	0.47	7.7
XGB	20.19	4	22.11	11	+7	13.71	2.81e+05	0.46	0.9
LGBM	20.28	5	22.26	12	+7	13.74	2.80e+05	0.46	0.8
GRU	20.62	6	21.11	4	-2	14.99	2.27e+05	0.52	594.4
Poly2	21.13	7	23.38	13	+6	16.51	2.41e+05	0.41	0.1
LSTM	21.80	8	20.22	1	-7	17.00	2.01e+05	0.46	495.1
MLR	22.18	9	21.95	9	0	17.79	1.26e+05	0.35	0.0
Ridge	22.18	10	21.95	8	-2	17.79	1.26e+05	0.35	0.0
ElasticNet	22.18	11	21.95	7	-4	17.79	1.26e+05	0.35	0.0
Lasso	22.18	12	21.95	10	-2	17.79	1.26e+05	0.35	0.0
CNN	23.96	13	25.92	15	+2	17.94	6.29e+05	0.35	12.1
BiLSTM	24.49	14	21.10	3	-11	20.09	2.21e+05	0.32	676.2
Poly3	36.99	15	44.59	17	+2	27.40	3.94e+28	-0.96	2.5
TCN	37.75	16	23.47	14	-2	32.32	4.97e+05	-0.62	40.0
RNN	48.42	17	34.64	16	-1	43.76	7.09e+07	-2.20	272.2
≤5 Engines (Extreme Scarcity)									
Lasso	23.38	1	20.86	4	+3	18.76	1.30e+05	0.28	0.0
ElasticNet	23.38	2	20.86	3	+1	18.76	1.30e+05	0.28	0.0
MLR	23.38	3	20.85	2	-1	18.76	1.30e+05	0.28	0.0
Ridge	23.38	4	20.85	1	-3	18.76	1.30e+05	0.28	0.0
XGB	23.55	5	25.16	10	+5	15.95	3.39e+05	0.26	0.7
RF	23.57	6	21.84	5	-1	16.98	1.77e+05	0.26	1.0
LGBM	23.65	7	25.12	8	+1	15.97	3.62e+05	0.25	0.5
Transformer	25.20	8	22.29	6	-2	19.62	2.27e+05	0.27	29.4
ANN	31.24	9	25.14	9	0	24.72	3.44e+06	-0.67	7.6
GRU	31.88	10	26.37	12	+2	25.49	4.47e+06	-0.32	463.3
Poly2	32.08	11	39.55	15	+4	23.95	1.01e+17	-0.68	0.1
CNN	34.78	12	25.56	11	-1	27.44	7.38e+05	-0.44	11.4
LSTM	35.43	13	27.06	13	0	27.15	9.43e+06	-0.65	480.7
BiLSTM	42.65	14	28.03	14	0	36.03	8.42e+06	-1.53	554.8
TCN	43.23	15	23.74	7	-8	36.14	1.10e+06	-1.16	37.6
Poly3	48.04	16	65.82	17	+1	34.76	4.73e+28	-2.26	0.7
RNN	76.77	17	52.86	16	-1	70.51	1.50e+08	-6.01	164.1

In the high data range the GRU model achieves the best performance in RMSE, AUC, and C-MAPSS-Score and R^2 with a score of 13.68, 13.29, 3.48×10^4 and 0.79 respectively. Its MAE score of 9.93 is in close second behind the Transformer, which achieved a 9.43 MAE. The GRU, LSTM, and BiLSTM recurrent architectures, together with the Transformer, occupy the top four RMSE positions. This suggests that models capable of representing temporal dependencies benefit most from the larger training set available in the high-data settings. Notably, the RNN ranked relatively low with an RMSE and AUC-RMSE score in 10th and 7th positions, respectively. The Transformer achieves competitive accuracy (RMSE of 14.34) while requiring only 38 seconds for training compared to GRU's 1522 seconds. The tree-based models (XGB, RF, LGBM) show moderate relative performance with similar RMSE values around 17.13 to 17.16. Moreover, the regression models all show

the exact same performance as MLR, indicating that the low regulariser hyperparameter selected through Optuna had little to no effect. These linear models' RMSE exceeded 20 RMSE scores and performed the poorest in RMSE, AUC-RMSE, MAE and R^2 . Notably the polynomial regression models both yielded the worst C-MAPSS scores. This indicates that, compared to the other models, they tend to make late predictions (since the score function heavily penalises late predictions). The RMSE and AUC-RMSE difference analysis shows that XGB, RF, and LGBM have substantial declines in ranking (Diff +2 to +4). This indicates that their errors are concentrated at specific stages in the lifecycle rather than distributed uniformly. This trend can be visualised through the AUC-RMSE curve presented later in [subsection 6.1.1](#).

As expected, when data availability is reduced, there is a notable decrease in performance between models. In the moderate data availability range, all models experienced degraded performance. GRU maintains its lead in RMSE (15.08), MAE (11.09), C-MAPSS score (5.03×10^4) and R^2 (0.74), following BiLSTM closely in AUC-RMSE, scoring 14.81. Although the BiLSTM model ranked first for AUC-RMSE (14.60), it only ranked ninth for RMSE (18.30) with a rank difference of -8. Tree-based and linear regression models were least affected by the reduction in training data. The tree-based models showed only minor increases in RMSE of approximately 1.87% to 2.86%, while linear regression models changed only 0.10%. The neural-network models, excluding RNN, showed greater sensitivity, with RMSE increases ranging from 1.63% to 23.23%. Notably, the RNN failed catastrophically with an increase in RMSE of 112.02%. Polynomial regression also shows severe performance degradation, particularly for the C-MAPSS score, increasing from 4.28×10^5 to 4.64×10^{11} indicating that in this data range it is unstable and makes anomalously late predictions.

In the low-data range, the Transformer overtakes the GRU in RMSE ranking, with an RMSE of 17.56 compared to 18.17. LSTM shows notable AUC-RMSE accuracy and achieves the best AUC-RMSE (16.86), while ranking fourth for RMSE. Tree-based models maintain their positions (ranks 5-7) and show notable resilience to data scarcity compared to the neural architectures. Furthermore, BiLSTM continues its pattern of superior AUC-RMSE performance (rank 2) despite its RMSE ranking (9). Moreover, the performance of RNN's collapses catastrophically (RMSE of 79.87 and R^2 of -6.22), indicating a complete failure to generalise the test data.

Transformer maintains a top ranking (RMSE of 18.83) and therefore demonstrates notable robustness under severe data scarcity. Furthermore, the tree-based relative RMSE performance against neural-networks increases, where RF, XGB, and LGBM rise to ranks 2, 4, and 5, respectively. LSTM and BiLSTM show notable RMSE and AUC-RMSE rank differences (Diff=-7 and -11), with BiLSTM achieving the third-best AUC-RMSE despite ranking 14th for RMSE. This difference suggests these models, while struggling with overall accuracy, maintain more consistent error patterns across life-cycle stages than their

RMSE rankings indicate. Moreover, GRU’s performance notably degrades to rank 6, and architectures like CNN and TCN show poor relative performance for both RMSE and AUC-RMSE. In addition, it can be seen that polynomial regression catastrophically fails in terms of the C-MAPSS score with a score of 3.94×10^{28} . This is an indication that the third-degree polynomial is significantly unstable at very low data settings and can make predictions that are significantly late

In the scenarios where the training data are extremely low, it is evident that the models struggle to explain the variance of the data well. Where the linear regression models find the best fit in the data out of all models and are able to explain only 28% of the variance. Moreover, under these low data settings, the results suggest that the simpler linear models are more robust. The best-performing models are the linear regression models, all scoring the lowest RMSE and AUC-RMSE values, with MLR and Ridge regression performing very slightly better in AUC-RMSE than Lasso and ElasticNet. In addition, their C-MAPSS score shows the best performance out of all models. Notably, the tree-based models perform relatively competitively with linear models, with RMSE score differences of less than 1. Of all the neural-network-based models, the Transformer remained the most robust, achieving an RMSE and AUC-RMSE rank of 8 and 6, respectively. All models ranked below the Transformer, notably, report negative R^2 values. This indicates that their predictions perform worse than a simple mean-response baseline, which suggests significantly poor generalisation under the extreme-scarcity setting.

To further visualise the robustness of each model under different levels of data scarcity, [Figure 6.1](#) is presented. The plot shows the increase in RMSE relative to the high data range.

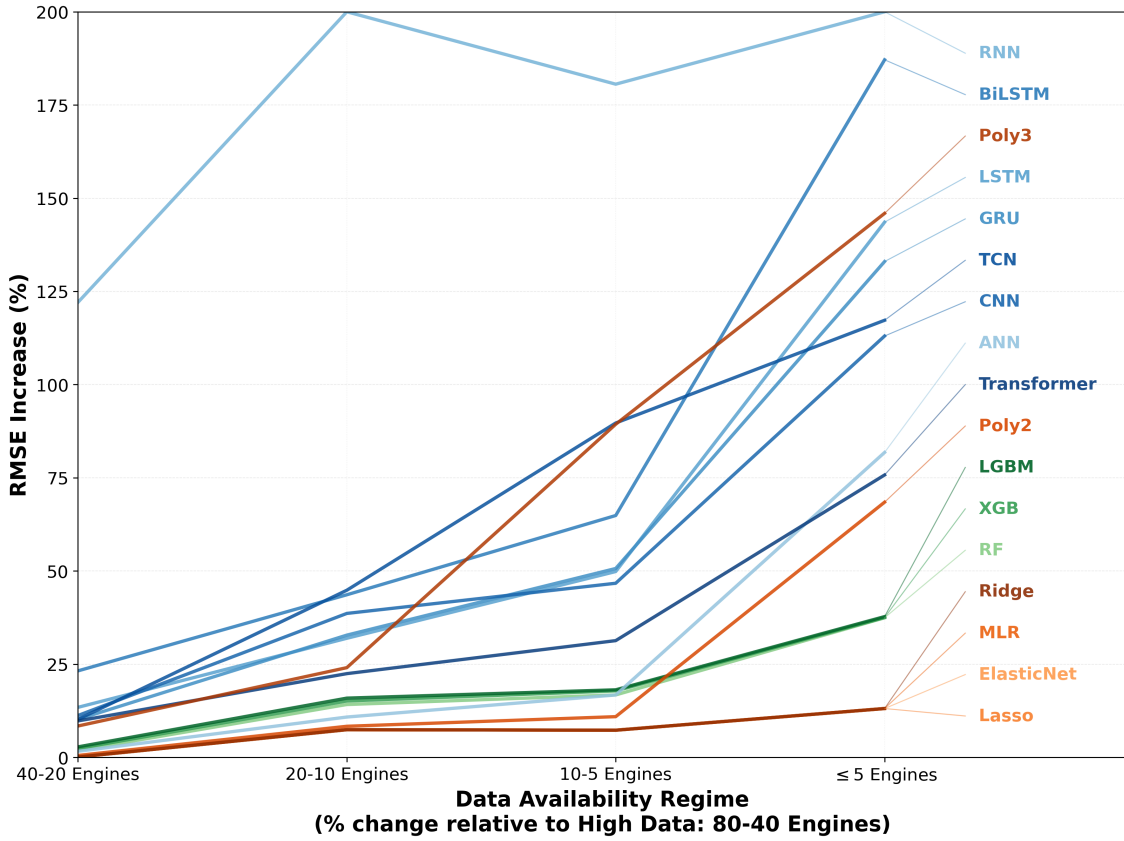


Figure 6.1: Relative RMSE degradation across data availability scenarios (including 40 to 20 engines; 20 to 10 engines; 10 to 5 engines and less than or equal to 5 engines) compared with the high-data baseline range of 80 to 40 engines.

As shown by Figure 6.1 the RMSE increases differently between model families. The RNN models are generally the most sensitive to reductions in training data. This is particularly evident for the basic RNN, which shows the largest and most unstable degradation profile across the data scarcity ranges, jumping to 122% degradation in the 40-20 engine range and exceeding 200% in the most extreme settings. The gated recurrent models also degrade as the amount of training data decreases, although less severely than the basic RNN. GRU, LSTM and BiLSTM perform strongly in the high-data setting, but their relative degradation increases notably in the very-low and extreme-scarcity settings, reaching 133%, 144%, and 187%, respectively, at 5 engines. The Transformer shows a different pattern. Although it does not maintain the lowest absolute RMSE under extreme scarcity, it shows comparatively strong robustness across the low-data ranges (76% degradation at 5 engines) and remains the strongest neural-network model when the training data is severely restricted. The tree-based models show a more gradual degradation profile. RF, XGB, and LGBM do not achieve the best absolute performance in the high-data range, but their relative increase in RMSE remains more controlled than the neural-network models as the number of engines decreases, staying below 38% even at 5 engines. Linear regression models show the flattest degradation pattern in most data scenarios. Their

absolute performance is weak in the high-data setting, but they degrade only marginally as the number of training engines decreases, remaining below 14% degradation even at extreme scarcity.

It should be noted that these extreme-scarcity scenarios refer specifically to the amount of labelled training data available, as the experimental design retained a fixed validation set of 20 labelled engines. The one- to five-engine results should therefore be interpreted with this limitation in mind, since in a real-world setting with similarly limited labelled data, it may not be possible to reserve such a comparatively large validation set for hyperparameter selection.

6.1.1 AUC-RMSE Analysis

To interpret the AUC-RMSE curves, it should be noted that this work follows the common practice of clipping RUL labels to a maximum of 125 cycles (see [section 3.3](#)). Subsequently, 30.5% of the training windows carry the maximum label of 125 cycles, representing the engines in their healthy early-life phase. Since the models are exposed to a disproportionately large number of observations at this label, they tend to predict the capped value reliably, resulting in relatively low RMSE at high RUL. Performance then degrades through the mid-lifecycle range, where training observations are scarcer, and degradation signals are less significant. The accuracy then increases near the EOL as the sensor degradation signals become increasingly significant. Moreover, since the test RUL labels are also capped at 125, the test label distribution is similarly skewed, such that metrics that weight samples equally, such as standard RMSE, are disproportionately influenced by the high-RUL observations. [Figure 6.2](#) shows the distribution of true RUL labels across the test windows.

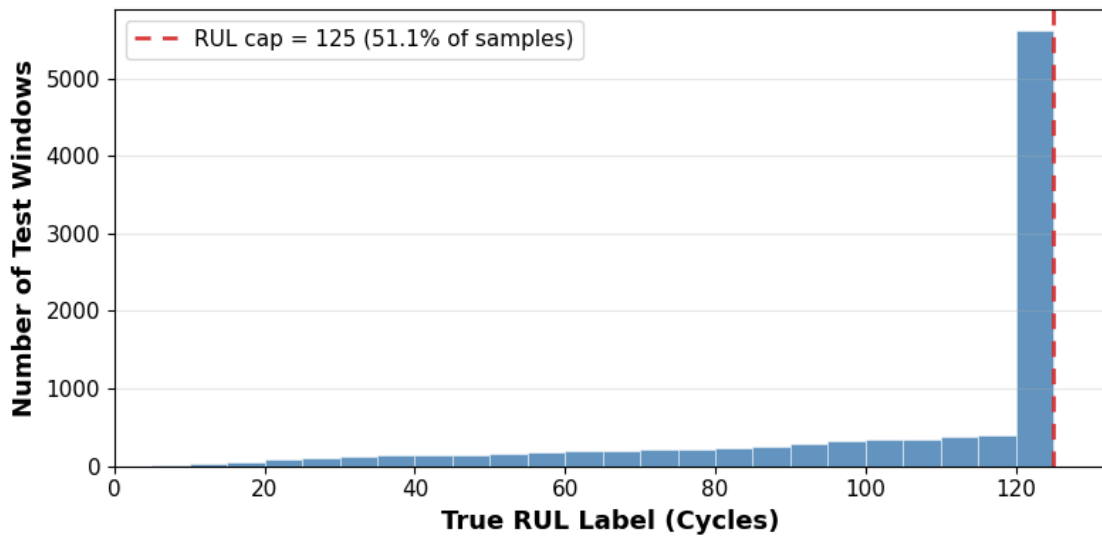


Figure 6.2: Number of test windowed observations for true RUL Label.

As can be seen, 51.1% of test windows carry the capped RUL label of 125 cycles. Furthermore, since the test set consists of partial rather than full degradation trajectories, observations become increasingly scarce at lower RUL values. This distribution further motivates the use of AUC-RMSE, which assigns equal weight to each RUL bin regardless of sample count, which subsequently ensures that EOL performance is not masked by the number of healthy state predictions.

Figure 6.3 shows the AUC-RMSE curves at 80, 40, 10 and 5 engines for GRU, BiLSTM, Transformer and MLR. These models are selected for further analysis as they achieved the top AUC-RMSE score for a certain data-setting range. It should be noted that since all the linear models achieved near-identical scores, MLR is selected as the representative. The solid curves represent lifecycle-specific RMSE values across RUL bins, while the horizontal stippled lines indicate the standard RMSE values.

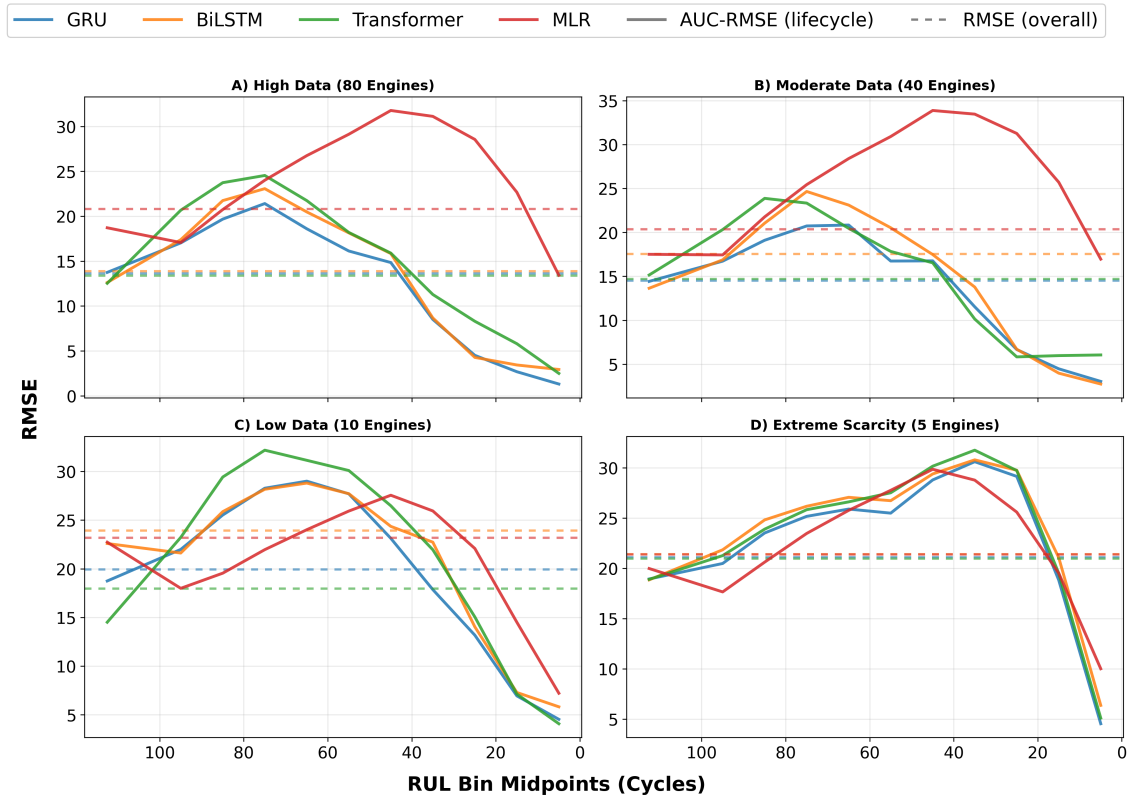


Figure 6.3: AUC-RMSE lifecycle curves (solid) and standard RMSE (stippled) for GRU, BiLSTM, Transformer, and MLR across (A) high-data, (B) moderate-data, (C) low-data, and (D) extreme-data scarcity.

Lifecycle RMSE curves reveal how model performance varies across the RUL range, particularly near EOL, for high- to low-data settings. The MLR model shows a left-skewed AUC-RMSE curve, suggesting weak predictive performance near the EOL. However, the neural-network models show more right-skewed curves that taper toward lower error values as the EOL approach. This suggests that these models provide more reliable predictions

near failure. In plots A and B (80 and 40 engines), MLR's EOL RMSE is greater than 14 whereas, GRU, BiLSTM, and Transformer show errors below 3 cycles. In Plot C (10 engines), MLR maintains its left-skewed pattern while the neural-networks show a reduced right-skewed shape but still outperform MLR near EOL.

Notably in Plot D (5 engines) all models show similar standard RMSE values (20.96 to 21.41) and their AUC-RMSE curves largely overlap. Moreover, the neural-network base models remain competitive even at 5 engines, with GRU achieving the best AUC-RMSE (22.85) despite the severe data constraint. This appears to contrast with Table 6.1, where simpler models outperform more complex architectures in the extreme-scarcity range of equal or less than 5 engines. To investigate why MLR achieved competitive average performance in the less than or equal to 5 engine range (as observed in Table 6.1), Figure 6.4 examines the AUC-RMSE degradation from 10 engines down to 1 engine.

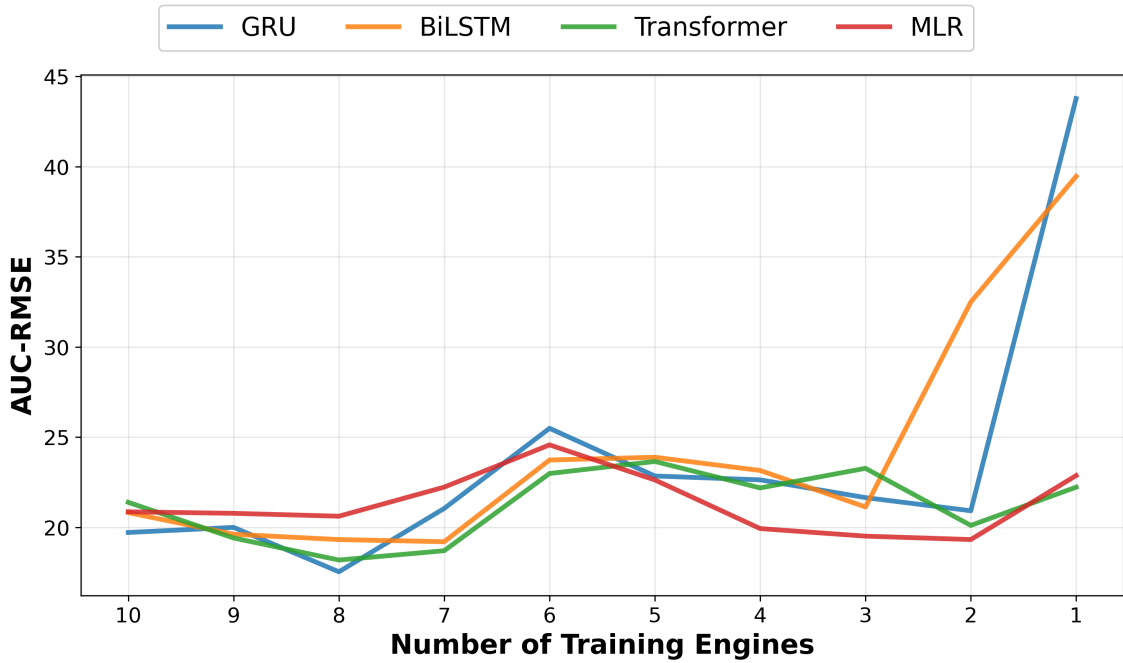


Figure 6.4: AUC-RMSE from very low to extreme data scarcity (10 to 1 training engines scenarios) for GRU, BiLSTM, Transformer and MLR.

As shown in Figure 6.4, the performance discrepancy between the GRU and BiLSTM models and MLR is mainly associated with the sharp degradation of these recurrent models in scenarios with fewer than 3 engines (576 windowed observations). Whereas the MLR and Transformer models perform more stably across all engine count scenarios.

6.2 Feature Engineering Impact Under Data Scarcity

The feature engineering techniques implemented are FS and FE. For FS, two approaches were implemented, namely, RF importance and correlation-based selection. As presented in [subsection 5.2.1](#) (correlation-based selection) and [subsection 5.2.2](#) (tree-based selection), the two approaches identified a similar subset of features, where correlation-based selection includes two additional features. Based on the validation results, the later experiments make use of the 14-feature subset. However, for comparison, this section analyses the performance of the baseline models trained on both.

The FE techniques investigated in this study were implemented as hybrid neural-network architectures. Each architecture combines a FE component with a GRU-based predictive component, where the feature extractor first transforms the input sequence into a learnt representation, and the GRU then uses this representation to predict RUL. Both components are trained jointly, so that the parameters of the feature extractor and the predictive model are optimised together during training. Since the GRU showed the strongest validation performance among the baseline models, it was selected as the predictive layer for the hybrid FE experiments. The specific architectures evaluated include CNN-GRU, TCN-GRU and AE-GRU.

6.2.1 Feature Selection on Baseline Models Results

This section presents the results of the eight selected (based on the best performers in each family in terms of validation) models trained on selected subset features. The models include Lasso and Poly2 to represent the classical regression models, RF and XGB for the tree-based models, and LSTM, BiLSTM, GRU and the Transformer for the neural-network-based models.

[Table 6.2](#) presents the percentage change in performance when FS is applied relative to the baseline (trained on all 24 features). Both scenarios of FS selected subsets are presented. Moreover, negative values indicate an improvement for RMSE, AUC-RMSE, MAE, C-MAPSS score, and training time. Whereas positive values indicate an improvement for R^2 .

Table 6.2: Percentage change in test performance after applying correlation- and RF-based feature selection relative to the corresponding full-feature baseline, averaged across five training-data scarcity ranges. For RMSE, AUC-RMSE, MAE, C-MAPSS, and training time, negative values indicate an improvement, while for R^2 , positive values indicate an improvement.

80 to 40 Engines (High Data)							
Model	FS Method	RMSE %	AUC-RMSE %	MAE %	C-MAPSS %	R^2 %	Time %
GRU	Correlation	0.9	-4.9	7.7	-5.0	-0.5	-96.6
	RF	3.7	0.1	7.9	5.7	-2.0	-96.4
BiLSTM	Correlation	-3.6	-1.5	-2.7	-13.4	2.6	-94.4
	RF	-2.5	-1.4	-3.2	-13.8	2.0	-93.1
LSTM	Correlation	-3.3	-1.1	-1.1	-4.9	2.1	-96.0
	RF	-0.2	-4.2	3.9	-12.0	0.1	-95.4
Transformer	Correlation	-3.0	-6.0	-3.9	-14.8	1.8	-12.6
	RF	-3.9	-4.7	-3.7	-15.5	2.3	-24.3
Lasso	Correlation	0.4	0.4	0.5	-1.4	-0.9	-96.4
	RF	0.5	0.7	0.7	-0.9	-1.4	-96.4
Poly2	Correlation	-0.1	-0.1	0.0	-1.1	0.2	-73.0
	RF	0.2	0.5	0.2	-0.3	-0.3	-84.5
RF	Correlation	-0.2	0.4	-1.3	7.4	0.3	8.8
	RF	0.3	1.5	-1.4	12.4	-0.3	17.8
XGB	Correlation	-0.1	0.0	-0.1	-0.2	0.1	-35.9
	RF	0.5	0.8	0.6	1.8	-0.7	-45.7
40 to 20 Engines (Moderate Data)							
Model	FS Method	RMSE %	AUC-RMSE %	MAE %	C-MAPSS %	R^2 %	Time %
GRU	Correlation	2.4	-4.4	9.3	-4.5	-1.6	-96.4
	RF	5.6	-8.4	13.9	-9.9	-4.4	-96.6
BiLSTM	Correlation	-6.6	7.5	-9.0	15.3	7.4	-93.9
	RF	-3.3	1.4	-4.4	3.0	3.3	-93.9
LSTM	Correlation	-5.0	-1.4	2.1	-17.5	4.4	-94.8
	RF	4.8	-13.9	4.6	-15.1	-4.5	-93.4
Transformer	Correlation	-0.0	-6.1	-7.1	-0.3	-0.0	-17.7
	RF	-4.0	-4.2	-13.4	-7.1	3.1	-31.8
Lasso	Correlation	0.4	0.4	0.5	-1.0	-0.9	-39.3
	RF	0.5	0.8	0.7	-0.7	-1.4	-53.9
Poly2	Correlation	-0.2	-0.3	-0.1	-1.4	0.4	-77.6
	RF	-0.2	-0.3	-0.2	-1.9	0.4	-81.0
RF	Correlation	-0.2	0.7	-1.4	10.9	0.3	6.0
	RF	0.1	1.9	-1.7	18.1	-0.2	14.9
XGB	Correlation	0.1	-0.3	0.4	-0.4	-0.1	-36.3
	RF	0.5	0.6	0.4	1.6	-0.7	-38.6
20 to 10 Engines (Low Data)							
Model	FS Method	RMSE %	AUC-RMSE %	MAE %	C-MAPSS %	R^2 %	Time %
GRU	Correlation	9.7	-2.7	20.1	23.3	-13.9	-94.6
	RF	17.1	-7.5	31.4	51.9	-23.1	-95.1
BiLSTM	Correlation	11.9	9.6	6.0	70.9	-31.8	-92.5
	RF	5.3	4.3	-4.2	46.2	-11.8	-91.4

LSTM	Correlation	2.5	0.6	19.6	-9.8	-4.4	-94.1
	RF	11.8	-2.2	16.6	18.4	-18.8	-92.7
Transformer	Correlation	0.4	-8.7	-2.7	-11.5	-0.4	-40.9
	RF	-2.7	-11.9	-13.1	-22.8	2.8	-35.0
Lasso	Correlation	0.4	0.3	0.6	-0.4	-1.7	-54.7
	RF	0.5	0.8	0.7	-0.1	-1.8	-62.5
Poly2	Correlation	-0.3	-0.2	-0.2	-2.3	0.8	-72.5
	RF	-0.6	-0.8	-0.6	-4.0	1.6	-79.7
RF	Correlation	-0.0	0.9	-1.1	12.0	-0.0	2.7
	RF	-0.0	2.2	-1.8	18.8	0.0	11.1
XGB	Correlation	0.3	0.3	0.4	4.1	-0.5	-28.6
	RF	0.5	1.2	0.3	4.1	-1.1	-32.8
10 to 5 Engines (Very Low Data)							
Model	FS Method	RMSE %	AUC-RMSE %	MAE %	C-MAPSS %	R ² %	Time %
GRU	Correlation	7.7	-2.8	15.7	38.5	-15.8	-92.1
	RF	3.4	0.5	8.5	6.6	-6.6	-91.7
BiLSTM	Correlation	2.6	0.7	3.5	4.9	-12.2	-90.8
	RF	2.4	2.2	-2.0	40.5	-14.5	-89.7
LSTM	Correlation	6.6	0.6	6.4	21.3	-18.0	-90.1
	RF	7.4	-0.9	10.8	11.2	-18.9	-90.5
Transformer	Correlation	0.1	-4.5	-2.2	-6.9	0.0	-40.7
	RF	0.2	-0.3	-2.9	24.3	-0.3	-40.9
Lasso	Correlation	0.5	0.2	0.7	-0.5	-2.0	-57.0
	RF	0.6	0.8	0.8	0.3	-2.2	-66.0
Poly2	Correlation	-0.7	-0.3	-0.6	-2.6	1.8	-50.8
	RF	-1.1	-1.2	-1.1	-8.0	3.2	-72.3
RF	Correlation	0.3	1.2	-1.2	11.7	-0.6	-3.0
	RF	0.2	2.6	-2.0	19.4	-0.5	6.3
XGB	Correlation	0.2	0.2	0.2	2.8	-0.5	-24.8
	RF	0.4	1.1	-0.0	5.0	-0.9	-29.4
≤5 Engines (Extreme Scarcity)							
Model	FS Method	RMSE %	AUC-RMSE %	MAE %	C-MAPSS %	R ² %	Time %
GRU	Correlation	42.2	5.9	58.5	81.0	-466.1	-91.2
	RF	23.2	8.5	32.0	200.5	-267.1	-89.9
BiLSTM	Correlation	1.3	0.2	2.7	18.8	-6.0	-87.3
	RF	-1.1	2.5	-0.8	-6.1	3.7	-86.9
LSTM	Correlation	14.3	-7.9	27.6	-44.8	-89.7	-89.7
	RF	26.7	6.2	37.7	14.3	-157.1	-89.7
Transformer	Correlation	-0.7	1.6	-1.3	7.7	3.0	-39.1
	RF	4.3	2.4	3.3	23.6	-32.6	-37.5
Lasso	Correlation	-0.6	2.9	-0.6	3.1	2.3	-53.4
	RF	-1.8	6.0	-1.9	7.2	8.5	-60.1
Poly2	Correlation	-20.8	-34.6	-16.7	-100.0	120.0	-54.6
	RF	-23.1	-35.0	-19.2	-100.0	127.9	-28.0
RF	Correlation	-0.7	2.7	-1.9	6.0	3.5	-11.4
	RF	-1.5	6.6	-3.8	15.1	8.4	-4.0
XGB	Correlation	-0.2	-0.5	-0.2	-4.7	1.5	-22.9
	RF	-1.5	0.3	-1.7	-5.4	9.0	-27.7

From [Table 6.2](#), it is evident that the main advantage of applying FS is the reduction in training time. This is most notable for the recurrent models, which consistently show training-time reductions of approximately 87% to 97% across the data ranges. Furthermore, all models except RF benefit computationally from FS. RF does not show the same computational benefit, and, in some cases, its training time increases, particularly in the moderate to high-data ranges, where increases of approximately 6% to 17.8% are observed. This suggests that the computational cost of RF is not necessarily reduced by lowering the input dimensionality and may be more strongly influenced by tree-construction behaviour and selected hyperparameters, such as tree depth and leaf-size constraints (see [section 2.1.2](#), which discusses the theory of RF).

However, the gain in efficiency is not representative of the predictive performance metrics. For the GRU, RMSE generally increases after FS across the high- to very-low-data ranges, with the degradation becoming especially severe under extreme scarcity. This indicates that the removed features still contain useful degradation information for the GRU. Similar degradation is visible for the LSTM in the low and very low data ranges. However, for the GRU with correlation-based FS, the AUC-RMSE improves consistently from high to very low data ranges. This suggests that although the average RMSE worsens, the reduced input representation may produce more consistent error behaviour throughout the degradation lifecycle.

The Transformer behaves differently from the recurrent models. In most high- and moderate data scenarios, FS improves RMSE, AUC-RMSE, MAE, and C-MAPSS-score, while the low- and very low data settings show smaller changes. This suggests that the Transformer is less negatively affected by the reduced feature space than the recurrent models. However, the training-time reduction for the Transformer is also smaller compared to the recurrent networks, indicating that FS provides less computational benefit for this architecture in comparison.

For the classical linear and tree-based models, the impact of FS is generally less than for the recurrent models. Lasso, RF and XGB show only marginal changes in RMSE and MAE across most data ranges. This suggests that these models are less sensitive to the reduced feature space. A notable exception is Poly2, which improves across most data ranges and improves substantially under extreme scarcity. This improvement is expected, as polynomial regression is sensitive to high-dimensional feature spaces, as discussed in [section 2.1.1](#).

Notably, the RMSE and C-MAPSS score do not always move in the same direction. Some models show worsened RMSE but improved C-MAPSS scores, indicating that FS can worsen average error performance while still reducing the type of late prediction error that is more heavily penalised by the C-MAPSS score. For example, in the 40 to 20 engine range, the LSTM model trained in the RF-selected subset shows a 4.8% increase

in RMSE but a 15.1% decrease in the C-MAPSS score.

When comparing the two FS methods, neither correlation-based nor RF-based FS is consistently superior. The correlation-based subset is less damaging for some recurrent models, while the RF-based subset gives stronger improvements for some models that benefit from dimensionality reduction, such as the Transformer and Poly2.

Furthermore, it is evident that the results do not overturn the broader observation in the baseline analysis, where recurrent models perform strongly when sufficient data are available, the Transformer remains comparatively robust under lower data availability, and simpler models become more competitive under extreme scarcity. However, there are notable differences in relative ranking performance within each of the model families, as highlighted in [Table 6.3](#).

6.2.2 AUC-RMSE Analysis for FS

[Table 6.3](#) presents a comparison of the RMSE and AUC-RMSE rankings to visualise the difference. The cases where AUC-RMSE attains a higher ranking are marked in green, while those in which it falls in rank relative to RMSE are shown in red and the cases without differences are highlighted in yellow.

Table 6.3: Comparison of RMSE and AUC-RMSE rankings for feature-selected models across five training-data scarcity ranges. R and A denote the RMSE and AUC-RMSE ranks, respectively, while D indicates the change in rank when evaluated using AUC-RMSE instead of RMSE. Negative values (green) indicate an improved rank under AUC-RMSE, positive values (red) indicate a decreased rank, and zero values (yellow) indicate no change. Corr. and RF denote correlation-based and Random Forest-based feature selection, respectively.

80 to 40 Engines (High Data)					10 to 5 Engines (Very Low Data)				
Model	FS	R	A	D	Model	FS	R	A	D
Transformer	RF	1	6	+5	Transformer	Corr.	1	1	0
	Corr.	3	4	+1		RF	2	5	+3
GRU	Corr.	2	1	-1	RF	RF	3	11	+8
	RF	5	3	-2		Corr.	4	9	+5
LSTM	Corr.	4	5	+1	XGB	Corr.	5	13	+8
	RF	8	2	-6		RF	6	14	+8
BiLSTM	Corr.	6	7	+1	Poly2	RF	7	15	+8
	RF	7	8	+1		Corr.	8	16	+8
XGB	Corr.	9	11	+2	GRU	RF	9	6	-3
	RF	12	12	0		Corr.	10	4	-6
RF	Corr.	10	9	-1	Lasso	Corr.	11	10	-1
	RF	11	10	-1		RF	12	12	0
Poly2	Corr.	13	13	0	LSTM	Corr.	13	3	-10
	RF	14	14	0		RF	14	2	-12
Lasso	Corr.	15	15	0	BiLSTM	RF	15	8	-7
	RF	16	16	0		Corr.	16	7	-9
40 to 20 Engines (Moderate Data)					<5 Engines (Extreme Scarcity)				
Model	FS	R	A	D	Model	FS	R	A	D
Transformer	RF	1	6	+5	Lasso	RF	1	2	+1
	Corr.	4	4	0		Corr.	4	1	-3
GRU	Corr.	2	3	+1	XGB	RF	2	9	+7
	RF	5	2	-3		Corr.	6	8	+2
LSTM	Corr.	3	7	+4	RF	RF	3	6	+3
	RF	7	1	-6		Corr.	5	3	-2
BiLSTM	Corr.	6	8	+2	Poly2	RF	7	10	+3
	RF	12	5	-7		Corr.	9	11	+2
RF	Corr.	8	9	+1	Transformer	Corr.	8	4	-4
	RF	9	10	+1		RF	10	5	-5
XGB	Corr.	10	11	+1	GRU	RF	11	14	+3
	RF	11	12	+1		Corr.	16	12	-4
Poly2	RF	13	15	+2	LSTM	Corr.	12	7	-5
	Corr.	14	16	+2		RF	15	16	+1
Lasso	Corr.	15	13	-2	BiLSTM	RF	13	15	+2
	RF	16	14	-2		Corr.	14	13	-1
20 to 10 Engines (Low Data)									
Model	FS	R	A	D					
Transformer	RF	1	3	+2					
	Corr.	2	5	+3					
RF	RF	3	10	+7					
	Corr.	4	9	+5					
LSTM	Corr.	5	2	-3					
	RF	12	1	-11					
XGB	Corr.	6	11	+5					
	RF	7	13	+6					
GRU	Corr.	8	6	-2					
	RF	11	4	-7					
Poly2	RF	9	15	+6					
	Corr.	10	16	+6					
Lasso	Corr.	13	12	-1					
	RF	14	14	0					
BiLSTM	RF	15	7	-8					
	Corr.	16	8	-8					

Table 6.3 further shows that FS changes the relative ranking of the selected models. Notably, the Transformer becomes the highest RMSE-ranked model across the high, moderate, low, and very-low-data ranges, with the exception of the extreme-scarcity setting. This differs from the full-feature baseline results, where the GRU achieved the strongest RMSE performance in the high and moderate data ranges. Moreover, it is observed that the FS strategy does play a role in relative performance, but there is no clear advantage in using the RF versus the correlation subset. Furthermore, although the ranking differences with respect to the findings observed in Table 6.1 are different, the evaluated neural-network models generally achieved stronger performance than the evaluated tree-based and regression models in the high- and moderate-data ranges. Furthermore, the ranking comparison also shows that the model with the best RMSE rank is not always the model with the best AUC-RMSE rank. For example, a Transformer with RF-based FS ranks first by RMSE in the high- and moderate-data ranges but only sixth by AUC-RMSE. In contrast, LSTM and BiLSTM often rank substantially better under AUC-RMSE than under RMSE, particularly in the low- and very-low-data ranges. However, RF, XGB and Poly2 show the opposite pattern in the low- and very-low-data ranges. These models rank relatively well under RMSE, however drop substantially when evaluated by AUC-RMSE. Under extreme scarcity, Lasso remains the strongest RMSE-ranked model, while XGB and RF also maintain performance above the neural-network models. The Transformer remains the strongest neural-network model.

6.2.3 Hybrid neural-network Feature Extraction Results

The hybrid models analysed consist of a GRU predictive layer. The selected FE layers are those that have shown promising results in the PdM literature (see subsection A.7.6). Subsequently, a GRU predictor with CNN, TCN and AE feature extractors is investigated to assess whether this layer adds a benefit to the GRU predictor under the different levels of data scarcity. These three models are trained on the correlation feature dataset consisting of 14 features. Table 6.4 presents the hybrid FE models' (as well as, for reference, the baseline FS GRU) average test performance across data scarcity ranges.

Table 6.4: Test performance of the CNN-GRU, TCN-GRU, and AE-GRU hybrid feature-extraction architectures compared with the GRU baseline across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the rankings according to RMSE and AUC-RMSE, respectively, and Time(s) denotes model training time in seconds.

Architecture	RMSE	AUC-RMSE	MAE	C-MAPSS	R^2	Time(s)	R-Rank	A-Rank
80 to 40 Engines (High Data)								
AE-GRU	13.45	12.34	10.00	3.19e+04	0.7951	138.9	1	1
GRU	13.81	12.64	10.69	3.31e+04	0.7838	51.8	2	2

TCN-GRU	14.72	14.19	10.14	4.80e+04	0.7547	382.4	3	4
CNN-GRU	14.91	13.76	11.30	4.60e+04	0.7483	38.2	4	3
40 to 20 Engines (Moderate Data)								
TCN-GRU	15.05	14.95	10.29	5.25e+04	0.7421	261.9	1	3
GRU	15.45	14.16	12.11	4.81e+04	0.7295	42.5	2	2
AE-GRU	15.47	13.62	11.60	4.58e+04	0.7250	105.6	3	1
CNN-GRU	16.32	15.52	12.39	6.96e+04	0.6974	34.5	4	4
20 to 10 Engines (Low Data)								
TCN-GRU	18.87	17.29	13.22	1.04e+05	0.5915	235.2	1	1
GRU	19.94	18.14	16.19	1.13e+05	0.5361	37.9	2	3
AE-GRU	20.10	17.56	15.24	1.20e+05	0.5372	87.8	3	2
CNN-GRU	23.81	18.66	20.48	1.35e+05	0.3097	27.7	4	4
10 to 5 Engines (Very Low Data)								
TCN-GRU	22.10	21.31	16.81	2.10e+05	0.4456	179.2	1	4
AE-GRU	22.20	22.04	17.01	2.82e+05	0.4419	65.0	2	3
GRU	22.21	20.51	17.34	3.14e+05	0.4365	47.1	3	2
CNN-GRU	38.80	17.43	35.10	3.20e+05	-0.7497	23.1	4	1
≤5 Engines (Extreme Scarcity)								
AE-GRU	29.86	23.66	25.53	2.36e+05	-0.0459	75.2	1	1
TCN-GRU	33.37	25.55	26.67	5.83e+05	-0.3193	193.2	2	3
GRU	45.34	27.92	40.39	8.09e+06	-1.7851	40.6	3	4
CNN-GRU	62.88	24.74	57.97	3.09e+06	-3.5886	19.4	4	2

The results show that FE can improve GRU performance. However, this depends on the feature-extraction method as well as the data availability range. In the high-data range, AE-GRU achieves the best results for all predictive accuracy metrics. It achieves a RMSE of 13.45 compared to the baseline of 13.81 GRU. Although an improvement is observed, the effect is only marginal, whereas the AE-GRU's computational efficiency is substantially worse than the GRU baseline. In particular, the other two feature-extraction methods perform worse than the baseline, with CNN-GRU achieving an RMSE of 14.91 and TCN-GRU achieving an RMSE of 14.72. However, the CNN-GRU architecture shows the best computational efficiency among the hybrid models, whereas the TCN feature extractor significantly increases training time.

In the moderate and low data availability ranges, the TCN-GRU model performs best in terms of RMSE, MAE, C-MAPSS score, and R^2 . In the moderate-data range, it achieves RMSE, MAE, C-MAPSS, and R^2 values of 15.05, 10.29, 5.25×10^4 , and 0.7421, respectively. Notably, AE-GRU ranks first in terms of AUC-RMSE in the moderate range, whereas TCN-GRU ranks third for AUC-RMSE. However, in the low-data range, TCN-GRU ranks first in all predictive metrics. Furthermore, CNN-GRU again remains the most computationally efficient model, but it produces the weakest predictive performance among the considered architectures.

In the very-low-data range, TCN-GRU continues to perform the strongest in terms of

RMSE, MAE, C-MAPSS score, and R^2 . However, CNN-GRU achieves the best AUC-RMSE despite having a substantially worse RMSE and a negative R^2 . This suggests that CNN-GRU may perform relatively well across several lifecycle regions, although its overall sample-weighted predictive accuracy is poor.

Under extreme scarcity, AE-GRU becomes the strongest model again. It achieves the best RMSE, AUC-RMSE, MAE, C-MAPSS score, and R^2 , while the baseline GRU degrades substantially. The GRU baseline has an RMSE of 45.34 and an R^2 of -1.7851, while AE-GRU achieves an RMSE of 29.86 and an R^2 of -0.0459. Although AE-GRU still does not explain the target variance strongly, its performance is substantially more stable than the baseline and the other hybrid variants. The CNN-GRU model specifically is observed to perform poorly in this setting, with an RMSE and MAE of 62.88 and 57.97, respectively, and a R^2 of -3.59. This suggests that, on average, when the training sample size is five engines or fewer, these models struggle to explain the variance and make reliable predictions. Despite CNN-GRU's poor RMSE and MAE scores, it still achieves a relatively competitive AUC-RMSE score, ranking second in the extreme scarcity scenario.

AUC-RMSE Analysis

To further investigate the performance of the FE hybrid models, this section presents the AUC-RMSE curves at specific engine counts. This includes scenarios where 80, 40 10, and 5 engine degradation trajectories were used. For comparison, the GRU, Lasso and Transformer models are plotted. All models compared were trained on the 14 selected features (identified using correlation-based FS).

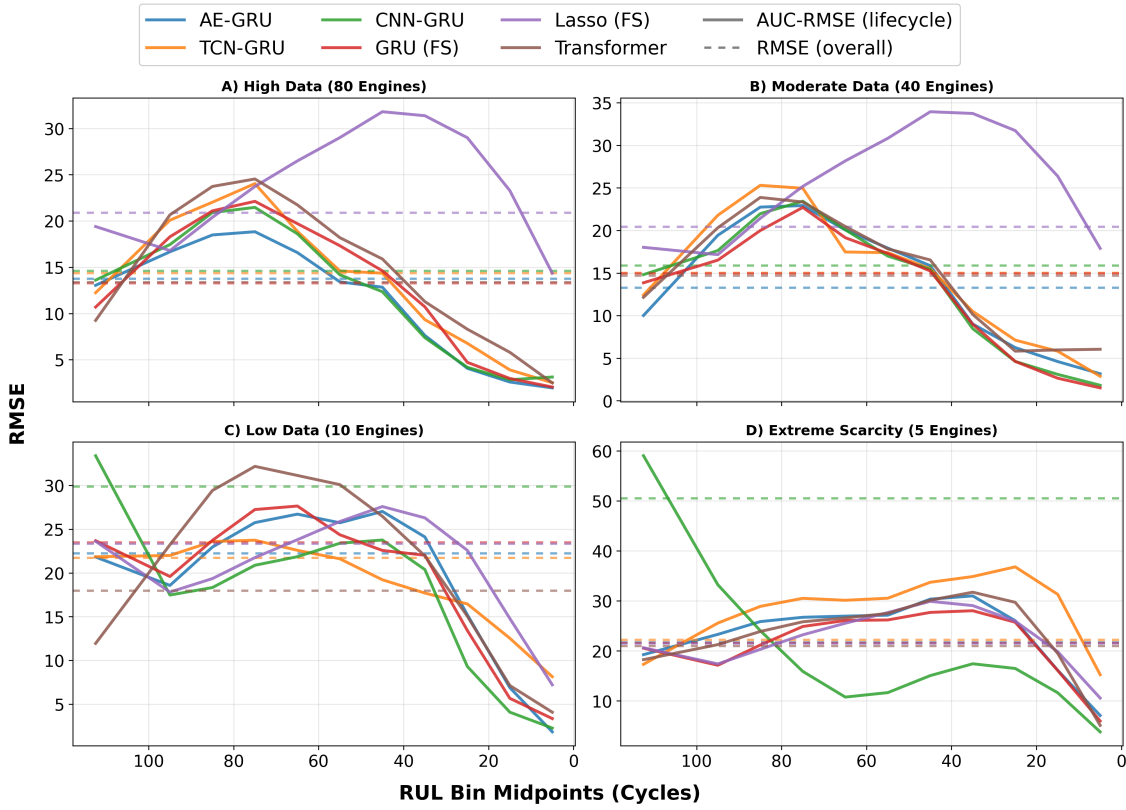


Figure 6.5: AUC-RMSE lifecycle curves (solid) and standard RMSE (stippled) for CNN-GRU, TCN-GRU and AE-GRU as well as the GRU and Lasso models trained on correlated selected features across (A) high-data, (B) moderate-data, (C) low-data, and (D) extreme-data scarcity.

Similarly to [subsection 6.1.1](#), models generally show lifecycle-dependent error patterns, where RMSE often peaks in the mid-lifecycle range before improving near the end of life. Moreover, the neural-networks show a right-skewed shape for the 80 and 40 engine scenarios, indicating that they perform best at near EOL. In the lower data scenarios, they become more left-skewed. In addition, Lasso shows the same pattern as [subsection 6.1.1](#), in which it significantly performs worse in high-level data scenarios with a left-skewed shape and becomes more competitive in lower data scenarios. Furthermore, despite CNN-GRU reporting the weakest aggregate RMSE (recall [Table 6.4](#)), particularly under extreme data scarcity, its lifecycle curve reveals a different pattern. Although it notably performs worse at the early engine lifecycles in the low data scenarios, it remains competitive across most lifecycle stages and shows particularly strong near EOL accuracy under the low- and extreme-data scenarios. This discrepancy highlights the limitation of standard RMSE, as CNN-GRU’s poor aggregate score is inflated by its large errors at the RUL cap rather than across the degradation curve as a whole.

In the 80-engine scenario (Plot A), AE-GRU achieves the strongest lifecycle-wide performance, remaining below the other models across much of the early- to mid-life range, and

performing competitively with CNN-GRU at near EOL stages. All neural-network-based models converge to very low RMSE values near EOL. At 40 engines (Plot B), the neural architectures follow an even closer trajectory through the mid-life range. The main separation occurs near EOL, where the transformer remains slightly higher while the GRU-based architectures continue to converge at lower RMSE values.

At 10 engines (Plot C), CNN-GRU reports a high overall RMSE, mainly due to its poor predictions in the healthy high-RUL region, where many test labels are concentrated due to the RUL cap discussed in [Figure 6.2](#). However, the lifecycle curve shows that this poor aggregate score does not reflect its behaviour across the entire degradation trajectory. Once the engine moves beyond the capped healthy region, CNN-GRU becomes increasingly competitive and achieves some of the lowest errors near EOL. The Transformer shows the opposite pattern, struggling more than the other models through the mid-life range, although it also converges toward a lower RMSE near failure. At 5 engines (Plot D), the neural-network models begin to take on a curve shape more similar to that of Lasso. CNN-GRU again performs poorly in the healthy high-RUL region, causing its overall RMSE to rise above 50, but it becomes the strongest model across much of the mid to late-life range. Therefore, this provides further evidence that its poor aggregate RMSE is driven by early-life prediction errors rather than uniformly weak life-cycle performance.

6.3 Comparison with Published FD001 Results

For better context for the predictive performance presented in this work, the outcomes from the current investigation were compared to other RUL prediction publications that made use of the C-MAPSS FD001 dataset. This comparison provides broader context for the predictive performance achieved in the present study relative to previously published results on the same benchmark dataset. However, the reported values should be interpreted with the acknowledgement that differences in preprocessing, input-window construction, validation procedures, hyperparameter optimisation, and test-set evaluation protocols can influence the resulting RMSE values. Moreover, limited studies made evaluations over different scenarios of data availability, and subsequently the reported values are compared based on the full 80-engine training dataset in this study, with the acknowledgement that the other studies may have used different data-split approaches.

As shown in the table, among the selected published studies included for comparison, the lowest RMSE values were reported by more specialised architectures, including the Attention TCN-BiGRU model of Gong et al. (2022) with an RMSE of 11.17 and the Transformer-based approaches of Kim et al. (2024) and Zhang et al. (2022), which reported RMSE values of 11.36 and 11.43, respectively. The AE-GRU and GRU models evaluated in the present study achieved RMSE values of 13.45 and 13.68, respectively. For additional context, other published GRU-based implementations included in the com-

Table 6.5: Contextual comparison of selected RUL-prediction architectures evaluated on the C-MAPSS FD001 dataset.

Study	Architecture	FD001 RMSE
Gong et al. (2022)	Attention TCN-BiGRU	11.17
Kim et al. (2024)	Transformer + ARR	11.36
Zhang et al. (2022)	DAST Transformer	11.43
Present study	GRU	13.68
Kim and Lee (2026)	GRU	14.268
Present study	CNN-GRU	14.91
Adryan Fitra Azyus et al. (2025)	CNN-GRU	16.29

parison reported RMSE values of 14.268 for the GRU model of Kim and Lee (2026) and 16.29 for the CNN-GRU model of Adryan Fitra Azyus et al. (2025). In terms of comparable GRU-based architectural implementations, the models evaluated in the present study achieved competitive performance, while the compared attention- and Transformer-based studies also demonstrate that more specialised state-of-the-art architectures can achieve substantially lower RMSE values on FD001.

The architectures with the lowest RMSE scores differ considerably from the less complex configurations investigated in this study. Gong et al. (2022) combined RF-based feature selection and variational mode decomposition-based signal decomposition with an attention mechanism, TCN and BiGRU to incorporate multiple levels of feature extraction and temporal modelling into one framework for predicting the RUL. The advantage of this architecture is that the individual components address different aspects of the prediction problem including feature selection and signal decomposition reduce irrelevant information and noise, while the attention mechanism identified important multi-sensor information and the TCN-BiGRU combination captured degradation behaviour across the temporal sequence. Kim et al. (2024), on the other hand, developed a Transformer-based architecture using Adaptive RUL-wise Re-weighting (ARR) to adjust the contribution of samples from different RUL regions during training. This is advantageous because the Transformer can model long-range dependencies across the multivariate time series through self-attention, while reportedly the ARR reduced the dominance of over-represented RUL regions and placed greater emphasis on regions that the model finds more difficult to predict. Zhang et al. (2022) presented the Dual-Aspect Self-Attention based on Transformer (DAST) architecture, where two parallel self-attention encoder modules were used to represent sensor-wise and time-step-wise information before fusing the extracted features to predict the RUL. This is advantageous because the architecture can independently identify both the sensors and time steps that contain the most relevant degradation information, while the self-attention mechanism enables long-term dependencies to be captured without

relying on recurrent processing.

In contrast, the GRU and CNN-GRU implementations evaluated in the present study achieved lower RMSE values than the architecturally similar implementations reported by Kim and Lee (2026) and Adryan Fitra Azyus et al. (2025), respectively. The present GRU achieved an RMSE of 13.68 compared with 14.268 reported by Kim and Lee (2026) for their conventional GRU under the capped RUL evaluation. However, the methodological emphasis of their study was primarily on the formulation of the RUL target rather than systematic optimisation of the GRU architecture. In the present study, the GRU architecture and training parameters were systematically optimised using Optuna based on validation RMSE, including the number of recurrent layers, number of units, dropout rate, learning rate, and batch size. This resulted in a single GRU layer with 256 units, a dropout rate of 0.3, a learning rate of 0.0005, and a batch size of 32. The systematic optimisation of these parameters may therefore have contributed to the competitive GRU performance obtained in the present study.

A similar distinction is evident for the CNN-GRU comparison. Adryan Fitra Azyus et al. (2025) evaluated six predefined CNN-GRU configurations by varying the number of CNN and GRU layers, while using a fixed batch size of 512, learning rate of 0.001, and maximum of 200 training epochs. In comparison, the CNN-GRU architecture in the present study was optimised using 30 Optuna trials, with convolutional depth, number of filters, kernel size, pooling, and batch size included in the search. The selected configuration consisted of one convolutional layer with 32 filters, a kernel size of 5, no pooling, and a batch size of 256. This more systematic exploration of the CNN-GRU configuration may partly explain the lower RMSE of 14.91 obtained in the present study compared with the 16.29 reported by Adryan Fitra Azyus et al. (2025). However, the studies also differ in their input formulation and preprocessing, and the performance difference should therefore be interpreted contextually rather than as evidence that one optimisation procedure is inherently superior.

6.4 Supervised Transfer Learning Results

This section addresses R4, which investigates whether supervised transfer learning strategies can use source-domain knowledge to improve RUL prediction performance when limited labelled target-domain-data are available. To evaluate this, GRU models pretrained in the source domains FD002 and FD003 were transferred to the FD001 target domain and fine-tuned using different amounts of labelled FD001 data. Three transfer learning strategies are considered: full fine-tuning (TL1), layer-freezing FE (TL2), and progressive unfreezing (TL3). Their performance is compared against the GRU baseline trained directly on FD001 using the correlation-selected feature set. Table 6.6 presents the RMSE, AUC-RMSE, MAE, C-MAPSS, R^2 and Training time. In addition, it includes the relative

RMSE rank denoted by R-Rank and the AUC-RMSE rank denoted by A-Rank.

Table 6.6: Test performance of the supervised transfer-learning strategies TL1, TL2, and TL3 using FD002 and FD003 as source domains, compared with the supervised GRU baseline trained directly on FD001, across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the rankings according to RMSE and AUC-RMSE, respectively, and Time(s) denotes model training time in seconds.

Model	RMSE	AUC-RMSE	MAE	C-MAPSS	R^2	Time(s)	R-Rank	A-Rank
80 to 40 Training Engines (High Data)								
GRU	13.81	12.56	10.69	3.31e+04	0.7838	51.8	1	1
TL3 (FD003)	14.81	15.27	11.02	4.62e+04	0.7518	130.5	2	3
TL3 (FD002)	15.26	16.41	11.17	5.83e+04	0.7364	114.4	3	5
TL1 (FD003)	15.26	14.54	11.34	4.54e+04	0.7429	60.0	4	2
TL2 (FD003)	16.61	17.19	13.19	8.00e+04	0.6848	98.0	5	6
TL1 (FD002)	16.73	15.90	11.75	4.70e+04	0.7363	77.2	6	4
TL2 (FD002)	20.54	24.72	16.36	1.40e+05	0.5188	178.0	7	7
40 to 20 Training Engines (Moderate Data)								
TL3 (FD003)	15.33	16.23	11.14	5.31e+04	0.7337	68.6	1	3
GRU	15.45	14.24	12.11	4.81e+04	0.7295	42.5	2	1
TL1 (FD003)	15.49	15.89	11.56	5.10e+04	0.7230	42.3	3	2
TL3 (FD002)	16.14	16.56	11.44	6.38e+04	0.7048	74.9	4	4
TL2 (FD003)	16.72	17.47	12.91	8.98e+04	0.6826	46.2	5	5
TL1 (FD002)	18.83	18.60	12.12	5.95e+04	0.7026	57.2	6	6
TL2 (FD002)	20.99	26.17	16.30	1.77e+05	0.4878	128.7	7	7
20 to 10 Training Engines (Low Data)								
TL1 (FD003)	16.76	16.00	12.23	7.01e+04	0.6697	19.0	1	1
TL2 (FD003)	16.98	17.12	13.46	8.28e+04	0.6667	29.5	2	2
TL3 (FD003)	18.64	17.60	13.12	8.80e+04	0.5981	42.8	3	3
GRU	19.94	18.31	16.19	1.13e+05	0.5361	37.9	4	4
TL3 (FD002)	20.32	18.52	14.83	2.18e+05	0.5206	49.0	5	5
TL1 (FD002)	20.59	21.25	14.41	1.50e+05	0.5524	35.7	6	6
TL2 (FD002)	22.07	25.33	18.23	2.44e+05	0.3419	101.9	7	7
10 to 5 Training Engines (Very Low Data)								
TL2 (FD003)	17.10	17.16	13.61	8.30e+04	0.6594	26.6	1	2
TL1 (FD003)	17.83	16.44	12.60	8.02e+04	0.6406	9.1	2	1
TL3 (FD003)	19.02	18.50	13.14	1.11e+05	0.5892	30.4	3	3
TL1 (FD002)	21.38	22.65	15.68	2.29e+05	0.4501	28.4	4	6
TL3 (FD002)	22.06	21.54	15.13	2.81e+05	0.4478	43.4	5	5
GRU	22.21	20.64	17.34	3.14e+05	0.4365	47.1	6	4
TL2 (FD002)	23.06	25.61	19.70	3.35e+05	0.2245	54.1	7	7
≤5 Training Engines (Extreme Scarcity)								
TL2 (FD003)	16.93	17.52	13.13	9.12e+04	0.6686	32.7	1	2
TL1 (FD003)	17.70	17.04	12.72	7.64e+04	0.6363	8.0	2	1
TL3 (FD003)	20.92	18.43	14.74	1.22e+05	0.5005	25.7	3	3
TL1 (FD002)	23.78	24.91	20.66	3.93e+05	0.0172	29.4	4	5
TL2 (FD002)	24.02	27.69	22.71	6.89e+05	-0.1018	43.5	5	6

TL3 (FD002)	28.71	24.21	19.93	3.80e+05	0.0475	33.2	6	4
GRU	45.34	28.30	40.39	8.09e+06	-1.7851	40.6	7	7

In the high-data scenarios, the evaluated GRU configuration performs better than the evaluated supervised TL configurations across all accuracy metrics and in training time. The TL3 approach is the top-performing TL strategy, ranking second and third in RMSE of 14.81, 15.26 for both FD003 and FD002 respectively. Notably, the TL2 strategy that involves a frozen GRU layer performs worse for all metrics when trained on the FD002 dataset (with a RMSE of 20.54) likely because the frozen GRU retains source-only representations and cannot adapt to the larger FD002 to FD001 domain shift.

In the Moderate data range, the TL3 strategy trained on the source FD003 starts to show the potential benefit. It marginally improves on RMSE, MAE and R^2 showing a score of 15.33, 11.14, and 0.73, respectively. However, GRU still scores the best AUC-RMSE score (of 14.24) and C-MAPSS-score (of 4.81×10^4). GRU is still comparatively more efficient than the other approaches, with the exception of TL1 trained on FD003 differing by only 0.2 s. Furthermore, the training times of the TL methods decrease substantially in this range, with TL1 (FD003) showing a training time comparable to the GRU baseline.

In the low data range, the advantage of using supervised TL techniques becomes more apparent. TL1 shows the greatest improvement over the GRU that performs the best in all reported metrics (scoring an RMSE of 16.76 and an AUC-RMSE of 16.00). In addition, it is observed that all TL methods significantly decreased in training time, where, for example, the GRU only decreased by 4.6 s while TL1 (FD003) decreased by 23.3 s. The low data range scores also clearly show the effects of the domain shift in terms of performance for TL techniques since all TL techniques trained on the FD002 dataset scored worse in RMSE and AUC-RMSE and C-MAPSS score compared to the GRU-baseline. TL2 (FD002) consistently ranks as the weakest TL configuration in low-data metrics, while TL3 (FD002) performs closest to the GRU baseline among FD002-based strategies and improves on the GRU in terms of MAE, achieving 14.83 compared to 16.19.

In the very low data range, all FD003 trained TL models continue to outperform the GRU baseline. TL2 ranks first for RMSE (scoring 17.10) and R^2 (scoring 0.6594) while TL1 continues to perform the best on the AUC-RMSE, MAE and C-MAPSS score, with values of 16.44, 12.60 and 8.02×10^4 , respectively. In addition, TL1 is observed to be significantly more efficient than all other strategies, taking only 9.1 s to train.

Notably, within the evaluated configurations, the supervised TL strategies show a substantial benefit over the GRU baseline in the extreme-scarcity scenarios. Recall in [Table 6.4](#) and [Table 6.1](#) (the Hybrid Feature Extraction and Baseline Model Test Performance Across Data Scarcity) it was observed that the models struggled to explain meaningful variance in the data. For instance, among the hybrid models, the best-fitting model is the AE-GRU,

which shows an R^2 of -0.0459, while among the baseline models, the linear regression model provides the best test fit with an R^2 of 0.28. In contrast, the TL approaches using the source domain FD003 dataset could all explain 50% or more of the variance, with TL2 scoring an R^2 of 0.67. Moreover, TL1 (FD003) continues to be the top performer in AUC-RMSE, MAE, C-MAPSS score and training time, with values of 17.04, 12.72, 7.64×10^4 and 8.0 s, respectively. TL2 (FD003) achieves the best RMSE and R^2 , with values of 16.93 and 0.6686, respectively. The GRU model performed significantly worse in all prediction metrics. Notably, the TL methods trained on FD002 have difficulty accounting for the variance in the data at this level, with the best of the three, TL3, achieving an R^2 of only 0.0475.

To visualise the added benefit or negative transfer when using TL approaches, Figure 6.6 presents the percentage improvement of each TL strategy compared to the baseline GRU. The figure shows the improvement in terms of RMSE and AUC-RMSE performance.

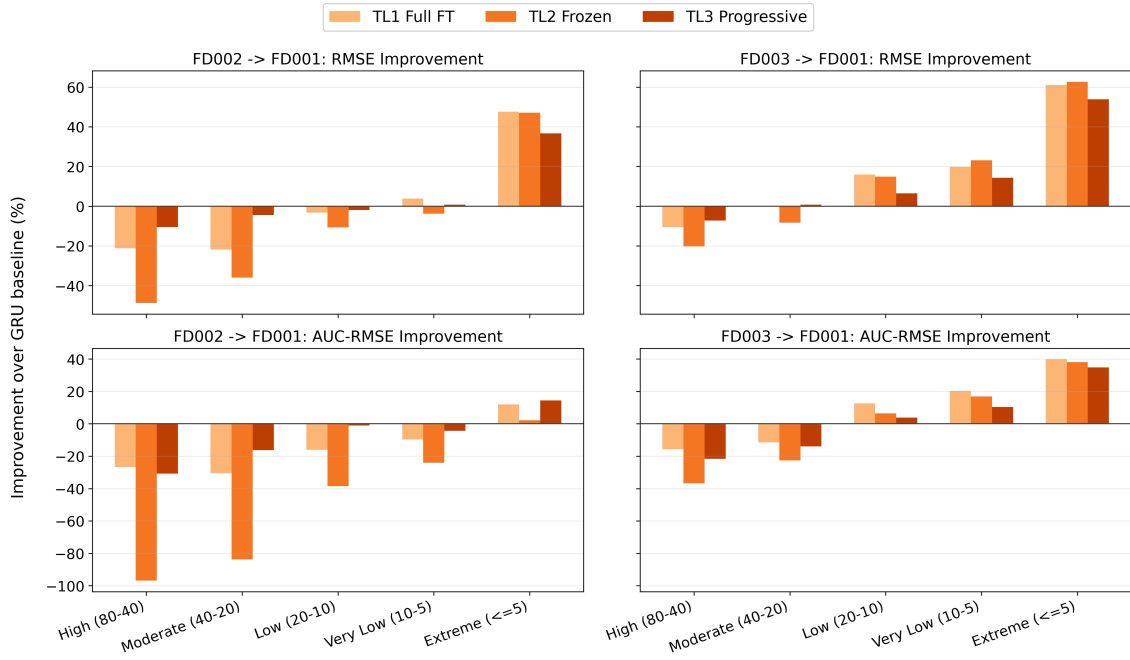


Figure 6.6: Improvement of supervised TL models over the GRU baseline, where TL1, TL2 and TL3 refer to full fine-tuning, layer freezing and progressive unfreezing strategies, respectively.

As shown in Figure 6.6, applying the proposed TL strategies in the high-data setting does not provide an additional advantage over training directly on the target domain. Instead, all TL strategies show a deterioration in performance, indicating negative transfer when sufficient labelled target data is available. This is most severe for the TL2 strategy, particularly when FD002 is used as the source domain, where performance deteriorates by 49.1% in RMSE and 97.0% in AUC-RMSE. However, it can be observed that as the data availability decreases the TL strategies become more competitive. Particularly in the

low to extreme data availability scenarios, a consistent improvement over the baseline is evident. The advantage is, most apparent for the TL strategies trained on FD003 where all three show an improvement above 50% for RMSE and above 30% for AUC-RMSE. Notably, the TL models trained on FD002 show less promise in the low- and very-low-data ranges, where improvements over the GRU baseline are smaller and less consistent. Although all FD002-based TL strategies improve RMSE by more than 35% under extreme scarcity, the AUC-RMSE improvement remains marginal, with TL3, TL1 and TL2 improving by 14%, 12% and 2%, respectively. To fully visualise the performance of the supervised TL models for all investigated data availability scenarios, Figure 6.7 plots the RMSE results for all independent engine count scenarios for TL1, TL2 and TL3 and GRU baseline.

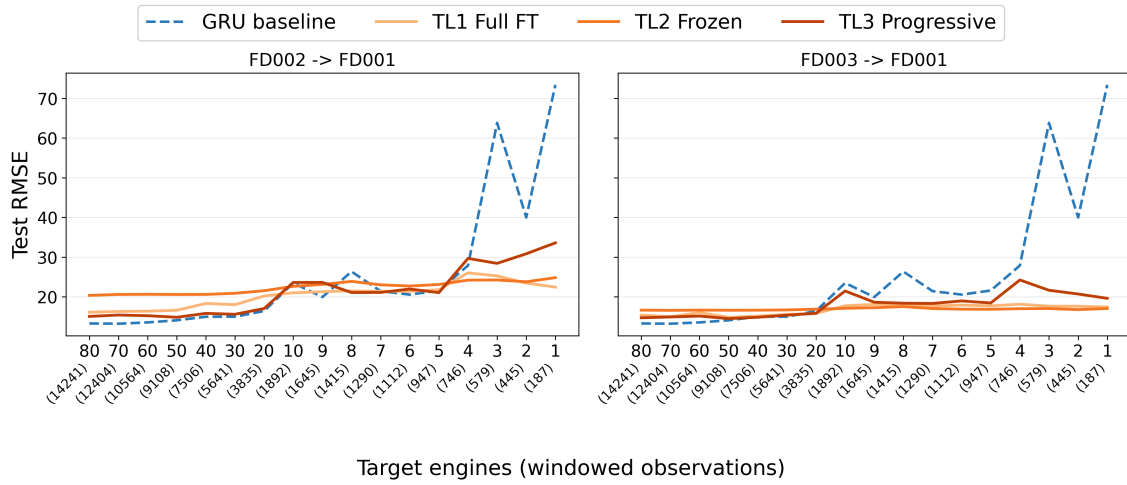


Figure 6.7: Test RMSE across target-engine-count scenarios for supervised TL strategies using FD002 and FD003 as source domains, compared with the GRU baseline

Notably, across all tested training data availability scenarios, the TL models perform significantly more consistently. The performance gain can particularly be seen when the total number of observations falls below the 4 engine trajectories (746 observations). Notably, the TL models trained on the FD003 data show a flatter/more horizontal line compared to the FD002 TL models which slowly increase as data degrades. This further suggests that FD002 models depend more on the FD001 dataset, likely because of domain discrepancy, whereas the similarity between FD003 and FD001 domains leads to more consistent model performance.

6.4.1 AUC-RMSE Analysis of Supervised TL strategies

To investigate the reliability of the model performance across unweighted life cycle RUL predictions, Figure 6.8 likewise presents the AUC-RMSE outcomes across all independent engine-count settings. It includes TL1, TL2, TL3, and the GRU baseline.

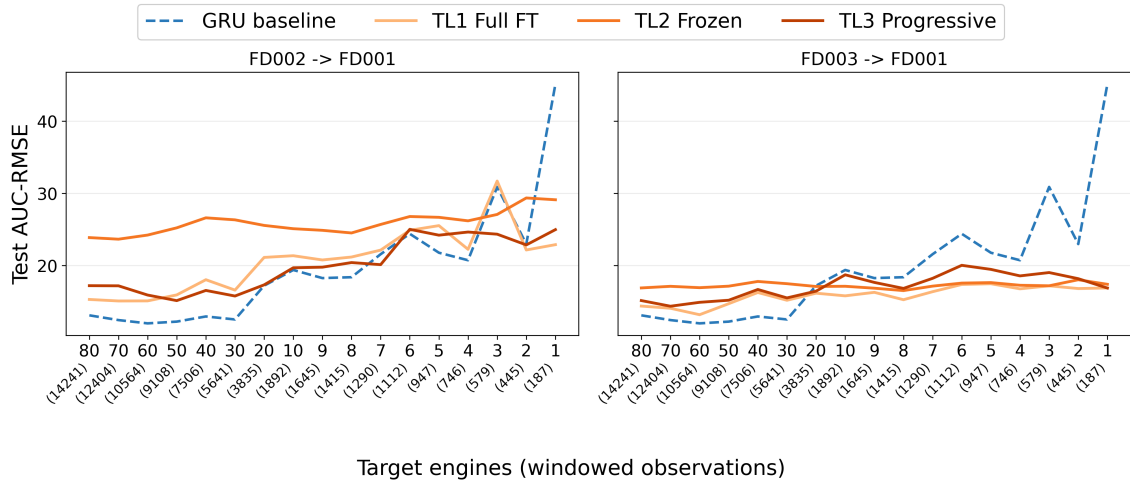


Figure 6.8: Test AUC-RMSE across target-engine-count scenarios for supervised TL strategies using FD002 and FD003 as source domains, compared with the GRU baseline.

As observed earlier in terms of AUC-RMSE, the added benefit becomes less apparent for the FD002 trained TL models. As the TL models mostly stay above the baseline and only show an advantage in the very extreme scarcity cases (where the engine count is less than or equal to 2). However, the FD003 trained TL models show clear benefits under data limitations, where they consistently perform better than the baseline in scenarios with fewer than 20 engine trajectories and remain stable throughout all training data availability scenarios. Figure 6.9 additionally presents the AUC-RMSE values for specific training data availability scenarios, where plot A uses 80 engines, plot B uses 40 engines, plot C uses 10 engines and plot D uses 5 engines.

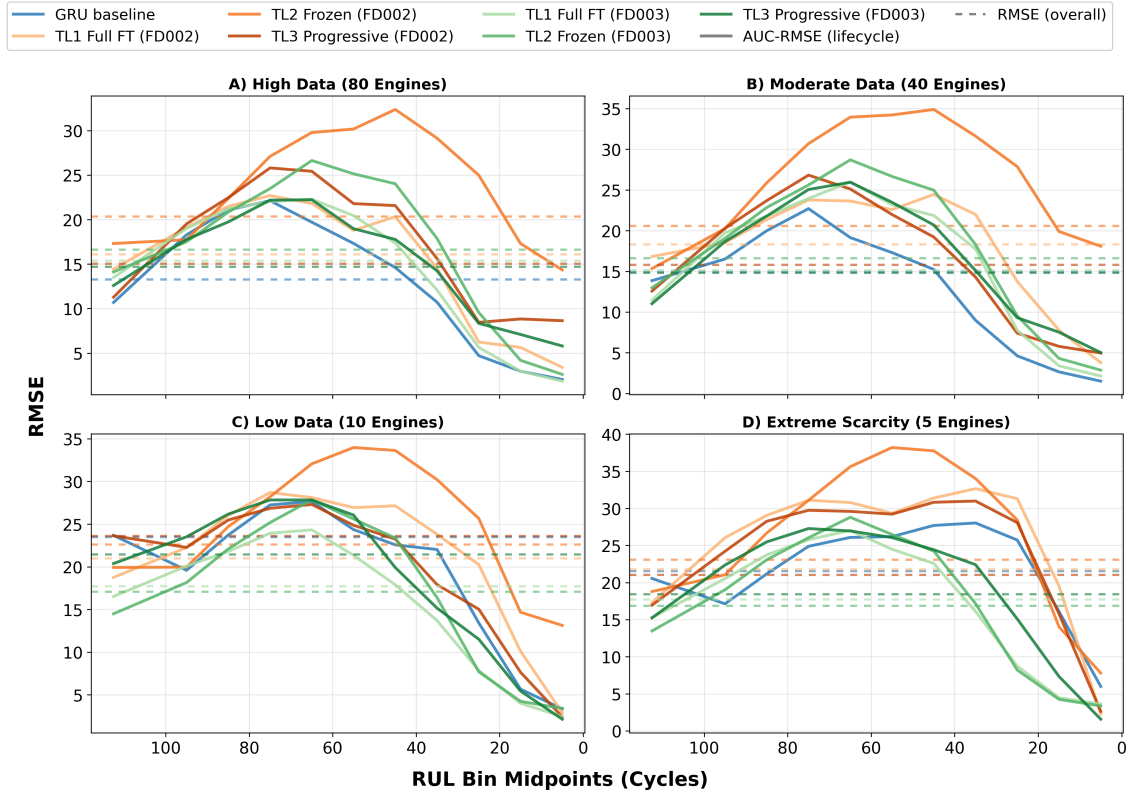


Figure 6.9: AUC-RMSE lifecycle curves (solid) and standard RMSE (stippled) for TL1, TL2 and TL3, as well as the GRU and Lasso models, across four data-availability scenarios using correlation-selected features as input: (A) 80 engines, (B) 40 engines, (C) 10 engines and (D) 5 engines.

From these plots, it can also be observed that the GRU-baseline performs better in the moderate to high data ranges across the entire lifecycle, with a particularly strong advantage near EOL. In contrast, in the lower ranges, especially close to EOL, FD003-trained TL models outperform the baseline. Furthermore, it can be observed that the TL2 strategy exhibits the strongest right-skewed distribution and, especially in the higher data scenarios, has difficulty making predictions near EOL compared with the other models.

6.5 Unsupervised Adaptation Results

This section addresses [R6](#), which investigates whether unsupervised domain adaptation can improve models trained only in the labelled source domain and how closely such adapted models can compare with the performance of supervised training models. The models compared are LSTM-DANN, GRU-DANN, and a GRU trained only on the labelled source domain (referred to here as the source-only baseline). It should be noted that the source-only model is trained exclusively on the source data set, and therefore its presented results stay unchanged because it performs inference only on the fixed target data set

while the source data remain constant (one scenario). Moreover, the FD002 and FD003 are used as separate source domains for all unsupervised models presented. Moreover, as discussed in [section 5.5](#), DANN variants use labelled source data together with unlabelled target data from FD001 through adversarial domain training. All models use the same 14 correlation-based selected features. [Table 6.7](#) presents the average test-set performance across data scarcity scenarios for these approaches. It also includes the supervised GRU results to assess how closely the adapted models perform compared with a supervised model trained on the labelled target domain.

Table 6.7: Test performance of the source-only and DANN domain-adaptation models using FD002 and FD003 as source domains, compared with the supervised GRU baseline trained directly on FD001, across five training-data scarcity ranges. Lower RMSE, AUC-RMSE, MAE, and C-MAPSS scores indicate better performance, while higher R^2 indicates better model fit. R-Rank and A-Rank denote the rankings according to RMSE and AUC-RMSE, respectively, and Time(s) denotes model training time in seconds.

Model	RMSE	AUC-RMSE	MAE	C-MAPSS	R^2	Time(s)	R-Rank	A-Rank
High Data: Average over 80–40 Training Engines								
GRU	13.81	12.64	10.69	3.31e+04	0.78	51.8	1	1
Source-only (FD003)	18.41	24.51	12.59	2.15e+05	0.62	79.6	2	2
GRU-DANN (FD003)	20.34	27.66	14.13	2.72e+05	0.53	96.6	3	3
LSTM-DANN (FD003)	21.36	31.87	14.36	4.92e+05	0.48	186.7	4	5
LSTM-DANN (FD002)	24.39	31.93	20.24	2.14e+05	0.32	103.3	5	6
GRU-DANN (FD002)	27.27	31.09	24.54	1.65e+05	0.15	125.2	6	4
Source-only (FD002)	36.71	43.39	25.04	1.91e+06	-0.53	205.6	7	7
Moderate Data: Average over 40–20 Training Engines								
GRU	15.54	14.42	12.05	5.01e+04	0.73	43.4	1	1
Source-only (FD003)	18.41	24.51	12.59	2.15e+05	0.62	79.6	2	2
GRU-DANN (FD003)	20.36	26.98	14.06	2.64e+05	0.53	85.0	3	3
LSTM-DANN (FD003)	21.00	29.60	14.28	3.88e+05	0.50	159.5	4	4
LSTM-DANN (FD002)	23.38	30.55	18.28	2.12e+05	0.38	130.6	5	6
GRU-DANN (FD002)	25.13	29.84	22.02	1.39e+05	0.28	124.2	6	5
Source-only (FD002)	36.71	43.39	25.04	1.91e+06	-0.53	205.6	7	7
Low Data: Average over 20–10 Training Engines								
Source-only (FD003)	18.41	24.51	12.59	2.15e+05	0.62	79.6	1	2
GRU	19.94	18.14	16.19	1.13e+05	0.5361	37.9	2	1
GRU-DANN (FD003)	20.02	26.11	14.35	2.19e+05	0.55	64.3	3	3
LSTM-DANN (FD003)	20.90	30.61	14.01	4.49e+05	0.51	176.5	4	5
LSTM-DANN (FD002)	25.24	28.04	20.08	2.10e+05	0.27	107.4	5	4
GRU-DANN (FD002)	28.40	33.83	25.74	2.00e+05	0.09	137.2	6	6
Source-only (FD002)	36.71	43.39	25.04	1.91e+06	-0.53	205.6	7	7
Very Low Data: Average over 10–5 Training Engines								
Source-only (FD003)	18.41	24.51	12.59	2.15e+05	0.6166	79.6	1	2
GRU-DANN (FD003)	20.20	26.25	14.64	2.17e+05	0.5380	66.0	2	3
LSTM-DANN (FD003)	20.94	30.18	14.32	4.15e+05	0.5037	149.6	3	5
GRU	22.21	20.51	17.34	3.14e+05	0.4365	47.1	4	1
LSTM-DANN (FD002)	26.65	29.44	23.47	1.48e+05	0.1866	94.4	5	4

GRU-DANN (FD002)	29.03	32.18	26.46	1.96e+05	0.0458	117.7	6	6
Source-only (FD002)	36.71	43.39	25.04	1.91e+06	-0.5253	205.6	7	7
Extreme Scarcity: Average over ≤ 5 Training Engines								
Source-only (FD003)	18.41	24.51	12.59	2.15e+05	0.62	79.6	1	1
GRU-DANN (FD003)	19.94	25.94	14.41	2.13e+05	0.55	62.5	2	2
LSTM-DANN (FD003)	21.11	30.71	14.46	4.12e+05	0.50	131.3	3	6
LSTM-DANN (FD002)	22.64	30.25	16.95	3.20e+05	0.42	142.1	4	5
GRU-DANN (FD002)	25.60	26.79	21.87	1.37e+05	0.25	104.2	5	3
Source-only (FD002)	36.71	43.39	25.04	1.91e+06	-0.51	205.6	6	7
GRU	45.34	27.92	40.39	8.09e+06	-1.79	40.6	7	4

In the high-data range, the supervised GRU model performs best overall. It achieves the lowest RMSE, AUC-RMSE, C-MAPSS score and training time and the highest R^2 . The source-only baseline trained on FD003 performs best out of all unsupervised models. The model scores an RMSE of 18.41, AUC-RMSE of 24.51, MAE of 12.59, C-MAPSS score of 2.15×10^5 , and R^2 of 0.6166. However, the source-only model trained on FD002 performs substantially worse, with an RMSE of 36.71, AUC-RMSE of 43.39, MAE of 25.04, C-MAPSS score of 1.91×10^6 , and a negative R^2 of -0.5253. Therefore, within the evaluated FD002 to FD001 configurations, both FD002-based DANN models achieved better performance than the source-only FD002 baseline, indicating a benefit from adversarial training under this larger domain-shift setting. Moreover, in the high data setting the GRU-DANN trained on FD003 is the strongest DANN variant overall, with an RMSE of 20.34, AUC-RMSE of 27.66, MAE of 14.13, and R^2 of 0.5317. It also has a relatively low training time of 96.6 seconds compared with the LSTM-DANN trained on FD003, which requires 186.7 seconds. However, the benefit of using a GRU-based DANN over an LSTM-based DANN is not consistent across source domains. The LSTM-DANN performs better than the GRU-DANN in RMSE, MAE, C-MAPSS score, and R^2 , (scoring an RMSE of 24.39 and R^2 of 0.3234) compared with the GRU-DANN FD002 model (scoring RMSE of 27.27 and R^2 of 0.1459).

In the moderate-data range, the supervised GRU remains the best model overall. However, it is observed that the performance gap between supervised target-domain training and the FD003 transfer models becomes smaller. The supervised GRU RMSE increases from 13.81 in the high-data range to 15.54 in the moderate-data range. The source-only models remain almost unchanged in performance across the data ranges. The FD003 source-only model becomes relatively more competitive as the amount of labelled FD001 data decreases since it is trained on a fixed amount of source data. Moreover, the DANN models become slightly more competitive in the moderate-data range. The GRU-DANN FD003 model achieves an RMSE of 20.36, AUC-RMSE of 26.98, MAE of 14.06, and R^2 of 0.5310. Compared with the high-data range, its AUC-RMSE improves from 27.66 to 26.98, while its other metrics remain relatively stable. Similarly, the LSTM-DANN FD003 model improves from an RMSE of 21.36 to 21.00 and from an R^2 of 0.48 to 0.50. The FD002-

based DANN models also improve slightly, particularly the GRU-DANN FD002 model, whose R^2 increases from 0.15 to 0.28. These results suggest that the DANN models are less directly affected by the reduction in labelled target data than the supervised GRU model.

Notably, in the low-data range, the source-only FD003 model achieves the best performance in terms of RMSE, MAE, and R^2 , with an RMSE of 18.41, an MAE of 12.59, and an R^2 of 0.6166. In comparison, the supervised GRU model drops in to an RMSE of 19.94, an MAE of 16.19, and an R^2 of 0.5361. However, the supervised GRU still performs best in terms of AUC-RMSE in the low-data range, with a value of 18.14 compared with 24.51 for the source-only FD003 model. This suggests that although the source-only FD003 model achieves better average pointwise accuracy, the supervised GRU maintains a stronger prediction trajectory across the life cycle. Furthermore, the FD003-based DANN models are also competitive in the low-data range compared to the supervised GRU. The GRU-DANN FD003 model achieves an RMSE of 20.02 and an R^2 of 0.5464, while the LSTM-DANN FD003 model achieves an RMSE of 20.90 and an R^2 of 0.5059. These values are comparable to the supervised GRU model's RMSE of 19.94 and R^2 of 0.5361.

In the very-low-data range the advantage of the FD003-based transfer models becomes more apparent. The source-only FD003 model remains the best-performing model for RMSE, MAE, and R^2 , with an RMSE of 18.41, MAE of 12.59, and R^2 of 0.6166. The GRU-DANN FD003 model also remains stable, with an RMSE of 20.20, MAE of 14.64, and R^2 of 0.5380. The LSTM-DANN FD003 model records an RMSE of 20.94 and R^2 of 0.5037, all values stronger than the supervised GRU model in terms of RMSE, MAE and R^2 , where the GRU achieves an RMSE of 22.21, MAE of 17.34, and R^2 of 0.4365. This result shows that the supervised GRU model becomes increasingly limited when the labelled target data is severely reduced. Although it still achieves the best AUC-RMSE score of 20.51, its general error metrics deteriorate substantially. Notably, the FD003-based transfer models remain comparatively stable across the low- and very-low-data ranges. For example, the GRU-DANN FD003 model changes only slightly from an RMSE of 20.02 in the low-data range to 20.20 in the very-low-data range, while its R^2 decreases only slightly from 0.5464 to 0.5380. Moreover, the models trained on the FD002 source are still failing when there are low amounts of data available. The highest performing DANN model using FD002 as base, by R^2 performance, is the LSTM-DANN (FD002) with a R^2 value of 0.19. In spite of being the most successful at achieving the lowest C-MAPSS scores in very-low-data sets with a C-MAPSS score of 1.48×10^5 , the RMSE is 26.65 and MAE is 23.47. Furthermore, GRU-DANN (FD002) performed even worse than LSTM-DANN (FD002) with RMSE of 29.03 and R^2 of 0.0458. However, in particular, both FD002-based DANN models outperformed the source-only FD002 baseline, with the FD002 baseline attaining an RMSE of 36.71 and a negative R^2 of -0.5253. This suggests that for datasets such as FD002, where the source-target domain shift is larger, adversarial learning is more effective than the source-only method.

In the extreme-scarcity scenario, the source-only FD003 model again achieves the best RMSE, AUC-RMSE, MAE, and R^2 , with an RMSE of 18.41, AUC-RMSE of 24.51, MAE of 12.59, and R^2 of 0.6166. The GRU-DANN FD003 model is the second-best model, with an RMSE of 19.94, AUC-RMSE of 25.94, MAE of 14.41, and R^2 of 0.5499. The FD002-based DANN models improve marginally compared with the very-low-data setting. For example, the GRU-DANN FD002 model improves from an RMSE of 29.03 to 25.60 and from an R^2 of 0.0458 to 0.2510. The LSTM-DANN FD002 model also achieves an RMSE of 22.64 and R^2 of 0.4196, which is significantly better than its very-low-data scenario score of R^2 of 0.1866. Moreover, the source-only (FD002) and supervised GRU models perform significantly worse than all DANN variants, scoring negative R^2 .

To further answer [item R7](#), which investigates the effect of domain shift on performance, [Figure 6.10](#) plots the relative improvement of the DANN models compared to the GRU trained only on the source domain. Specifically, it plots the relative improvement for the FD002 to FD001 and FD003 to FD001 transfer settings in terms of RMSE and AUC-RMSE.

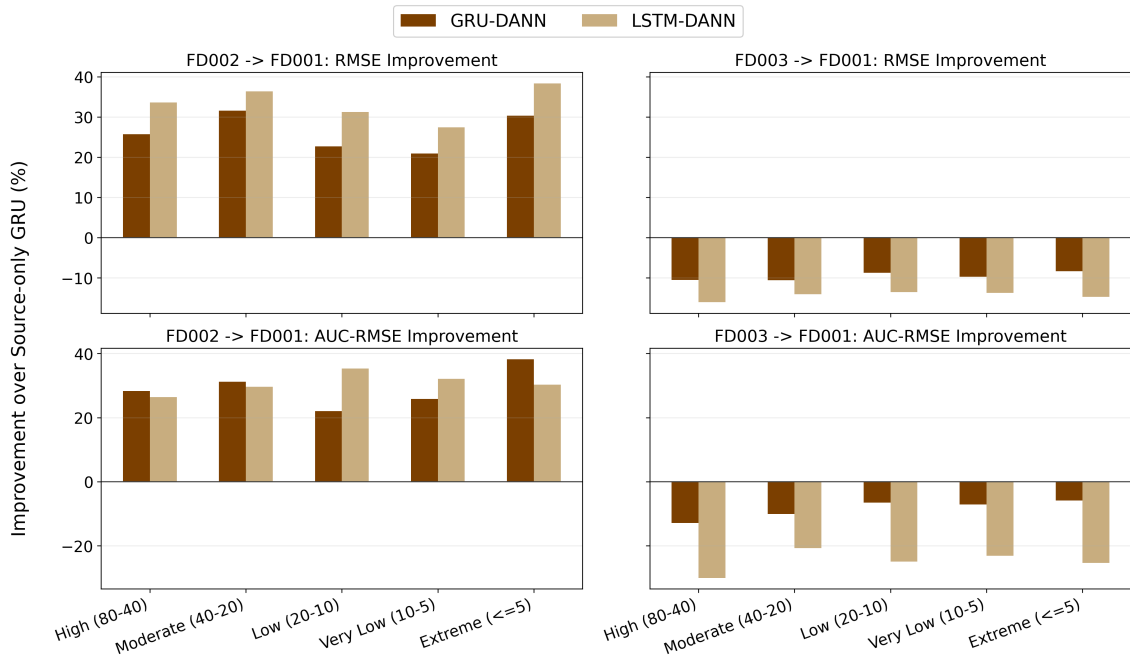


Figure 6.10: Improvement of unsupervised TL, including GRU-DANN and LSTM-DANN models, over the source-only GRU

From [Figure 6.10](#) shows that, within the evaluated FD002 to FD001 configurations, the DANN models provided an improvement over the source-only baseline under the larger domain-shift setting. In particular, the added benefit is slightly more prevalent for the LSTM-DANN architecture in terms of overall RMSE improvement above 27% for all data scenarios (with the highest improvement observed in the extreme scarcity scenarios of 38%). However, in terms of AUC-RMSE, there is no clear superior DANN variant, where

GRU-DANN performs slightly better in the high, moderate, and extreme data scenarios, and the LSTM-DANN performs better in the low to very low ranges. However, when the domain shift is smaller (as in the FD003 to FD001 setting), the added benefit of DANN models is not observed. In this evaluated FD003 to FD001 setting, the DANN models perform worse than the source-only baseline. Notably, the LSTM-DANN is observed to yield the highest negative transfer for all data ranges. To further visualise the performance of the DANN models, Figure 6.11, plots the test RMSE results for all independent engine count scenarios for GRU-DANN, LSTM-DANN, and source-only GRU. In addition, for comparison, the supervised GRU is also plotted.

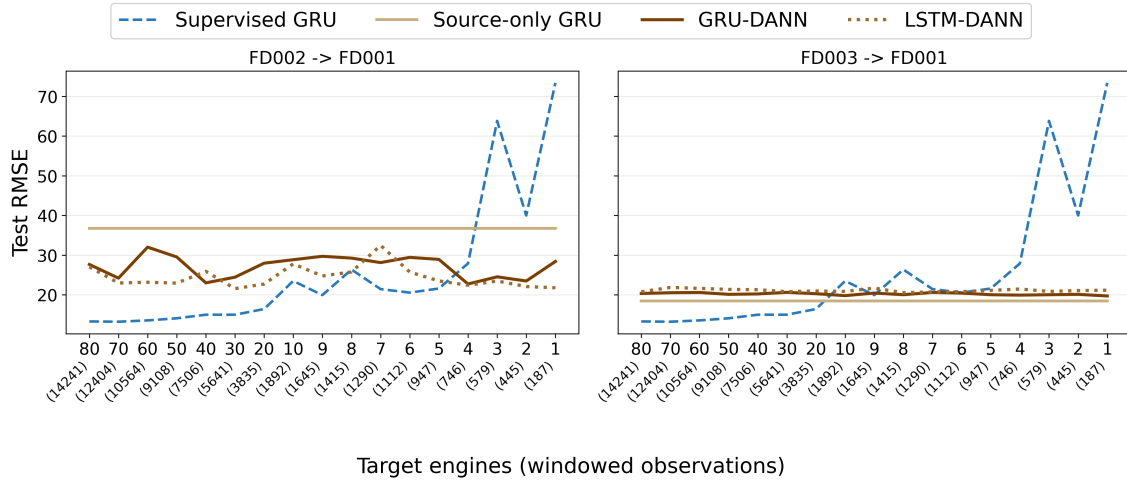


Figure 6.11: Test RMSE results for all independent engine-count scenarios for GRU-DANN, LSTM-DANN and the source-only GRU, as well as the supervised GRU baseline.

Through Figure 6.11, it is observed that in the case of FD002 to FD001 TL, the performance of the DANN models, in different data scarcity scenarios, is stable compared to the supervised GRU. Moreover, within the evaluated FD002 to FD001 setting, the DANN models achieved lower RMSE than the source-only GRU across all engine-count scenarios. In the case of FD003 to FD001 transfer learning, the models show very consistent horizontal performance. In particular, the single-source GRU slightly performs lower RMSE values than the DANN variants. Moreover, the graphs show that it is not so clear which DANN variant performs best since in both FD002 and FD003 source data settings, they alternate in best performance at varying engine availability scenarios.

6.5.1 AUC-RMSE Analysis of Domain Adaptation TL Strategies

Figure 6.12 plots the test AUC-RMSE results for all independent engine count scenarios for GRU-DANN, LSTM-DANN and source-only GRU as well as the supervised GRU baseline.

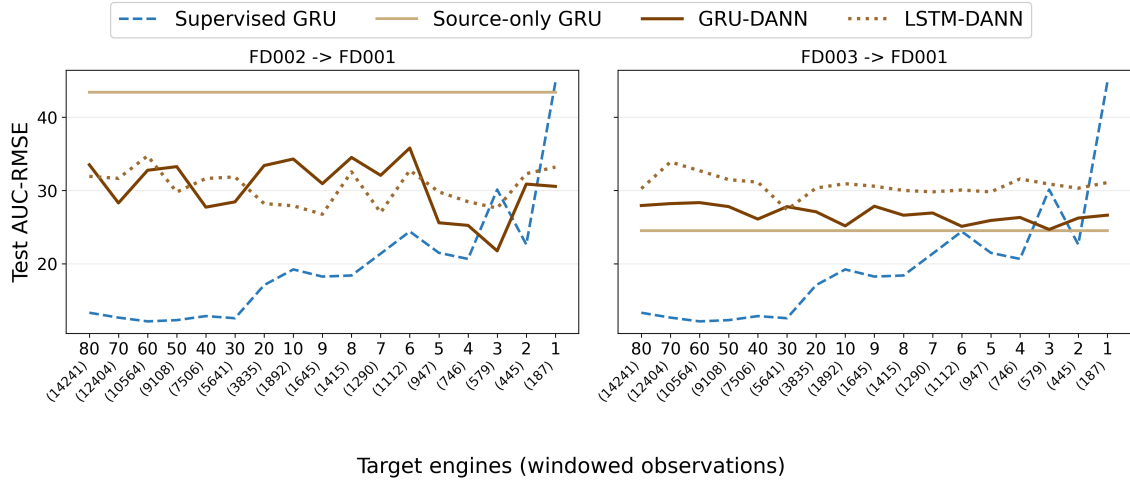


Figure 6.12: The test AUC-RMSE results for all independent engine count scenarios for GRU-DANN, LSTM-DANN and source-only GRU as well as the supervised GRU baseline.

Compared with [Figure 6.11](#), the AUC-RMSE trends differ for both the FD002 to FD001 and FD003 to FD001 transfer scenarios. For both cases, the AUC-RMSE scores are notably more unstable across the different engine-availability scenarios. For the evaluated FD002 to FD001 transfer setting, both DANN models achieved lower AUC-RMSE than the source-only model across all engine-count scenarios, while neither DANN variant was consistently best. In the FD003 to FD001 transfer setting, the GRU-DANN is observed to perform consistently better than the LSTM-DANN for almost all engine-availability scenarios. However, as observed earlier, the source-only baseline performs consistently better than the DANN models. Furthermore, the advantage of having labelled data for supervised learning becomes more apparent in the high-data ranges and remains superior when training-data scarcity is not low.

To provide a more detailed assessment of AUC-RMSE performance, [Figure 6.13](#) displays the test AUC-RMSE lifecycle trajectories (solid lines) together with the corresponding standard RMSE scores (dotted lines) for GRU-DANN, LSTM-DANN and the source-only GRU trained on FD002. Moreover, these are compared against the supervised GRU baseline. The results are reported for four target-data availability settings using the correlation-based feature subset.

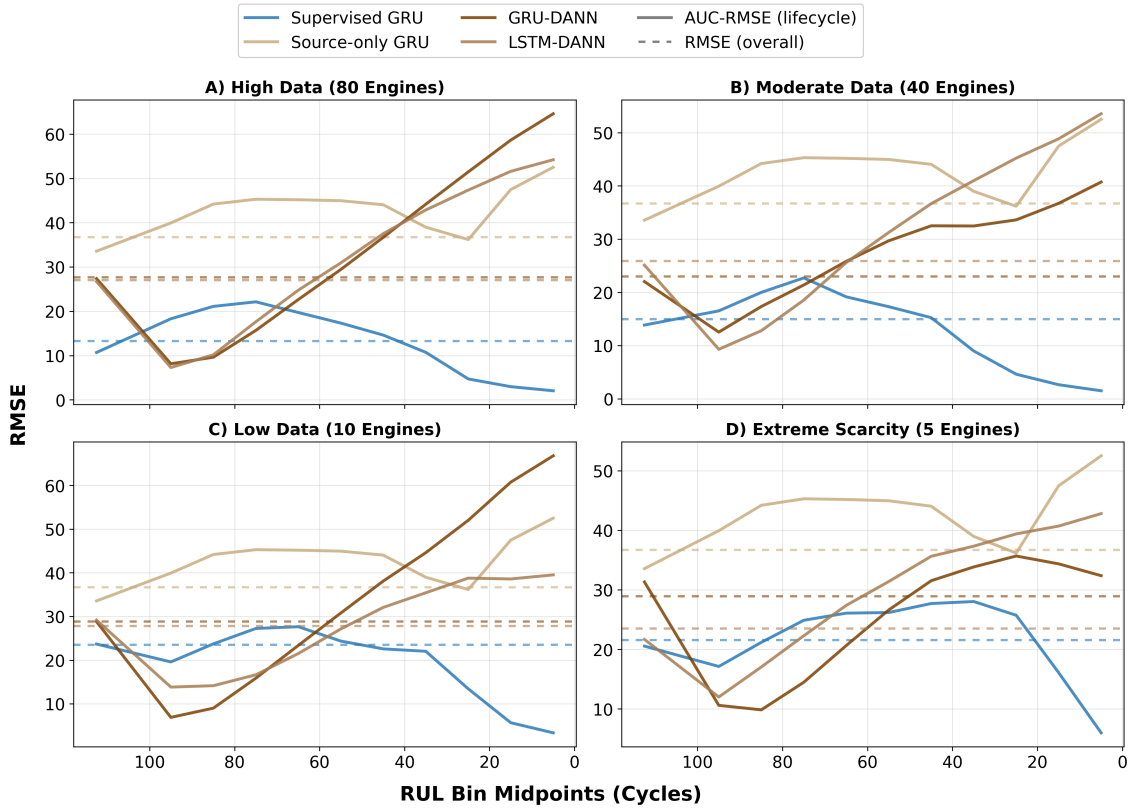


Figure 6.13: AUC-RMSE lifecycle curves (solid lines) and standard RMSE values (stippled lines) for GRU-DANN, LSTM-DANN and source-only GRU trained on **FD002**, compared with the supervised GRU baseline trained on labelled FD001 target-domain-data. Results are shown for four target-data availability scenarios using the correlation-based selected feature subset: (A) 80 engines, (B) 40 engines, (C) 10 engines and (D) 5 engines.

Interestingly, from [Figure 6.13](#), it is observed that DANN models perform better compared to the supervised GRU model in the early stages (90 to midlife 60 RUL cycles) for all data availability scenarios. However, it performs notably worse at the RUL cap (125) and in the near EOL RUL. This upward trend in RMSE suggests that the FD002-based DANN models do not capture the near-EOL degradation behaviour as reliably as the supervised GRU. This can be further visualised through [Figure 6.14](#), which plots the predicted versus actual RUL on the FD001 test set for the GRU-DANN (FD002 as source) and similarly [Figure 6.15](#) for the LSTM-DANN (FD002 as source). Each point index is one sliding-window test observation, and the blue and red show true and predicted RUL values, respectively. In addition, a similar plot for the source-only baseline is presented in [Figure 6.16](#).

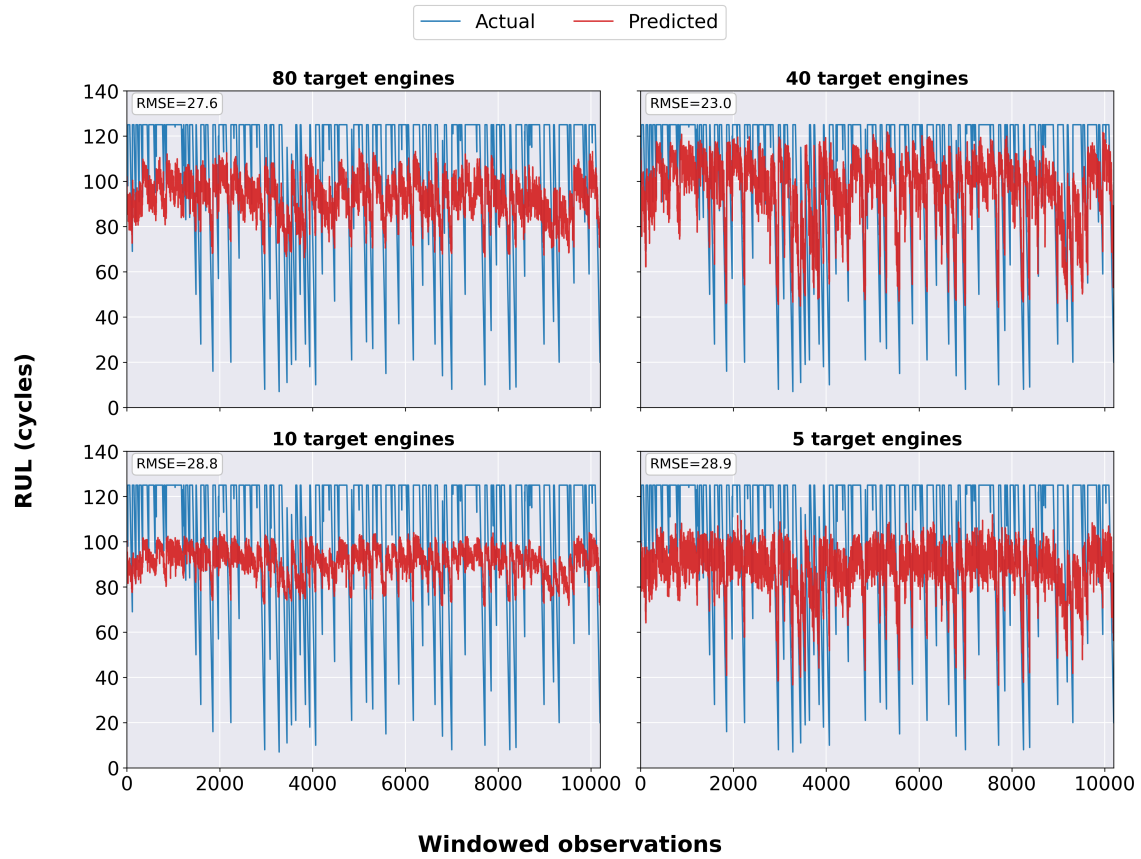


Figure 6.14: Predicted versus actual RUL on the FD001 test set for the **GRU-DANN** model using **FD002** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.

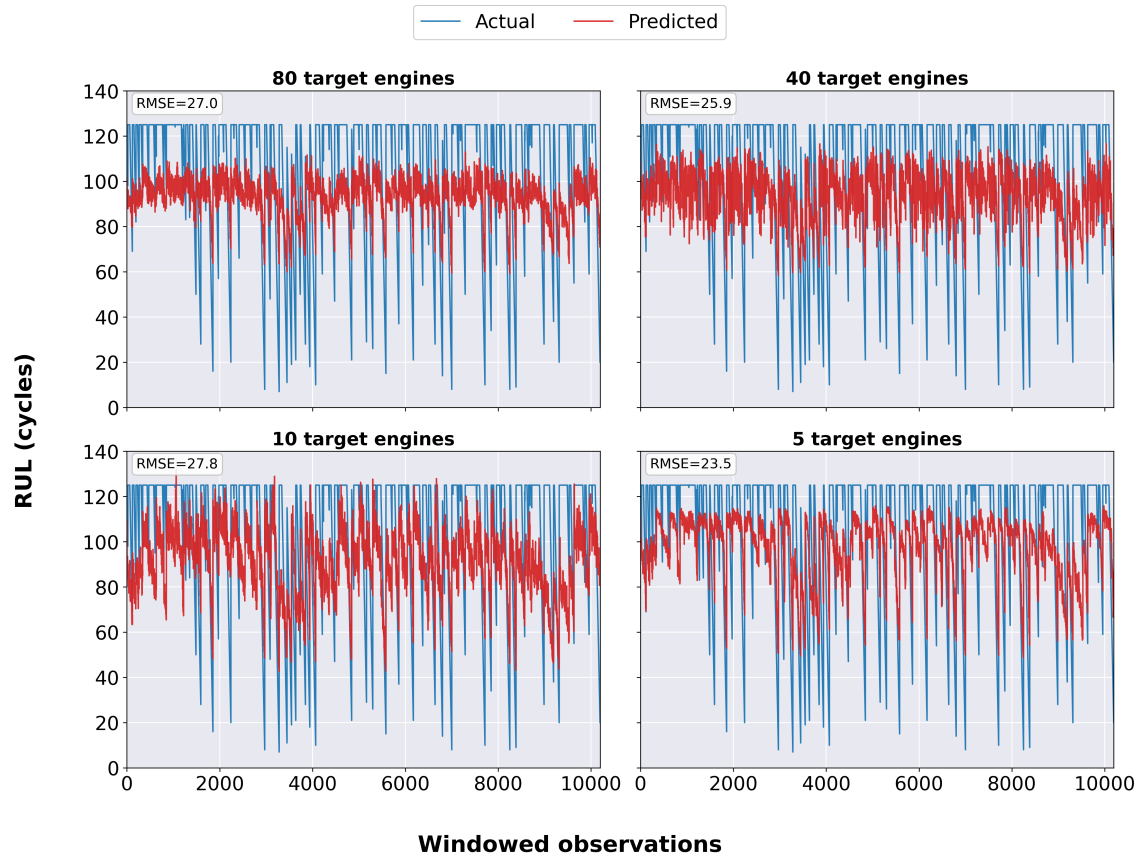


Figure 6.15: Predicted versus actual RUL on the FD001 test set for the **LSTM-DANN** model using **FD002** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.

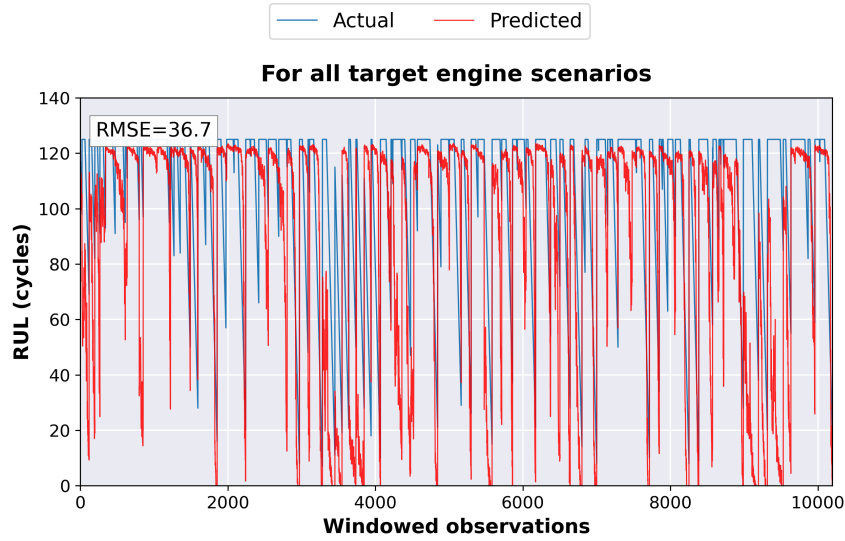


Figure 6.16: Predicted versus actual RUL on the FD001 test set for the **source-only GRU** model using **FD002** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.

As evident by [Figure 6.14](#) and [Figure 6.15](#). It is further visualised that the DANN models struggle to make predictions at the very early healthy life cycles and the near EOL RUL cycles and make most of their predictions between 100 and 60 RUL cycles (early- to mid-term life cycles). In contrast [Figure 6.16](#) the source-only baseline shows significant noise and collapses toward very low or near-zero RUL while the true RUL is relatively healthy.

As highlighted in [Table 6.7](#) the AUC-RMSE scores show more promising results when using FD003 (closer domain similarity). [Figure 6.17](#) plots the AUC-RMSE lifecycle curves for the FD003 source data case.

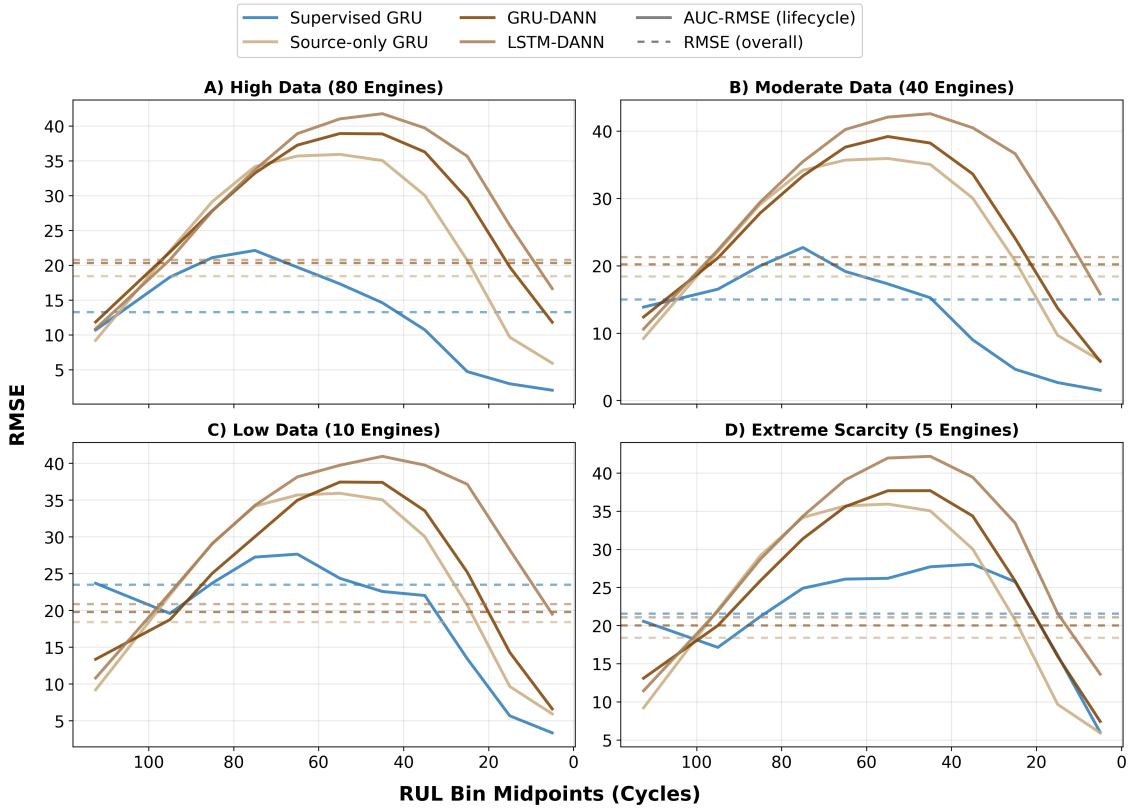


Figure 6.17: AUC-RMSE lifecycle curves (solid lines) and standard RMSE values (stippled lines) for GRU-DANN, LSTM-DANN and source-only GRU trained on **FD003**, compared with the supervised GRU baseline trained on labelled FD001 target-domain-data. Results are shown for four target-data availability scenarios using the correlation-based selected feature subset: (A) 80 engines, (B) 40 engines, (C) 10 engines and (D) 5 engines.

As observed in [Figure 6.17](#), the DANN models show the expected bell-shaped lifecycle error pattern, indicating stronger predictive reliability near the EOL RUL cycles and higher errors around the mid-life region. The FD003-based models perform comparably during the early healthy stages of the engine lifecycle, but their behaviour diverges more notably as the lifecycle approaches EOL. In low-data and extreme-scarcity scenarios, the supervised GRU shows higher RMSE during the early lifecycle regions, which increases its overall RMSE. However, across much of the remaining lifecycle, the supervised GRU remains the model with the strongest performance. Notably, in the extreme-scarcity scenario, the source-only baseline outperforms the supervised GRU near EOL.

To further demonstrate the DANN and source-only models' ability to predict the RUL near EOL the predicted versus true RUL plots can be referred to in [Figure 6.18](#) (GRU-DANN), [Figure 6.19](#) (LSTM-DANN), [Figure 6.20](#) (source-only GRU). In addition, the supervised GRU-predicted versus true RUL plots are also presented ([Figure 6.21](#)).

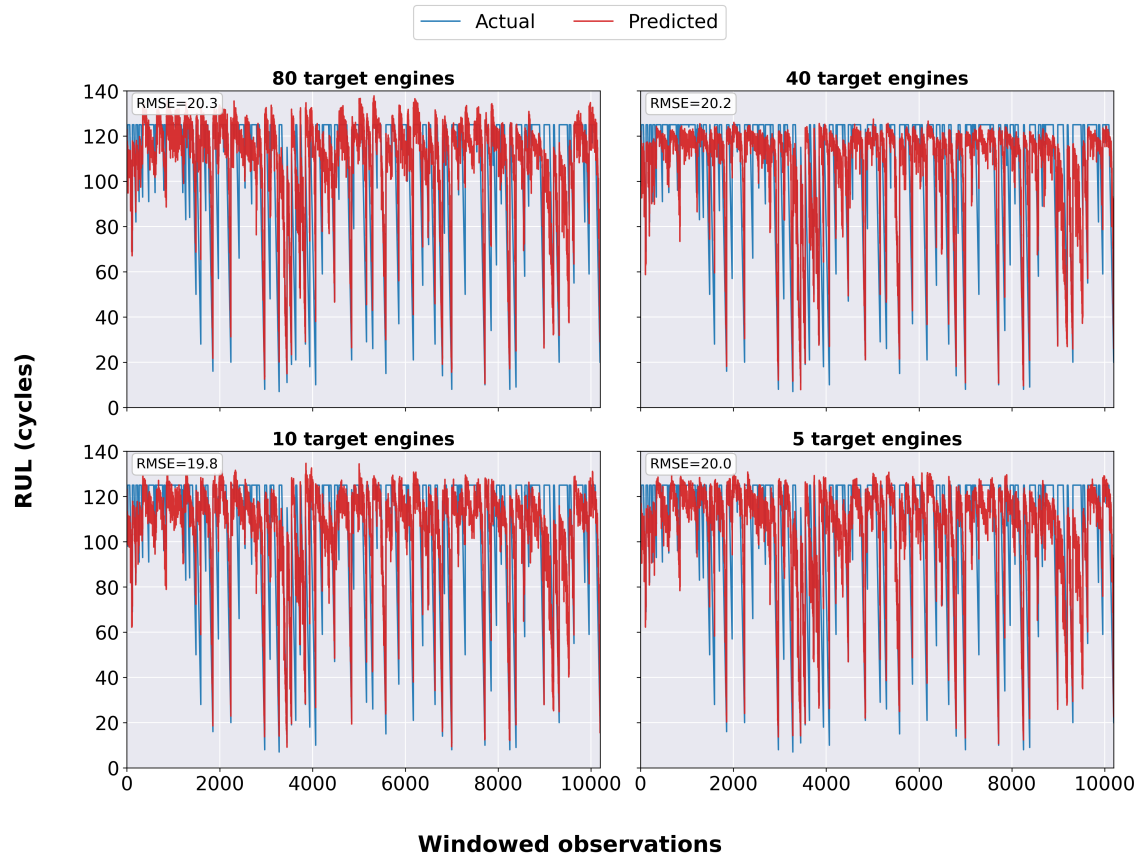


Figure 6.18: Predicted versus actual RUL on the FD001 test set for the **GRU-DANN** model using **FD003** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.

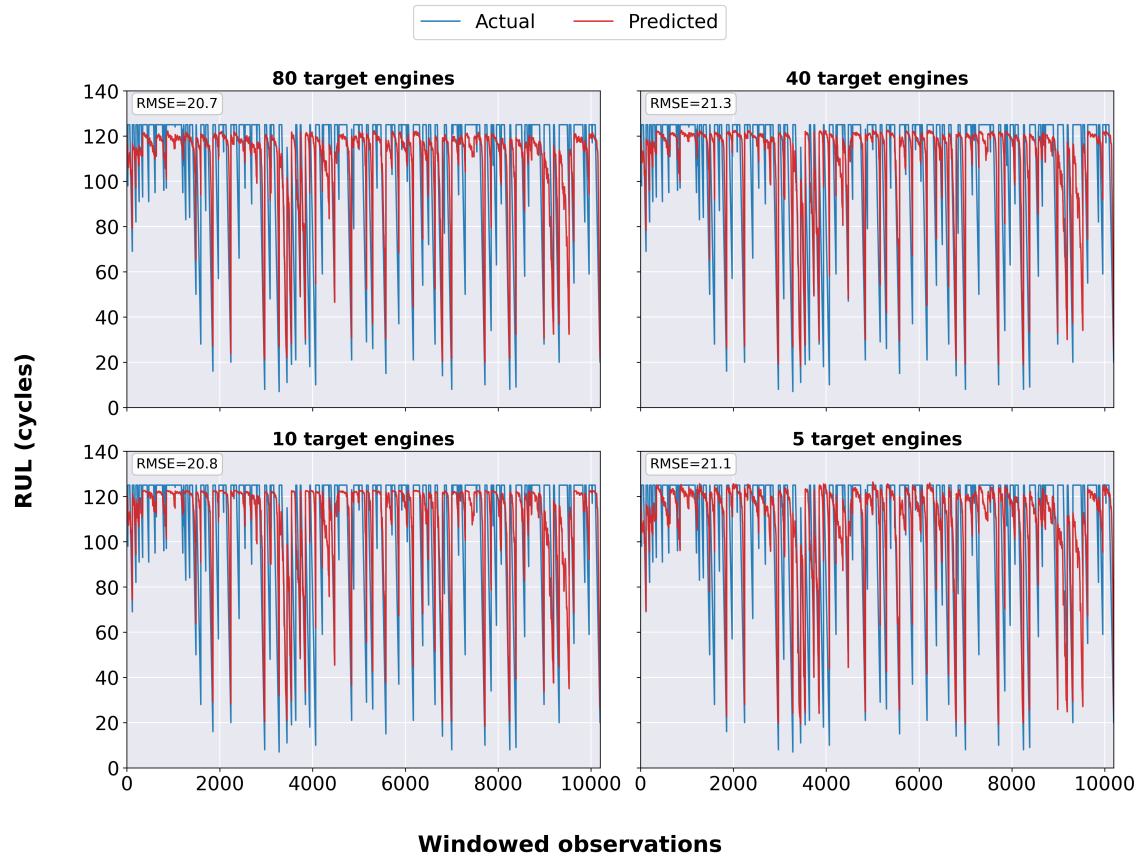


Figure 6.19: Predicted versus actual RUL on the FD001 test set for the **LSTM-DANN** model using **FD003** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.

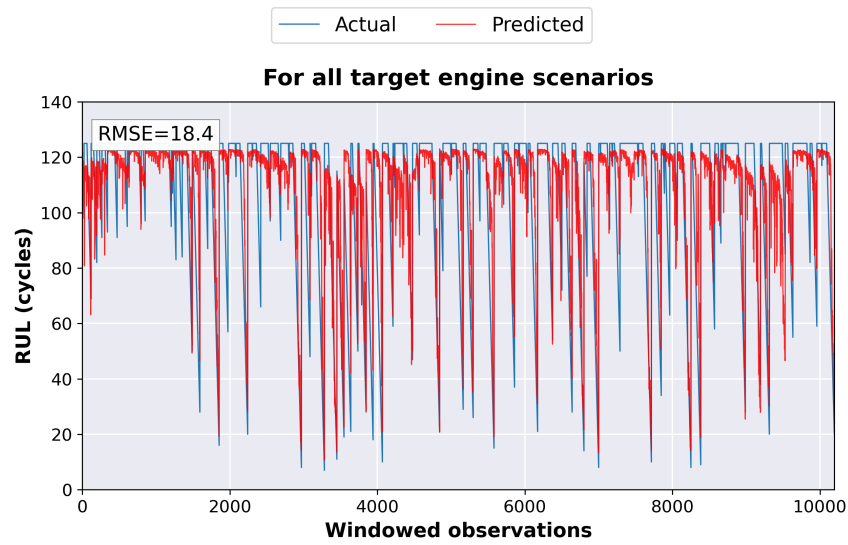


Figure 6.20: Predicted versus actual RUL on the FD001 test set for the **source-only GRU** model using **FD003** as the source domain. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.

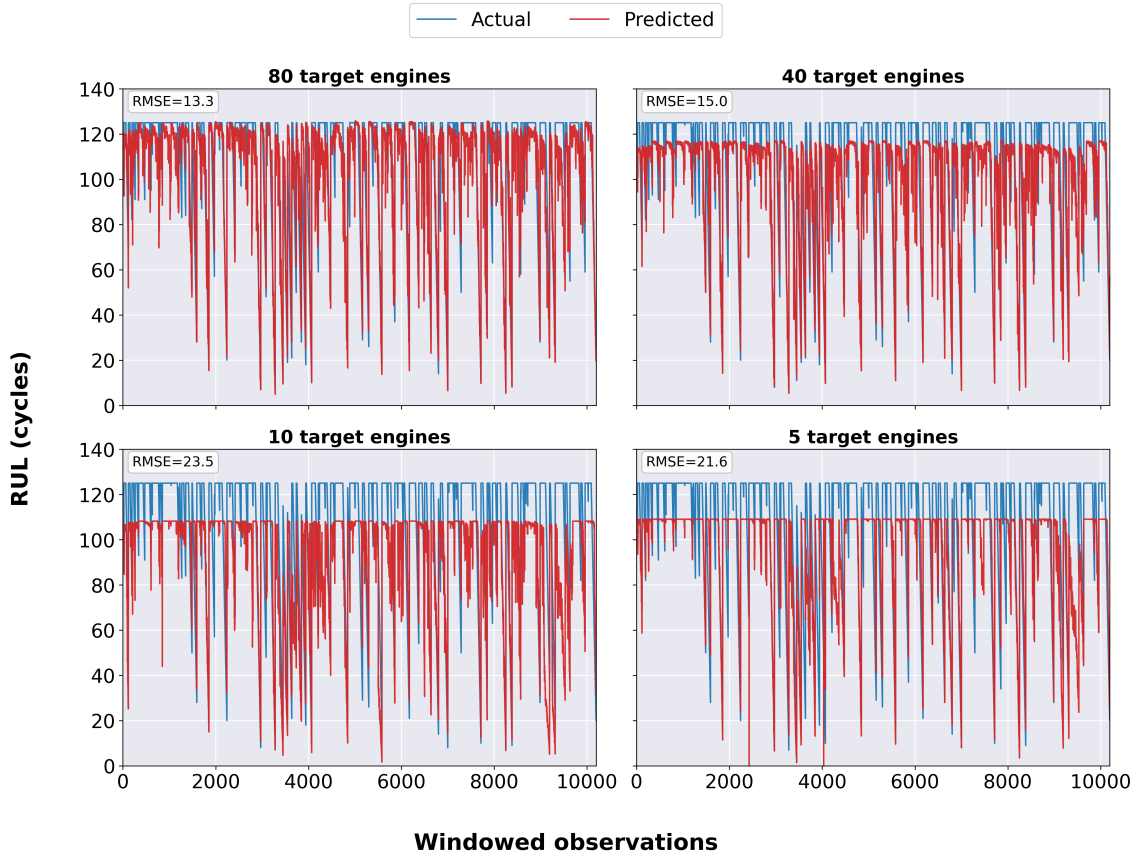


Figure 6.21: Predicted versus actual RUL on the FD001 test set for the **Supervised-GRU** model using **labelled FD001** training data. Each point index represents one sliding-window test observation, with the blue series showing the true RUL and the red series showing the predicted RUL.

In [Figure 6.18](#) and [Figure 6.19](#), it can be observed that the DANN variants trained with the FD003 data set show a much better ability to predict the RUL across the full lifecycle range compared to the DANN models trained with FD002. Notably, the GRU-DANN model makes a substantial amount of over-predictions in the early-life region, where the predicted RUL values hover slightly above the capped RUL. However, these predictions are still within close proximity of the RUL cap, whereas compared to the LSTM-DANN, the GRU-DANN predictions are also more noisy across the windowed observations.

The LSTM-DANN model, shows a different pattern of predictions. It still follows the general shape of the actual RUL values, but it consistently undershoots the RUL cap by a marginal amount in the early-life region. In addition, it is apparent that the model consistently overshoots when the actual RUL nears the EOL region. However, both DANN variants show a consistent performance across the different target-engine scenarios. The source-only GRU shown in [Figure 6.20](#) also shows an ability to make predictions throughout the RUL lifecycle. In most cases, it is able to predict the RUL reasonably well as the actual RUL nears EOL. However, for the source-only GRU and the two DANN variants,

it is observed that in some cases the predictions significantly overshoot the actual RUL.

As observed in [Figure 6.21](#), the supervised GRU model demonstrates strong overall predictive performance in the 80 target-engine setting, achieving an RMSE of 13.3. The RUL values generally closely follow the actual RUL values, particularly within the capped high-RUL region. In particular, compared to the DANN variants and the source-only GRU, the supervised GRU has fewer instances where significant overshooting occurs near the EOL region. Furthermore, it is observed that as the number of target training engines decreases, the supervised GRU starts to struggle more with predicting the early-life capped RUL region. This is especially notable in the 10 and 5 engine scenarios, where the predictions show lower deviations more frequent from the actual capped RUL. However, across all target-data scenarios, it is still observed that the predictions follow the actual RUL relatively closely in the mid to near-EOL RUL cycles.

6.6 Discussion

This section aims to consolidate the findings by addressing the seven research questions defined in [subsection 1.1.5](#). Subsequently, the models that perform best ([R1](#)) and the effects of feature engineering ([R2](#)) under low-data scenarios are identified. Furthermore, a comparison between AUC-RMSE and traditional metrics is discussed [R3](#). This is followed by a discussion of whether applying supervised TL techniques improves model performance [R4](#). In response to [R5](#) it is assessed whether source-domain models provide competitive performance on target domains when target data is not available. Then [R6](#) is addressed by discussing the relative performance of unsupervised domain adaptation when labelled target data are not available. [R7](#) then discusses whether the severity of the domain shift between the source and target domains affects the relative effectiveness of the TL strategies.

6.6.1 Model Robustness Under Data Scarcity

The results show that the best-performing architecture under full-data conditions (80 training engines) is not necessarily the most robust under data scarcity. Of all baseline models, the GRU performed the best overall in the high and moderate data ranges. This indicates that its gating structure (see [section 2.1.3](#)) is well suited to these scenarios. However, the Transformer shows the most robustness out of all neural-network-based models as data become scarce. It ranked second in RMSE for the high- and moderate-data scenarios and consistently outperformed the other neural-network-based models in the lower-data settings. As discussed in [section 2.1.3](#), Transformers can model long-range dependencies and interactions, and their attention mechanism reduces the path length between distant positions to a constant. This may allow the Transformer to extract

informative relationships from each input window more effectively when fewer training engines are available.

Furthermore, it was observed that the neural-network-based models showed superior performance in higher-data settings compared to tree-based and regression models. However, as data becomes more scarce, the simpler linear models start to become more competitive, while neural-network-based models, particularly RNN, degrade significantly. This is likely because neural-networks have substantially more parameters than linear models and are therefore more prone to overfitting when less training data is available. However, the Transformer model was observed to perform the best in the lower-data settings, and only in the extreme-data settings does it perform marginally worse than the linear and tree-based models.

6.6.2 Effect of Feature Engineering Under Low-Data Conditions

With regard to [R2](#), FS did not provide a consistent predictive improvement or degradation, but showed an evident computational advantage. The main effect of FS was a substantial reduction in training time, especially for the RNNs. In terms of the prediction metrics, FS often worsened the GRU and LSTM RMSE, particularly under lower-data settings. This implies that some of the removed features still contained useful degradation information. However, the Transformer was less negatively affected and, in the lower ranges, an improvement was observed. Poly2 showed the greatest advantage from training on a dimensionally reduced dataset. This is consistent with the discussion in [section 2.1.1](#), where polynomial regression is described as highly sensitive to dimensionality because the number of terms grows combinatorially. Moreover, the application of FE showed benefit, particularly as the data was reduced. AE-GRU was strongest in the high-data and extreme-scarcity ranges, TCN-GRU was strongest in the moderate, low, and very-low-data ranges, and CNN-GRU showed a notable discrepancy between RMSE and AUC-RMSE, where it performed notably poorly in terms of RMSE but competitively in terms of AUC-RMSE.

6.6.3 Comparison Between AUC-RMSE and Traditional Metrics

In response to [R3](#), the AUC-RMSE metric is observed to expose different conclusions from the more traditionally used RMSE. This is expected since the RMSE metric can be biased if more test labels are concentrated within a specific stage of the asset life-cycle, as evident in [Figure 6.2](#). However, the AUC-RMSE metric weights predictions throughout the turbofan engine life cycle more evenly. Therefore, if a model performs anomalously poorly or anomalously well in a densely populated life cycle stage, this will not dominate the overall result to the same extent and the other stages of the life cycle will be weighted

more evenly. To observe these differences, the relative RMSE and AUC-RMSE ranks were compared to determine whether a model improved or decreased in rank depending on the metric used. The observed differences did not consistently show an increase or decrease in rank. For example, the most notable increase in rank when assessing based on AUC-RMSE was the BiLSTM in the 40 to 20 engine range, ranking 9th in RMSE but 1st in AUC-RMSE. In contrast, in the 10 to 5 engine setting, TCN-GRU scored first in RMSE but last of all hybrid approaches in terms of AUC-RMSE. Similarly, CNN-GRU showed significantly worse RMSE compared to all hybrid models, but ranked first in AUC-RMSE in the very low data range.

Moreover, AUC-RMSE shows a particular advantage when using the AUC-RMSE life-cycle plot, as it shows the predictive accuracy across each stage of the turbofan engine life cycle. This subsequently enables observations that expose whether models reliably predict RUL across all life-cycle ranges, or whether they perform better in the early-life region, where the engine is in a healthier state, or near EOL, where the engines start demonstrating more notable degradation patterns. For example, by assessing the shape of the plot, if it is right-skewed, as most notably shown by the Lasso and TL2, it can be observed that the models comparatively struggle with near-EOL predictions. Furthermore, it was observed that most models showed a bell-shaped curve, performing particularly well very near the EOL. This trend is anticipated, as the sensor data show more distinct and characteristic degradation patterns near EOL. However, as shown in [Figure 6.13](#), the LSTM-DANN (FD002) and GRU-DANN (FD002) curves showed an initial reduction in error, followed by a steady increase as they approached the EOL. This indicates that the FD002 dataset helps the DANN models primarily in learning the more general early-life behaviour of engine degradation but is not sufficient for accurate RUL predictions near EOL, since the models trained with FD003 display the expected bell-shaped curve.

6.6.4 Effectiveness of Supervised Transfer Learning

In response to [R4](#), it is observed that in the high-data range the TL strategies did not provide a benefit and instead caused a negative transfer compared to GRU. However, the benefit of TL became more evident as data scarcity settings became scarcer. In particular in low-engine scenarios, all supervised TL techniques improved performance, provided that they were trained with the FD003 dataset as the source. Across the evaluated low- to extreme-scarcity settings, the FD003-trained TL configurations generally achieved better performance than the GRU baseline.

In further response to [R4](#) and [R7](#), it was found that TL models using FD002, which has a larger domain shift, as the source data set generally struggled to outperform the baseline, providing an advantage only under the most data-scarce conditions. In contrast, TL models trained on FD003, which has a smaller domain shift, began to offer improvements

much earlier in low-engine settings (20 to 10 engines). Additionally, models trained with FD003 showed less pronounced negative transfer at higher data availability levels compared to those trained with FD002 as the source. This discrepancy highlights the difficulty for the supervised TL models to adapt to the target domain when the domain shift is large. Moreover, in response to R1, the best performing TL strategy was dependent on the data availability scenario, as well as the source data set used. In the higher-data scenarios, TL3 with both FD002 and FD003 performed the best. TL3 with FD003 continued to be the top-performing model in the moderate-data ranges. However, in the lower-data settings, TL1 and TL2 with FD003 performed consistently better. Of the FD002-trained models, TL3 showed the best performance in high to moderate-data settings, while TL1 showed the best performance in lower-data settings.

6.6.5 Source-Only Transfer and Unsupervised Domain Adaptation

In response to whether domain adaptation strategies that function without labelled target data achieve better performance than models trained only on source domain data (R5 and R6), the results showed the metric scores for two variants of DANN, namely LSTM-DANN and GRU-DANN, as well as GRU models trained only on the source domain. To further respond to R6, the supervised GRU was included for comparison. As expected, in the high- to moderate-data ranges, the supervised GRU performed substantially better than the unsupervised TL approaches. This is because the supervised GRU has access to a sufficient amount of target labelled data in these settings. In particular, the source-only baseline performed better than the DANN techniques when trained on the FD003 dataset, which has a domain gap closer to FD001. However, it consistently performed the worst when trained on the FD002 dataset, which has a larger domain gap. This was observed across all data ranges, where in the scenario where only FD002 is available for source training, the DANN variants consistently outperformed the source-only model. Within the evaluated source-target configurations, this pattern suggests that adversarial adaptation was beneficial in the larger FD002 to FD001 domain-shift setting, whereas the source-only approach remained competitive in the smaller FD003 to FD001 setting. Furthermore, although the supervised model shows the expected superior performance in the high to moderate-data ranges, it was observed that as the data become more scarce, the source-only and DANN models become more competitive. In particular, in the low-data range, the source-only baseline trained in FD003 outperformed the supervised GRU in terms of RMSE, MAE and R^2 . Moreover, in the very-low-data range, the DANN variants also showed better performance for these metrics, and in the extreme-scarcity settings, all source-only and DANN models performed better than the supervised GRU.

However, as exposed by Figure 6.13, the models trained with FD002 struggled to make near-EOL predictions compared to the supervised GRU. Nevertheless, the DANN models showed a clear benefit compared to the source-only model by providing superior early-life

predictions and competitive predictions near EOL. In contrast, the unsupervised models trained with FD003 showed a good ability to make predictions in the early- and near-EOL RUL prediction cycles, but the supervised GRU showed superiority for mid-life RUL predictions (Figure 6.17).

To provide contextual comparison with the published LSTM-DANN results Table 6.8 presents the LSTM-DANN RMSE results of Da Costa et al. (2020), whose LSTM-DANN architecture and selected hyperparameter configurations were adopted for this study.

Table 6.8: Contextual comparison of LSTM-DANN test RMSE with the results reported by Da Costa et al. Da Costa et al. (2020).

Transfer Setting	Da Costa et al.	Present Study
FD002 to FD001	28.10 ± 5.03	24.39
FD003 to FD001	31.74 ± 9.37	21.36

It is observed that the present study achieves notably lower RMSE values. However even though the current implementation produced lower RMSE scores for both transfer scenarios, it is necessary to note that the methodologies between Da Costa et al. (2020) and the present study differs. For example Da Costa et al. (2020) kept all 21 sensors and independently normalised each domain. The current methodology utilises the 14 correlation features chosen and standardised. In addition some of the discrepancies may be indicative of the inherent stochasticity associated with training DANN. Da Costa et al. (2020) indicated that there was significant variation amongst runs, specifically the standard deviation was approximately 5 RUL cycles for FD002 to FD001 and greater than 9 RUL cycles for FD003 to FD001. Other discrepancies include data preprocessing steps where Da Costa et al. (2020) used min-max scaler as opposed to standardisation. Despite these differences, the performance improvement achieved in the present study appears substantial. Future work should therefore investigate the sensitivity of DANN performance to the selected feature space, particularly the influence of non-informative or constant sensor measurements on domain alignment and RUL prediction.

6.6.6 Sensitivity Analysis

In order to assess the stability of the discussed findings, a sensitivity analysis is presented. Specifically, five seeds (42, 84, 123, 222 and 333) were used to evaluate representative models, including Lasso, Poly2, XGB, GRU, Transformer, all three fine-tuning techniques, and both the LSTM-DANN and GRU-DANN implementations. Each seed was applied to the training-engine subset selection and model training. Furthermore, the analysis considers the 40, 20, 10 and 5 training-engine scenarios, where greater variability is expected as the available training data decrease. Table 6.9 shows the sensitivity of test RMSE to the

selected training-engine count. The transfer-learning and domain-adaptation results are reported for both the FD003 to FD001 and FD002 to FD001 settings..

Table 6.9: Sensitivity analysis of test RMSE across selected training-engine counts, reported as mean $\pm$ standard deviation across five different seeded runs.

Model	40 eng	20 eng	10 eng	5 eng
FD003 to FD001				
TL1	15.44 ± 0.46	16.15 ± 0.39	17.05 ± 0.50	17.35 ± 0.41
TL2	16.90 ± 0.18	16.93 ± 0.25	17.25 ± 0.16	17.41 ± 0.22
TL3	15.26 ± 0.52	15.79 ± 0.32	17.74 ± 2.13	18.06 ± 0.55
LSTM-DANN	21.75 ± 0.84	21.53 ± 0.60	21.46 ± 0.44	21.70 ± 0.73
GRU-DANN	20.02 ± 0.74	20.12 ± 0.46	20.23 ± 0.74	20.55 ± 0.97
FD002 to FD001				
TL1	15.94 ± 0.41	16.79 ± 0.28	19.08 ± 1.99	20.69 ± 0.91
TL2	21.58 ± 0.67	21.93 ± 0.55	23.38 ± 1.48	25.07 ± 2.01
TL3	15.97 ± 0.35	16.62 ± 0.25	18.87 ± 2.72	20.34 ± 0.63
LSTM-DANN	26.78 ± 3.05	25.60 ± 2.01	30.90 ± 4.85	29.40 ± 2.73
GRU-DANN	29.29 ± 4.91	29.77 ± 3.83	33.59 ± 4.91	32.38 ± 6.32

From Table 6.9 the repeated runs show that the main model-family trends reported broadly align with the original experiments. The linear models remain comparatively stable as the amount of training data decreases. The GRU and Transformer continue to provide the strongest baseline performance in the higher-data scenarios. Notably, the poorer-performing models generally show greater variability. For example, GRU-DANN using FD002 as the source domain achieves a mean RMSE of 33.59 at 10 engines with a standard deviation of 4.91, while LSTM-DANN achieves 30.90 ± 4.85 under the same setting. In comparison, the GRU at 40 engines achieves a substantially lower mean RMSE of 13.97 with a standard deviation of only 0.38.

The repeated-seed results provide additional context to the previously reported Transformer robustness. Across the selected 40, 20, 10 and 5 engine scenarios, the GRU achieves a lower mean RMSE than the Transformer. However, the earlier robustness finding was primarily associated with the more extreme sub-five-engine scenarios, where the GRU performance degraded more sharply while the Transformer remained comparatively stable. A similar tendency is visible in the sensitivity analysis at the lower-data settings, where the Transformer shows lower variability than the GRU at both 10 engines (0.89 compared with 1.41) and 5 engines (1.13 compared with 1.78). This suggests that although the GRU achieves the lower mean RMSE at these selected engine counts, the Transformer remains slightly less sensitive to variation between seeded runs as the available training

data decrease.

For supervised TL, the repeated runs continue to support the general advantage of the FD003 to FD001 setting over FD002 to FD001, consistent with the smaller source-target discrepancy previously identified. However, the exact ranking between closely performing strategies should not be over-interpreted. For example, at 10 engines in the FD003 to FD001 setting, TL2 achieves 17.25 ± 0.16 RMSE while TL3 achieves 17.74 ± 2.13 . The relatively large variation in TL3 compared with the small difference in mean RMSE indicates that it would not be appropriate to make a strong claim regarding which of the two strategies is definitively superior in this range. A similar caution applies to other closely performing configurations.

The DANN results also remain consistent with the previously reported domain-shift findings. For both the LSTM-DANN and GRU-DANN implementations, the FD003 to FD001 configuration remains comparatively stable across the evaluated engine counts, whereas the FD002 to FD001 configuration exhibits both higher error and greater variability. For example, LSTM-DANN achieves standard deviations below one RUL cycle across all evaluated FD003 to FD001 scenarios, compared with standard deviations ranging from 2.01 to 4.85 for FD002 to FD001. This further indicates that the larger domain discrepancy between FD002 and FD001 presents a more challenging adaptation setting.

It should be acknowledged that the sensitivity analysis did not repeat the hyperparameter optimisation process across different Optuna seeds or alternative hyperparameter search ranges. Due to the wide experimental scope of the study, repeating the complete hyperparameter optimisation process across multiple seeds and model configurations was not computationally feasible. The selected hyperparameters were therefore retained during the repeated-seed analysis so that variability associated with training-data selection and stochastic model training could be assessed independently of changes in model configuration. Therefore, the sensitivity of the reported results to alternative optimisation outcomes was not quantified and remains an area for future investigation.

6.6.7 Limitations and Future Work

Although the findings provide key insight into RUL prediction under limited labelled target-domain-data, several limitations should be noted. First, the experiments were conducted only on the C-MAPSS dataset. It should be acknowledged that the experiments conducted in this study is done on a synthetic dataset and may not represent the complexities of real-world data where non-simulated variability exists. Although C-MAPSS is a widely used benchmark for RUL prediction (as seen in [section A.5](#), which discusses the types of data used in PdM research), degradation patterns can vary between different PdM datasets as they can differ in equipment types, operating environments, and fail-

ure modes. Furthermore, as shown in [Figure A.7](#), PdM applications span many forms of industrial equipment, which may exhibit substantially different degradation patterns. Therefore, future work should investigate whether the observed model behaviours remain consistent when applied to other PdM datasets from different domains.

It should also be noted that Lasso and Ridge were implemented in this study as regularised linear baselines using the original feature space. This enabled direct comparison with the non-regularised MLR model, while non-linear were investigated separately through the polynomial regression models. However, it is acknowledged that applying L1 or L2 regularisation to an expanded polynomial feature space could provide additional benefit by controlling the increased model complexity and reducing overfitting. Regularised polynomial models were not evaluated in the present experimental framework and could therefore be investigated in future work.

Furthermore, the experiments were trained using fixed hyperparameter configurations obtained from the 80-engine setting. This helped keep the experiments controlled and comparable across data-availability scenarios. However, these hyperparameters may not be optimal for the lower-data settings, where the best model configuration could differ as the amount of available training data becomes more limited. Future work could therefore assess how hyperparameter optimisation behaves under progressively reduced training data and whether models require different configurations in low-data and extreme-scarcity settings.

This limitation is particularly relevant for the source-only models. The source-only configurations were selected based on performance in relation to the FD001 target setting and may not have been optimally configured for the source datasets themselves. As a result, the comparison between source-only models and DANN variants may be influenced by the chosen hyperparameter configuration. Future work could therefore optimise source-only and DANN models separately for each source-target setup to determine whether the relative performance remains consistent.

It should also be noted that the extreme-scarcity scenarios refer specifically to the amount of labelled training data available, since the experimental design retained a fixed validation set of 20 labelled engines. In a real-world setting with extreme labelled-data scarcity, such a validation set may not be available, and model hyperparameters may consequently be less reliably selected. Future work could therefore investigate model selection and validation strategies in which the validation-set size is reduced along with the available training data.

Another limitation is that the main experimental results were obtained using a single fixed random seed of 42. Although this provided a controlled and reproducible experimental process, stochastic variability was not quantified across every model configuration and data-availability scenario. To provide additional insight into this limitation, a supplementary sensitivity analysis was conducted across five different random seeds for selected

representative models and engine-count scenarios (see [Table 6.9](#)). The analysis showed that the broader model-performance trends remained generally consistent, although some closely performing configurations showed variability in their relative performance. Therefore, small differences between closely competing models should still be interpreted with caution. Future work could extend this analysis by repeating the sensitivity analysis across a larger number of random seeds and all data scenarios.

Finally, the supervised fine-tuning TL strategies showed promise in lower-data settings when the source domain was sufficiently similar to the target domain. However, the benefit was less clear when the domain shift was more severe. Future work could therefore explore semi-supervised domain adaptation architectures that combine unsupervised domain alignment with supervised fine-tuning. For example, a DANN-based architecture could be extended with an adaptive supervised prediction head to utilise labelled source data, unlabelled target data, and limited labelled target data within a single architecture.

6.6.8 Synthesis of Findings

The findings directly support the objective of examining the difficulty of deploying RUL prediction models in PdM settings where labelled degradation data from the target domain are scarce or unavailable and where source-to-target domain shifts hinder model generalisation. Through the systematic assessment of models across different data-availability scenarios, including the case where no labelled target data are available, it is evident that no single modelling strategy is optimal across all scenarios. Instead, the most suitable approach depends on the amount of available labelled target data, the severity of the domain shift, and whether the model is assessed using penalised scores, computational efficiency, aggregate accuracy, or lifecycle-specific reliability. In general, neural-network-based models benefit from higher data availability, while the Transformer showed stronger robustness as target data became scarce. Within the evaluated configurations, supervised fine-tuning techniques provided an advantage when labelled target data were limited, particularly for the more similar investigated source-target setting. However, when labelled target data are unavailable, the results show that the choice between source-only transfer and adversarial domain adaptation strongly depends on the source-target domain gap. If the domain shift is relatively small, source-only transfer can remain competitive. However, when the domain shift is more substantial, DANN provides a notable benefit over the source-only approach. Therefore, when selecting a model for low-data PdM applications, it is important to first define the performance requirements and assess the amount and type of data available, as well as the expected source-target domain similarity, before committing to a specific algorithm and strategy.

Chapter 7

Conclusion

The prediction of RUL is a key component of predictive maintenance (PdM) systems, where data-driven models are used to identify degradation patterns and predict imminent equipment failure. This study investigates the challenge of implementing RUL prediction models for PdM when the availability of labelled target domain training data is restricted. Moreover, when source domain data is utilised, it investigates how the challenge of domain shift can affect model performance.

To address this problem, the study systematically evaluated RUL prediction strategies across varying levels of labelled target-data availability. This included the assessment of baseline regression, tree-based, and neural-network-based models, as well as FS, FE, supervised transfer learning, source-only transfer, and unsupervised domain adaptation.

7.1 Achievement of Research Objectives

To achieve the aim of the study (stated in [subsection 1.1.3](#)) 11 objectives were met. First, the study presented an SLR in [Appendix A](#) that identified the key challenges, prediction algorithms, optimisation approaches, and evaluation metrics used in PdM for the prediction of RUL.

Secondly, in [Chapter 3](#), a suitable dataset was selected that includes multiple operating conditions and fault mode variations. This enabled the investigation of different ML models as well as domain shift analysis and the implementation of TL strategies. The selected dataset was the C-MAPSS dataset. Specifically, the FD001 was used as the target dataset and FD002 and FD003 as source datasets.

The third objective was met in [Chapter 4](#), where the experimental methodology was defined. It specifically presented the design of the preprocessing, model training, validation,

and evaluation stages that were applied for all experiments (including the baseline, feature engineering, and transfer learning phases). The FS experiments evaluated representative models using reduced feature subsets, while the FE experiments combined the GRU baseline with CNN, TCN, and AE-based FE components. The transfer learning experiments further evaluated supervised fine-tuning, source-only transfer, and DANN-based domain adaptation. In addition, [Chapter 4](#) defined the evaluation metrics used in the study. This includes the AUC-RMSE metric for assessing RUL prediction reliability across the equipment RUL cycles (objective four).

To achieve the fifth objective, [Chapter 2](#) provided a theoretical overview of the techniques implemented in this study. This included regression, tree-based models, neural-network-based models, as well as ML techniques including FS, FE, and TL methods.

Objectives six to eleven were met through the implementation and validation as well as the test-set evaluation ([Chapter 5](#) and [Chapter 6](#) respectively). This specifically addressed the experimental objectives related to model robustness, FS, FE, supervised TL, source-only transfer, and unsupervised domain adaptation under different levels of target-data availability.

7.2 Key Findings and Contributions

The findings within the investigated C-MAPSS datasets and experimental design show that relative model performance was dependent on the availability of labelled target-domain data, the similarity of the investigated source-target domains, and the evaluation criteria used. It was observed that the model that performs best under higher-data conditions is not necessarily the most robust when data becomes scarce. For example, while the GRU achieved the strongest overall baseline performance in higher-data settings, the Transformer showed greater robustness as the available labelled target data were reduced. Moreover, in extreme data-scarcity settings, linear and tree-based models became more competitive. These results indicate that, within the evaluated scenarios, model performance under sufficient data availability alone was not representative of performance under more constrained data conditions.

Furthermore, it was observed that FS provided its most notable benefit through reduced computational cost, particularly for recurrent neural-network-based models. However, FS did not provide consistent predictive improvements across all models. Models such as the Transformer and Poly2 showed advantages in certain scenarios, whereas the GRU and LSTM showed degraded performance.

Another key finding was that, within the investigated datasets, the AUC-RMSE metric provided information that was not always reflected by aggregate RMSE alone. By eval-

uating prediction error across different RUL regions, AUC-RMSE provided an additional perspective on model behaviour across early-life, mid-life, and near-EOL stages. The relative performance of several models differed depending on whether RMSE or AUC-RMSE was considered. Moreover, the AUC-RMSE lifecycle plots were useful for identifying the lifecycle stages at which each evaluated model performed best and worst.

The TL and domain-adaptation results further showed that, across the investigated C-MAPSS source-target combinations, source-domain knowledge could improve predictive performance when labelled target data were limited or unavailable. However, the magnitude of this benefit varied according to the investigated source-target domain relationship, quantified using MMD. Supervised fine-tuning provided the greatest improvements when labelled target data were limited and the investigated source domain was more similar to the target domain. When labelled target data were unavailable, source-only transfer remained competitive in the smaller domain-shift setting, while DANN provided a clearer benefit in the larger domain-shift setting. These findings indicate that, within the evaluated scenarios, the relative effectiveness of the investigated transfer-learning strategies varied according to target-data availability and source-target similarity.

7.2.1 Practical Strategy for RUL Model Selection

Based on the findings of this study, selection of an appropriate RUL prediction strategy should not be based on a single modelling approach. It is instead dependent on several factors including availability of data, domain shifts between target and source domains, and application-specific evaluation requirements. [Figure 7.1](#) presents a high-level practical approach when making model selection decisions.

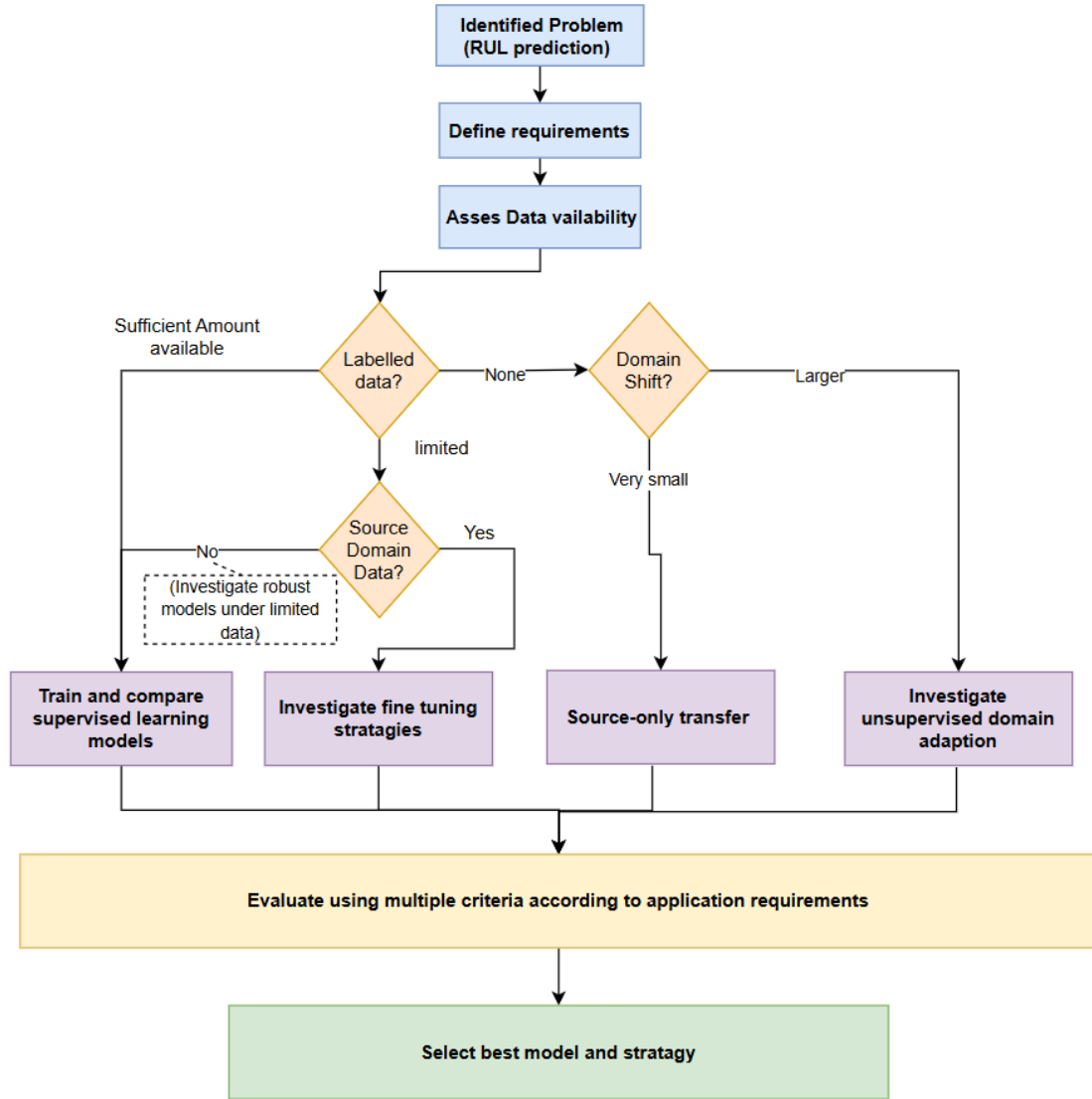


Figure 7.1: Practical high-level strategy for RUL model selection based on data availability, source-target domain shift, and application-specific evaluation requirements.

If there exists enough labelled data in the target-domain, then supervised models may be trained and evaluated separately in both domains. Under the experimental configurations of this study, neural-network based models were shown to perform better than other types of models under high-data conditions. However, although the model that performs best under high-data conditions was not always the one that could retain its predictive ability under low-data conditions. For example, the Transformer improved in relative ranking as the amount of labelled target-domain training data was progressively reduced. Similarly, simpler regression models, such as linear and polynomial regression, and tree-based models became more competitive under extreme data scarcity. The expected availability of labelled target data should therefore be considered when selecting candidate models rather than relying solely on their performance under high-data conditions.

If only limited amounts of labelled target-domain data are available, then additional sources of improvement in prediction accuracy may come from labelled source-domain data. When suitable labelled source-domain data are available, supervised fine tuning approaches may be useful. However, whether transfer learning improves prediction performance depends heavily on how closely related the source and target domains are to each other. The most significant benefits from transfer learning occur when the source domain is very similar to the target domain. However, the degree to which transfer learning is beneficial decreases rapidly as the degree of difference between the source and target domains increases. If no suitable labelled source-domain data are available, then the primary concern for selecting candidate models will be those that demonstrate good predictive ability when trained using a limited amount of labelled target-domain data.

When no labelled target-domain data are available but suitable labelled source-domain data exist, the relationship between the source and target domains becomes important. Amongst all of the examined source-target pairs, the performance of source-only transfer remained strong even though the degrees of similarity between each pair varied. However, as the degree of difference between the source- and target-domains increased, the performance advantages of DANN based unsupervised domain adaptation approaches became much more pronounced compared to simply training and testing directly on the target-domain after training on the source-domain. As such, if there is no labelled data available in the target-domain but there does exist a potentially suitable set of source-domain data, then assessing how strongly related each domain is before deciding whether to utilise a direct source-only transfer methodology or an unsupervised domain adaptation methodology would be recommended.

Furthermore when making decisions about selecting a model for use within a practical implementation, the best approach is to choose a model that fits the needs of the intended practical application based on its performance as determined by multiple metrics (aggregate) for accuracy of prediction and other factors including lifecycle specific reliability, computational costs and the consequences of any errors made with regard to predictions. For example, since RMSE can serve as an indication of total prediction error, the AUC-RMSE can also illustrate where different models perform differently over the entire RUL lifecycle which could be obscured when comparing RMSE for each model individually. In addition, the C-MAPSS score evaluates prediction error by penalising RUL over-predictions more heavily than under-predictions, reflecting the greater risk associated with predicting that equipment has more remaining life than it actually does.

In response to the aim, defined in [subsection 1.1.3](#), these findings provide guidance in selecting RUL prediction models under constrained PdM data conditions within the investigated C-MAPSS scenarios. It is evident from the investigated results that the selected approach must consider the environment of the PdM setting and be based on the amount of labelled target data available, the degree of source-target similarity and the evaluation

requirements. Future work should further validate these findings through more extensive repeated-seed experimentation across the complete model set and through evaluation on additional datasets to assess their generalisability.

Bibliography

- Achouch, M., Dimitrova, M., et al. (2022). “On Predictive Maintenance in Industry 4.0: Overview, Models, and Challenges”. *Applied Sciences*, 12(16):8081. DOI: [10.3390/app12168081](https://doi.org/10.3390/app12168081).
- Adryan Fitra Azyus, Sastra Kusuma Wijaya, et al. (2025). “Remaining Useful Life Prediction of Turbofan Engines Using CNN-GRU: A Comparative Study on C-MAPSS Dataset”. *Journal of Mechanical, Civil and Industrial Engineering*, 6(1):19–27. DOI: [10.32996/jmcie.2025.6.1.3](https://doi.org/10.32996/jmcie.2025.6.1.3).
- Ahuja, I. and Khamba, J. (2008). “Total productive maintenance: literature review and directions”. *International Journal of Quality & Reliability Management*, 25(7):709–756. DOI: [10.1108/02656710810890890](https://doi.org/10.1108/02656710810890890).
- Akiba, T., Sano, S., et al. (July 2019). “Optuna: A Next-generation Hyperparameter Optimization Framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage AK USA: ACM:pp. 2623–2631. DOI: [10.1145/3292500.3330701](https://doi.org/10.1145/3292500.3330701).
- Alkhalifah, T., Wang, H., et al. (2022). “MLReal: Bridging the gap between training on synthetic data and real data applications in machine learning”. *Artificial Intelligence in Geosciences*, 3:101–114. DOI: [10.1016/j.aiig.2022.09.002](https://doi.org/10.1016/j.aiig.2022.09.002).
- Alzubi, J., Nayyar, A., et al. (2018). “Machine Learning from Theory to Algorithms: An Overview”. *Journal of Physics: Conference Series*, 1142:012012. DOI: [10.1088/1742-6596/1142/1/012012](https://doi.org/10.1088/1742-6596/1142/1/012012).
- Apicella, A., Isgrò, F., et al. (2025). “Don’t push the button! Exploring data leakage risks in machine learning and transfer learning”. *Artificial Intelligence Review*, 58(11):339. DOI: [10.1007/s10462-025-11326-3](https://doi.org/10.1007/s10462-025-11326-3).
- Aremu, O.O., Hyland-Wood, D., et al. (2019). “A Relative Entropy Weibull-SAX framework for health indices construction and health stage division in degradation modeling of multivariate time series asset data”. *Advanced Engineering Informatics*, 40:121–134. DOI: [10.1016/j.aei.2019.03.003](https://doi.org/10.1016/j.aei.2019.03.003).
- Arnold, C., Biedebach, L., et al. (2024). “The role of hyperparameters in machine learning models and how to tune them”. *Political Science Research and Methods*, 12(4):841–848. DOI: [10.1017/psrm.2023.61](https://doi.org/10.1017/psrm.2023.61).

- Asutkar, S. and Tallur, S. (2023). “Deep transfer learning strategy for efficient domain generalisation in machine fault diagnosis”. *Scientific Reports*, 13(1):6607. DOI: [10.1038/s41598-023-33887-5](https://doi.org/10.1038/s41598-023-33887-5).
- Aszemi, N.M. and Dominic, P. (2019). “Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms”. *International Journal of Advanced Computer Science and Applications*, 10(6). DOI: [10.14569/IJACSA.2019.0100638](https://doi.org/10.14569/IJACSA.2019.0100638).
- Aydemir, G. and Paynabar, K. (2020). “Image-Based Prognostics Using Deep Learning Approach”. *IEEE Transactions on Industrial Informatics*, 16(9):5956–5964. DOI: [10.1109/TII.2019.2956220](https://doi.org/10.1109/TII.2019.2956220).
- Ayvaz, M. and De Lathauwer, L. (2022). “CPD-Structured Multivariate Polynomial Optimization”. *Frontiers in Applied Mathematics and Statistics*, 8:836433. DOI: [10.3389/fams.2022.836433](https://doi.org/10.3389/fams.2022.836433).
- Bai, S., Kolter, J.Z., et al. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. DOI: [10.48550/ARXIV.1803.01271](https://doi.org/10.48550/ARXIV.1803.01271).
- Baidu, I. (2013). *S.M.A.R.T. dataset*. URL: <https://pan.baidu.com/share/link?shareid=%20189977&uk=4278294944#list/path=%2F> (visited on 04/09/2024).
- Bala, A., Ismail, I., et al. (2020). “An Improved Grasshopper Optimization Algorithm Based Echo State Network for Predicting Faults in Airplane Engines”. *IEEE Access*, 8:159773–159789. DOI: [10.1109/ACCESS.2020.3020356](https://doi.org/10.1109/ACCESS.2020.3020356).
- Bank, D., Koenigstein, N., et al. (2023). “Autoencoders”. In: *Machine Learning for Data Science Handbook*. Ed. by L. Rokach, O. Maimon, et al. Cham: Springer International Publishing:pp. 353–374. DOI: [10.1007/978-3-031-24628-9_16](https://doi.org/10.1007/978-3-031-24628-9_16).
- Bauer, R. (2018). *Backblaze Hard Drive stats*. URL: <https://www.backblaze.com/blog/backblaze-hard-drive-stats-q1-2019/> (visited on 04/09/2024).
- Beg, A.H. and Islam, M.Z. (June 2016). “Advantages and limitations of genetic algorithms for clustering records”. In: *2016 IEEE 11th Conference on Industrial Electronics and Applications (ICIEA)*. Hefei, China: IEEE:pp. 2478–2483. DOI: [10.1109/ICIEA.2016.7604009](https://doi.org/10.1109/ICIEA.2016.7604009).
- Belete, D.M. and Huchaiah, M.D. (2022). “Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results”. *International Journal of Computers and Applications*, 44(9):875–886. DOI: [10.1080/1206212X.2021.1974663](https://doi.org/10.1080/1206212X.2021.1974663).
- Beltrami, M. and Da Silva, A.C.L. (2020). “A grid-quadtrees model selection method for support vector machines”. *Expert Systems with Applications*, 146:113172. DOI: [10.1016/j.eswa.2019.113172](https://doi.org/10.1016/j.eswa.2019.113172).
- Berahmand, K., Daneshfar, F., et al. (2024). “Autoencoders and their applications in machine learning: a survey”. *Artificial Intelligence Review*, 57(2):28. DOI: [10.1007/s10462-023-10662-6](https://doi.org/10.1007/s10462-023-10662-6).
- Bergstra, J. and Bengio, Y. (2012). “Random search for hyper-parameter optimization”. *Journal of machine learning research*, 13(2):282–205.
- Biau, G. (2012). “Analysis of a random forests model”. *The Journal of Machine Learning Research*, 13:1063–1095.

- Birkel, C. (2017). *Oxford Battery Degradation Dataset 1*. DOI: [10.5287/BODLEIAN:K02KDMYGG](https://doi.org/10.5287/BODLEIAN:K02KDMYGG).
- Bischl, B., Binder, M., et al. (2023). “Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges”. *WIREs Data Mining and Knowledge Discovery*, 13(2):e1484. DOI: [10.1002/widm.1484](https://doi.org/10.1002/widm.1484).
- Breiman, L. (2001). “Random Forests”. *Machine Learning*, 45(1):5–32. DOI: [10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324).
- Brookshire, G., Kasper, J., et al. (2024). “Data leakage in deep learning studies of translational EEG”. *Frontiers in Neuroscience*, 18:1373515. DOI: [10.3389/fnins.2024.1373515](https://doi.org/10.3389/fnins.2024.1373515).
- Bui, Q.-T., Chou, T.-Y., et al. (2021). “Gradient Boosting Machine and Object-Based CNN for Land Cover Classification”. *Remote Sensing*, 13(14):2709. DOI: [10.3390/rs13142709](https://doi.org/10.3390/rs13142709).
- Carvalho, T.P., Soares, F.A.A.M.N., et al. (2019). “A systematic literature review of machine learning methods applied to predictive maintenance”. *Computers & Industrial Engineering*, 137:106024. DOI: [10.1016/j.cie.2019.106024](https://doi.org/10.1016/j.cie.2019.106024).
- Castro, H. and Ferreira, J. (2022). “Linear and logistic regression models: when to use and how to interpret them?” *Jornal Brasileiro de Pneumologia*:e20220439. DOI: [10.36416/1806-3756/e20220439](https://doi.org/10.36416/1806-3756/e20220439).
- Catelani, M., Ciani, L., et al. (2021). “Remaining Useful Life Estimation for Prognostics of Lithium-Ion Batteries Based on Recurrent Neural Network”. *IEEE Transactions on Instrumentation and Measurement*, 70:1–11. DOI: [10.1109/TIM.2021.3111009](https://doi.org/10.1109/TIM.2021.3111009).
- Chai, T. and Draxler, R.R. (2014). “Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature”. *Geoscientific Model Development*, 7(3):1247–1250. DOI: [10.5194/gmd-7-1247-2014](https://doi.org/10.5194/gmd-7-1247-2014).
- Chan, F., Lau, H., et al. (2005). “Implementation of total productive maintenance: A case study”. *International Journal of Production Economics*, 95(1):71–94. DOI: [10.1016/j.ijpe.2003.10.021](https://doi.org/10.1016/j.ijpe.2003.10.021).
- Chen, B. and Chen, S. (2026). “Neural Networks”. In: *Machine Vision Technology*. Series Title: Advanced and Intelligent Manufacturing in China. Singapore: Springer Nature Singapore:pp. 251–272. DOI: [10.1007/978-981-95-1184-6_15](https://doi.org/10.1007/978-981-95-1184-6_15).
- Chen, D., Hong, W., et al. (2022). “Transformer Network for Remaining Useful Life Prediction of Lithium-Ion Batteries”. *IEEE Access*, 10:19621–19628. DOI: [10.1109/ACCESS.2022.3151975](https://doi.org/10.1109/ACCESS.2022.3151975).
- Chen, J.C., Chen, T.-L., et al. (2021). “Combining empirical mode decomposition and deep recurrent neural networks for predictive maintenance of lithium-ion battery”. *Advanced Engineering Informatics*, 50:101405. DOI: [10.1016/j.aei.2021.101405](https://doi.org/10.1016/j.aei.2021.101405).
- Chen, T. and Guestrin, C. (Aug. 2016). “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco California USA: ACM:pp. 785–794. DOI: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785).

- Cheng, Y., Wu, J., et al. (2021). “Remaining Useful Life Prognosis Based on Ensemble Long Short-Term Memory Neural Network”. *IEEE Transactions on Instrumentation and Measurement*, 70:1–12. DOI: [10.1109/TIM.2020.3031113](https://doi.org/10.1109/TIM.2020.3031113).
- Chin, C., Abdullah, S., et al. (2024). “A review of the wavelet transform for durability and structural health monitoring in automotive applications”. *Alexandria Engineering Journal*, 99:204–216. DOI: [10.1016/j.aej.2024.04.069](https://doi.org/10.1016/j.aej.2024.04.069).
- Cho, H., She, J., et al. (2024). “Machine Learning and Health Science Research: Tutorial”. *Journal of Medical Internet Research*, 26:e50890. DOI: [10.2196/50890](https://doi.org/10.2196/50890).
- Chollet, F. and Watson, M. (2025). *Deep Learning With Python*. 3rd ed. Manning.
- Chung, J., Gulcehre, C., et al. (2014). *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. Version Number: 1. DOI: [10.48550/ARXIV.1412.3555](https://doi.org/10.48550/ARXIV.1412.3555).
- Chung, J., Gulcehre, C., et al. (2015). “Gated Feedback Recurrent Neural Networks”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Vol. 37:pp. 2067–2075.
- Ciampiconi, L., Elwood, A., et al. (2023). *A survey and taxonomy of loss functions in machine learning*. Version Number: 2. DOI: [10.48550/ARXIV.2301.05579](https://doi.org/10.48550/ARXIV.2301.05579).
- Coble, J., Ramuhalli, P., et al. (2020). “A Review of Prognostics and Health Management Applications in Nuclear Power Plants”. *International Journal of Prognostics and Health Management*, 6(3). DOI: [10.36001/ijphm.2015.v6i3.2271](https://doi.org/10.36001/ijphm.2015.v6i3.2271).
- Cohen, G. (2024). “Polynomial Moving Regression Band Stocks Trading System”. *Risks*, 12(10):166. DOI: [10.3390/risks12100166](https://doi.org/10.3390/risks12100166).
- Costello, J.J.A., West, G.M., et al. (2017). “Machine Learning Model for Event-Based Prognostics in Gas Circulator Condition Monitoring”. *IEEE Transactions on Reliability*, 66(4):1048–1057. DOI: [10.1109/TR.2017.2727489](https://doi.org/10.1109/TR.2017.2727489).
- Da Costa, P.R.D.O., Akçay, A., et al. (2020). “Remaining useful lifetime prediction via deep domain adaptation”. *Reliability Engineering & System Safety*, 195:106682. DOI: [10.1016/j.ress.2019.106682](https://doi.org/10.1016/j.ress.2019.106682).
- Da Silva, R.F., Melani, A.H.D.A., et al. (2023). “Reliability and Risk Centered Maintenance: A Novel Method for Supporting Maintenance Management”. *Applied Sciences*, 13(19):10605. DOI: [10.3390/app131910605](https://doi.org/10.3390/app131910605).
- Daigle, M. and Goebel, K. (2021). “Model-based Prognostics with Fixed-lag Particle Filters”. In: *Annual Conference of the PHM Society*. Vol. 1. URL: <https://papers.phmsociety.org/index.php/phmconf/article/view/1598>.
- Daily, J. and Peterson, J. (2017). “Predictive Maintenance: How Big Data Analysis Can Improve Maintenance”. In: *Supply Chain Integration Challenges in Commercial Aerospace*. Ed. by K. Richter and J. Walther. Cham: Springer International Publishing:pp. 267–278. DOI: [10.1007/978-3-319-46155-7_18](https://doi.org/10.1007/978-3-319-46155-7_18).
- Dalzochio, J., Kunst, R., et al. (2020). “Machine learning and reasoning for predictive maintenance in Industry 4.0: Current status and challenges”. *Computers in Industry*, 123:103298. DOI: [10.1016/j.compind.2020.103298](https://doi.org/10.1016/j.compind.2020.103298).

- Dar, I.S. and Chand, S. (2024). “Bootstrap-quantile ridge estimator for linear regression with applications”. *PLOS ONE*, 19(4). Ed. by M.R. Abonazel:e0302221. DOI: [10.1371/journal.pone.0302221](https://doi.org/10.1371/journal.pone.0302221).
- Das, S., Tariq, A., et al. (2023). “Recurrent Neural Networks (RNNs): Architectures, Training Tricks, and Introduction to Influential Research”. In: *Machine Learning for Brain Disorders*. Ed. by O. Colliot. Vol. 197. Series Title: Neuromethods. New York, NY: Springer US:pp. 117–138. DOI: [10.1007/978-1-0716-3195-9_4](https://doi.org/10.1007/978-1-0716-3195-9_4).
- De Santo, A., Galli, A., et al. (2022). “Deep Learning for HDD Health Assessment: An Application Based on LSTM”. *IEEE Transactions on Computers*, 71(1):69–80. DOI: [10.1109/TC.2020.3042053](https://doi.org/10.1109/TC.2020.3042053).
- Devlin, J., Chang, M.-W., et al. (2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Minneapolis, Minnesota: Association for Computational Linguistics:pp. 4171–4186. DOI: [10.18653/v1/N19-1423](https://doi.org/10.18653/v1/N19-1423).
- Diao, E., Ding, J., et al. (Dec. 2019). “Restricted Recurrent Neural Networks”. In: *2019 IEEE International Conference on Big Data (Big Data)*. Los Angeles, CA, USA: IEEE:pp. 56–63. DOI: [10.1109/BigData47090.2019.9006257](https://doi.org/10.1109/BigData47090.2019.9006257).
- Dobra, A. and Gehrke, J. (July 2002). “SECRET: a scalable linear regression tree algorithm”. In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. Edmonton Alberta Canada: ACM:pp. 481–487. DOI: [10.1145/775047.775117](https://doi.org/10.1145/775047.775117).
- Dosovitskiy, A., Beyer, L., et al. (2021). “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *International Conference on Learning Representations*:pp. 1–22.
- Dubey, S.R., Singh, S.K., et al. (2022). “Activation functions in deep learning: A comprehensive survey and benchmark”. *Neurocomputing*, 503:92–108. DOI: [10.1016/j.neucom.2022.06.111](https://doi.org/10.1016/j.neucom.2022.06.111).
- Duc, T.N., Minh, C.T., et al. (2020). “Convolutional Neural Networks for Continuous QoE Prediction in Video Streaming Services”. *IEEE Access*, 8:116268–116278. DOI: [10.1109/ACCESS.2020.3004125](https://doi.org/10.1109/ACCESS.2020.3004125).
- Ehrig, C., Sonnleitner, B., et al. (2024). *The impact of data set similarity and diversity on transfer learning success in time series forecasting*. Version Number: 2. DOI: [10.48550/ARXIV.2404.06198](https://doi.org/10.48550/ARXIV.2404.06198).
- Engelbrechtsen, S. and Bohlin, J. (2019). “Statistical predictions with glmnet”. *Clinical Epigenetics*, 11(1):123. DOI: [10.1186/s13148-019-0730-1](https://doi.org/10.1186/s13148-019-0730-1).
- Ensarioğlu, K., İnkaya, T., et al. (2023). “Remaining Useful Life Estimation of Turbofan Engines with Deep Learning Using Change-Point Detection Based Labeling and Feature Engineering”. *Applied Sciences*, 13(21):11893. DOI: [10.3390/app132111893](https://doi.org/10.3390/app132111893).
- Fahimifar, S., Mousavi, K., et al. (2023). “Identification of the most important external features of highly cited scholarly papers through 3 (i.e., Ridge, Lasso, and Boruta)

- feature selection data mining methods”. *Quality & Quantity*, 57(4):3685–3712. DOI: [10.1007/s11135-022-01480-z](https://doi.org/10.1007/s11135-022-01480-z).
- Fan, B., Li, B., et al. (2017). “Modeling and experimental investigations on the relationship between wear debris concentration and wear rate in lubrication systems”. *Tribology International*, 109:114–123. DOI: [10.1016/j.triboint.2016.12.015](https://doi.org/10.1016/j.triboint.2016.12.015).
- Fan, Y., Nowaczyk, S., et al. (2020). “Transfer learning for remaining useful life prediction based on consensus self-organizing models”. *Reliability Engineering & System Safety*, 203:107098. DOI: [10.1016/j.ress.2020.107098](https://doi.org/10.1016/j.ress.2020.107098).
- Fathi, K., Van De Venn, H.W., et al. (2021). “Predictive Maintenance: An Autoencoder Anomaly-Based Approach for a 3 DoF Delta Robot”. *Sensors*, 21(21):6979. DOI: [10.3390/s21216979](https://doi.org/10.3390/s21216979).
- Foroutan, P. and Lahmiri, S. (2024). “Deep learning-based spatial-temporal graph neural networks for price movement classification in crude oil and precious metal markets”. *Machine Learning with Applications*, 16:100552. DOI: [10.1016/j.mlwa.2024.100552](https://doi.org/10.1016/j.mlwa.2024.100552).
- Frederick, D., DeCastro, J., et al. (2013). *User’s Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS)*. Technical Memorandum NASA/TM-2007-215026. Cleveland, OH, United States: NASA Technical Reports Server.
- Fu, J., Chu, J., et al. (2019). “Condition Monitoring of Wind Turbine Gearbox Bearing Based on Deep Learning Model”. *IEEE Access*, 7:57078–57087. DOI: [10.1109/ACCESS.2019.2912621](https://doi.org/10.1109/ACCESS.2019.2912621).
- Ganin, Y., Ustinova, E., et al. (2016). “Domain-adversarial training of neural networks”. *Journal of Machine Learning Research*, 17(59):1–35.
- Gao, H., Li, Y., et al. (2023). “Dual Channel Feature Attention-Based Approach for RUL Prediction Considering the Spatiotemporal Difference of Multisensor Data”. *IEEE Sensors Journal*, 23(8):8514–8525. DOI: [10.1109/JSEN.2023.3246595](https://doi.org/10.1109/JSEN.2023.3246595).
- Gianola, D., Okut, H., et al. (2011). “Predicting complex quantitative traits with Bayesian neural networks: a case study with Jersey cows and wheat”. *BMC Genetics*, 12(1):87. DOI: [10.1186/1471-2156-12-87](https://doi.org/10.1186/1471-2156-12-87).
- Goay, C.H., Ahmad, N.S., et al. (2023). “Temporal convolutional networks for transient simulation of high-speed channels”. *Alexandria Engineering Journal*, 74:643–663. DOI: [10.1016/j.aej.2023.05.059](https://doi.org/10.1016/j.aej.2023.05.059).
- Gomaa, A.H. (2025). “RCM 4.0: A Novel Digital Framework for Reliability-Centered Maintenance in Smart Industrial Systems”. *International Journal of Emerging Science and Engineering*, 13(5):32–43. DOI: [10.35940/ijese.E2595.13050425](https://doi.org/10.35940/ijese.E2595.13050425).
- Gong, R., Li, J., et al. (2022). “Remaining Useful Life Prediction Based on Multisensor Fusion and Attention TCN-BiGRU Model”. *IEEE Sensors Journal*, 22(21):21101–21110. DOI: [10.1109/JSEN.2022.3208753](https://doi.org/10.1109/JSEN.2022.3208753).
- Greff, K., Srivastava, R.K., et al. (2017). “LSTM: A Search Space Odyssey”. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232. DOI: [10.1109/TNNLS.2016.2582924](https://doi.org/10.1109/TNNLS.2016.2582924).

- Gretton, A., Borgwardt, K., et al. (2012). “A Kernel Two-Sample Test”. *Journal of Machine Learning Research*, 13:723–773.
- Han, J., Kamber, M., et al. (2012). “Classification”. In: *Data Mining Concepts and Techniques*. 3rd ed. 225 Wyman Street, Waltham, MA 02451, USA: Elsevier:pp. 330–331. URL: <https://linkinghub.elsevier.com/retrieve/pii/B9780123814791000083> (visited on 03/31/2026).
- Hassanat, A.B., Alqaralleh, M.K., et al. (2024). “A Novel Outlier-Robust Accuracy Measure for Machine Learning Regression Using a Non-Convex Distance Metric”. *Mathematics*, 12(22):3623. DOI: [10.3390/math12223623](https://doi.org/10.3390/math12223623).
- Hatipoğlu, A. and Yilmaz, E. (2025). “A Matrix-Statistics-Aware Attention Mechanism for Robust RUL Estimation in Aero-Engines”. *Applied Sciences*, 16(1):169. DOI: [10.3390/app16010169](https://doi.org/10.3390/app16010169).
- Hattula, E., Zhu, L., et al. (2023). “Advantages of Using Transfer Learning Technology with a Quantative Measurement”. *Remote Sensing*, 15(17):4278. DOI: [10.3390/rs15174278](https://doi.org/10.3390/rs15174278).
- Hazra, A. and Gogtay, N. (2017). “Biostatistics series module 10: Brief overview of multivariate methods”. *Indian Journal of Dermatology*, 62(4):358. DOI: [10.4103/ijd.IJD_296_17](https://doi.org/10.4103/ijd.IJD_296_17).
- He, J., Wu, C., et al. (2024). “Remaining Useful Life Prediction and Uncertainty Quantification for Bearings Based on Cascaded Multiscale Convolutional Neural Network”. *IEEE Transactions on Instrumentation and Measurement*, 73:1–13. DOI: [10.1109/TIM.2023.3347782](https://doi.org/10.1109/TIM.2023.3347782).
- Hong, C.W., Lee, C., et al. (2020). “Remaining Useful Life Prognosis for Turbofan Engine Using Explainable Deep Neural Networks with Dimensionality Reduction”. *Sensors*, 20(22):6626. DOI: [10.3390/s20226626](https://doi.org/10.3390/s20226626).
- Howard, J. and Ruder, S. (2018). “Universal Language Model Fine-tuning for Text Classification”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics:pp. 328–339. DOI: [10.18653/v1/P18-1031](https://doi.org/10.18653/v1/P18-1031).
- Hsu, C.-Y., Lu, Y.-W., et al. (2022). “Temporal Convolution-Based Long-Short Term Memory Network With Attention Mechanism for Remaining Useful Life Prediction”. *IEEE Transactions on Semiconductor Manufacturing*, 35(2):220–228. DOI: [10.1109/TSM.2022.3164578](https://doi.org/10.1109/TSM.2022.3164578).
- Hu, K., Cheng, Y., et al. (2023). “Deep Bidirectional Recurrent Neural Networks Ensemble for Remaining Useful Life Prediction of Aircraft Engine”. *IEEE Transactions on Cybernetics*, 53(4):2531–2543. DOI: [10.1109/TCYB.2021.3124838](https://doi.org/10.1109/TCYB.2021.3124838).
- Hu, X., Saremi, A., et al. (2016). “Machine learning to predict rapid progression of carotid atherosclerosis in patients with impaired glucose tolerance”. *EURASIP Journal on Bioinformatics and Systems Biology*, 2016(1):14. DOI: [10.1186/s13637-016-0049-6](https://doi.org/10.1186/s13637-016-0049-6).

- Hu, X., Xie, J., et al. (2025). “Predicting hydrocarbon reservoir quality in deepwater sedimentary systems using sequential deep learning techniques”. *Geomechanics and Geophysics for Geo-Energy and Geo-Resources*, 11(1):96. DOI: [10.1007/s40948-025-01030-5](https://doi.org/10.1007/s40948-025-01030-5).
- Hyndman, R.J. and Koehler, A.B. (2006). “Another look at measures of forecast accuracy”. *International Journal of Forecasting*, 22(4):679–688. DOI: [10.1016/j.ijforecast.2006.03.001](https://doi.org/10.1016/j.ijforecast.2006.03.001).
- Ilemobayo, J., Durodola, O., et al. (2024). “Hyperparameter Tuning in Machine Learning: A Comprehensive Review”. *Journal of Engineering Research and Reports*, 26(6):388–395. DOI: [10.9734/jerr/2024/v26i61188](https://doi.org/10.9734/jerr/2024/v26i61188).
- Iman, M., Arabnia, H.R., et al. (2023). “A Review of Deep Transfer Learning and Recent Advancements”. *Technologies*, 11(2):40. DOI: [10.3390/technologies11020040](https://doi.org/10.3390/technologies11020040).
- İnce, K. and Genc, Y. (2023). “Joint autoencoder-regressor deep neural network for remaining useful life prediction”. *Engineering Science and Technology, an International Journal*, 41:101409. DOI: [10.1016/j.jestch.2023.101409](https://doi.org/10.1016/j.jestch.2023.101409).
- Jafari, S. and Byun, Y.-C. (2023). “A CNN-GRU Approach to the Accurate Prediction of Batteries’ Remaining Useful Life from Charging Profiles”. *Computers*, 12(11):219. DOI: [10.3390/computers12110219](https://doi.org/10.3390/computers12110219).
- Jardillier, R., Koca, D., et al. (2022). “Prognosis of lasso-like penalized Cox models with tumor profiling improves prediction over clinical data alone and benefits from bi-dimensional pre-screening”. *BMC Cancer*, 22(1):1045. DOI: [10.1186/s12885-022-10117-1](https://doi.org/10.1186/s12885-022-10117-1).
- Jiang, C., Huang, Z., et al. (2024). “Network properties determine neural network performance”. *Nature Communications*, 15(1):5718. DOI: [10.1038/s41467-024-48069-8](https://doi.org/10.1038/s41467-024-48069-8).
- Jin, R., Chen, Z., et al. (2022). “Bi-LSTM-Based Two-Stream Network for Machine Remaining Useful Life Prediction”. *IEEE Transactions on Instrumentation and Measurement*, 71:1–10. DOI: [10.1109/TIM.2022.3167778](https://doi.org/10.1109/TIM.2022.3167778).
- Joas, M.J., Jurenaite, N., et al. (2025). “AUTOENCODIX: a generalized and versatile framework to train and evaluate autoencoders for biological representation learning and beyond”. *Nature Computational Science*, 6(1):96–108. DOI: [10.1038/s43588-025-00916-4](https://doi.org/10.1038/s43588-025-00916-4).
- Kamariotis, A., Tatsis, K., et al. (2024). “A metric for assessing and optimizing data-driven prognostic algorithms for predictive maintenance”. *Reliability Engineering & System Safety*, 242:109723. DOI: [10.1016/j.ress.2023.109723](https://doi.org/10.1016/j.ress.2023.109723).
- Kang, Z., Catal, C., et al. (2021). “Remaining Useful Life (RUL) Prediction of Equipment in Production Lines Using Artificial Neural Networks”. *Sensors*, 21(3):932. DOI: [10.3390/s21030932](https://doi.org/10.3390/s21030932).
- Kans, M. (2009). “The advancement of maintenance information technology: A literature review”. *Journal of Quality in Maintenance Engineering*, 15(1):5–16. DOI: [10.1108/13552510910943859](https://doi.org/10.1108/13552510910943859).

- Kapoor, S., Cantrell, E.M., et al. (2024). “REFORMS: Consensus-based Recommendations for Machine-learning-based Science”. *Science Advances*, 10(18):eadk3452. DOI: [10.1126/sciadv.adk3452](https://doi.org/10.1126/sciadv.adk3452).
- Ke, G., Meng, Q., et al. (2017). “LightGBM: A Highly Efficient Gradient Boosting Decision Tree”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U.V. Luxburg, et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.
- Khan, F.I. and Haddara, M.M. (2003). “Risk-based maintenance (RBM): a quantitative approach for maintenance/inspection scheduling and planning”. *Journal of Loss Prevention in the Process Industries*, 16(6):561–573. DOI: [10.1016/j.jlp.2003.08.011](https://doi.org/10.1016/j.jlp.2003.08.011).
- Khelif, R., Chebel-Morello, B., et al. (2017). “Direct Remaining Useful Life Estimation Based on Support Vector Regression”. *IEEE Transactions on Industrial Electronics*, 64(3):2276–2285. DOI: [10.1109/TIE.2016.2623260](https://doi.org/10.1109/TIE.2016.2623260).
- Kim, G., Choi, J.G., et al. (2024). “Using transformer and a reweighting technique to develop a remaining useful life estimation method for turbofan engines”. *Engineering Applications of Artificial Intelligence*, 133:108475. DOI: [10.1016/j.engappai.2024.108475](https://doi.org/10.1016/j.engappai.2024.108475).
- Kim, M.-J. and Lee, K.-W. (2026). “Deep Learning-Based Prediction of Remaining Useful Life of Aircraft Engines Using Non-Linear Labels”. *Electronics Letters*, 62(1):e70613. DOI: [10.1049/e112.70613](https://doi.org/10.1049/e112.70613).
- Kingma, D.P. and Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. Version Number: 9. DOI: [10.48550/ARXIV.1412.6980](https://doi.org/10.48550/ARXIV.1412.6980).
- Kollintza-Kyriakoulia, F., Maragoudakis, M., et al. (2018). “Measuring the Impact of Financial News and Social Media on Stock Market Modeling Using Time Series Mining Techniques”. *Algorithms*, 11(11):181. DOI: [10.3390/a11110181](https://doi.org/10.3390/a11110181).
- Krčėk, M., Wu, L., et al. (2024). “Shift-Invariance Robustness of Convolutional Neural Networks in Side-Channel Analysis”. *Mathematics*, 12(20):3279. DOI: [10.3390/math12203279](https://doi.org/10.3390/math12203279).
- Krizhevsky, A., Sutskever, I., et al. (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, et al. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.
- Kuiziniėnė, D., Savickas, P., et al. (2024). “A comparative study of feature selection and feature extraction methods for financial distress identification”. *PeerJ Computer Science*, 10:e1956. DOI: [10.7717/peerj-cs.1956](https://doi.org/10.7717/peerj-cs.1956).
- Kumar Sharma, D., Brahmachari, S., et al. (2022). “Data driven predictive maintenance applications for industrial systems with temporal convolutional networks”. *Computers & Industrial Engineering*, 169:108213. DOI: [10.1016/j.cie.2022.108213](https://doi.org/10.1016/j.cie.2022.108213).

- Kwak, S.G. (2025). “LASSO Regression Analysis: Applications in Dyslipidemia and Cardiovascular Disease Research”. *Journal of Lipid and Atherosclerosis*, 14(3):289. DOI: [10.12997/jla.2025.14.3.289](https://doi.org/10.12997/jla.2025.14.3.289).
- LeCun, Y., Bengio, Y., et al. (2015). “Deep learning”. *Nature*, 521(7553):436–444. DOI: [10.1038/nature14539](https://doi.org/10.1038/nature14539).
- Lemeš, L. and Akagic, A. (2023). “Prediction of Real Estate Market Prices with Regression Algorithms”. In: *Advanced Technologies, Systems, and Applications VII*. Ed. by N. Ademović, E. Mujčić, et al. Vol. 539. Series Title: Lecture Notes in Networks and Systems. Cham: Springer International Publishing:pp. 401–411. DOI: [10.1007/978-3-031-17697-5_32](https://doi.org/10.1007/978-3-031-17697-5_32).
- Li, B. and Li, X. (2026). “Best Practices for Developing Linear Models With Multiple Explanatory Variables”. *Advanced Genetics*, 7(1):e00024. DOI: [10.1002/ggn2.202500024](https://doi.org/10.1002/ggn2.202500024).
- Li, H., Huang, J., et al. (2019). “Bearing Fault Diagnosis with a Feature Fusion Method Based on an Ensemble Convolutional Neural Network and Deep Neural Network”. *Sensors*, 19(9):2034. DOI: [10.3390/s19092034](https://doi.org/10.3390/s19092034).
- Li, X., Wang, L., et al. (2024). “A method for predicting remaining useful life using enhanced Savitzky–Golay filter and improved deep learning framework”. *Scientific Reports*, 14(1):23983. DOI: [10.1038/s41598-024-74989-y](https://doi.org/10.1038/s41598-024-74989-y).
- Li, Y., Wang, X., et al. (2025). “Federated Domain Generalization: A Survey”. *Proceedings of the IEEE*, 113(4):370–410. DOI: [10.1109/JPROC.2025.3596173](https://doi.org/10.1109/JPROC.2025.3596173).
- Liaw, A. and Wiener, M. (2002). “Classification and regression by randomForest”. *R News*, 2(3):18–22.
- Liberati, A., Altman, D.G., et al. (2009). “The PRISMA statement for reporting systematic reviews and meta-analyses of studies that evaluate health care interventions: explanation and elaboration”. *Journal of Clinical Epidemiology*, 62(10):e1–e34. DOI: [10.1016/j.jclinepi.2009.06.006](https://doi.org/10.1016/j.jclinepi.2009.06.006).
- Lin, T., Wang, Y., et al. (2022). “A survey of transformers”. *AI Open*, 3:111–132. DOI: [10.1016/j.aiopen.2022.10.001](https://doi.org/10.1016/j.aiopen.2022.10.001).
- Lipton, Z.C., Berkowitz, J., et al. (2015). *A Critical Review of Recurrent Neural Networks for Sequence Learning*. Version Number: 4. DOI: [10.48550/ARXIV.1506.00019](https://doi.org/10.48550/ARXIV.1506.00019).
- Liu, C.-L., Xiao, B., et al. (2025). “Self-supervised learning for remaining useful life prediction using simple triplet networks”. *Advanced Engineering Informatics*, 64:103038. DOI: [10.1016/j.aei.2024.103038](https://doi.org/10.1016/j.aei.2024.103038).
- Liu, C., Zhang, L., et al. (2021). “Dual Attention-Based Temporal Convolutional Network for Fault Prognosis Under Time-Varying Operating Conditions”. *IEEE Transactions on Instrumentation and Measurement*, 70:1–10. DOI: [10.1109/TIM.2021.3053992](https://doi.org/10.1109/TIM.2021.3053992).
- Liu, Y., Pan, D., et al. (2024). “Structural-Guided Interaction and Attention-Enhancing Compensation Network for Machine Remaining Useful Life Prediction”. *IEEE Transactions on Instrumentation and Measurement*, 73:1–11. DOI: [10.1109/TIM.2024.3375984](https://doi.org/10.1109/TIM.2024.3375984).

- Lopes, I., Senra, P., et al. (2016). “Requirements Specification of a Computerized Maintenance Management System – A Case Study”. *Procedia CIRP*, 52:268–273. DOI: [10.1016/j.procir.2016.07.047](https://doi.org/10.1016/j.procir.2016.07.047).
- Lopez, E., Etxebarria-Elezgarai, J., et al. (2023). “The importance of choosing a proper validation strategy in predictive models. A tutorial with real examples”. *Analytica Chimica Acta*, 1275:341532. DOI: [10.1016/j.aca.2023.341532](https://doi.org/10.1016/j.aca.2023.341532).
- Lu, B.-L., Liu, Z.-H., et al. (2021). “A Deep Adversarial Learning Prognostics Model for Remaining Useful Life Prediction of Rolling Bearing”. *IEEE Transactions on Artificial Intelligence*, 2(4):329–340. DOI: [10.1109/TAI.2021.3097311](https://doi.org/10.1109/TAI.2021.3097311).
- Ma, J., Ding, Y., et al. (2019). “Spatiotemporal Prediction of PM2.5 Concentrations at Different Time Granularities Using IDW-BLSTM”. *IEEE Access*, 7:107897–107907. DOI: [10.1109/ACCESS.2019.2932445](https://doi.org/10.1109/ACCESS.2019.2932445).
- Mao, P., Lin, Y., et al. (2022). “Remaining Useful Life Estimation of Aircraft Engines Using Differentiable Architecture Search”. *Mathematics*, 10(3):352. DOI: [10.3390/math10030352](https://doi.org/10.3390/math10030352).
- Maré, T. (2026). *RUL-Prediction-CMAPSS*. URL: <https://github.com/T-Mare/RUL-Prediction-CMAPSS.git> (visited on 09/08/2026).
- May, S., Hartmann, S., et al. (2022). *Combined Pruning for Nested Cross-Validation to Accelerate Automated Hyperparameter Optimization for Embedded Feature Selection in High-Dimensional Data with Very Small Sample Sizes*. Version Number: 2. DOI: [10.48550/ARXIV.2202.00598](https://doi.org/10.48550/ARXIV.2202.00598).
- Mehdary, A., Chehri, A., et al. (2024). “Hyperparameter Optimization with Genetic Algorithms and XGBoost: A Step Forward in Smart Grid Fraud Detection”. *Sensors*, 24(4):1230. DOI: [10.3390/s24041230](https://doi.org/10.3390/s24041230).
- Meng, J., Yue, M., et al. (2023). “A Degradation Empirical-Model-Free Battery End-of-Life Prediction Framework Based on Gaussian Process Regression and Kalman Filter”. *IEEE Transactions on Transportation Electrification*, 9(4):4898–4908. DOI: [10.1109/TTE.2022.3209629](https://doi.org/10.1109/TTE.2022.3209629).
- Michelucci, U. (2022). *An Introduction to Autoencoders*. Version Number: 1. DOI: [10.48550/ARXIV.2201.03898](https://doi.org/10.48550/ARXIV.2201.03898).
- Mo, J., Wang, R., et al. (2023). “A hybrid temporal convolutional network and Prophet model for power load forecasting”. *Complex & Intelligent Systems*, 9(4):4249–4261. DOI: [10.1007/s40747-022-00952-x](https://doi.org/10.1007/s40747-022-00952-x).
- Mojtahedi, F.F., Yousefpour, N., et al. (2025). “Deep Learning for Time Series Forecasting: Review and Applications in Geotechnics and Geosciences”. *Archives of Computational Methods in Engineering*, 32(6):3415–3445. DOI: [10.1007/s11831-025-10244-5](https://doi.org/10.1007/s11831-025-10244-5).
- Moleǳa, M., Małysiak-Mrozek, B., et al. (2023). “From Corrective to Predictive Maintenance—A Review of Maintenance Approaches for the Power Industry”. *Sensors*, 23(13):5970. DOI: [10.3390/s23135970](https://doi.org/10.3390/s23135970).

- Morozova, O., Levina, O., et al. (2015). “Comparison of subset selection methods in linear regression in the context of health-related quality of life and substance abuse in Russia”. *BMC Medical Research Methodology*, 15(1):71. DOI: [10.1186/s12874-015-0066-2](https://doi.org/10.1186/s12874-015-0066-2).
- Mushagalusa, C.A., Fandohan, A.B., et al. (2024). “Random forest and spatial cross-validation performance in predicting species abundance distributions”. *Environmental Systems Research*, 13(1):23. DOI: [10.1186/s40068-024-00352-9](https://doi.org/10.1186/s40068-024-00352-9).
- Mutlag, W.K., Ali, S.K., et al. (2020). “Feature Extraction Methods: A Review”. *Journal of Physics: Conference Series*, 1591(1):012028. DOI: [10.1088/1742-6596/1591/1/012028](https://doi.org/10.1088/1742-6596/1591/1/012028).
- Na, L., Cai, B., et al. (2025). “A heterogeneous transfer learning method for fault prediction of railway track circuit”. *Engineering Applications of Artificial Intelligence*, 140:109740. DOI: [10.1016/j.engappai.2024.109740](https://doi.org/10.1016/j.engappai.2024.109740).
- NASA (2008). *PHM 2008 Challenge*. URL: <https://data.nasa.gov/dataset/phm-2008-challenge> (visited on 10/08/2026).
- NASA (2025). *Prognostics Center of Excellence Data Set Repository*. URL: https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/?utm_source=chatgpt.com (visited on 02/02/2026).
- Nectoux, P., Gouriveau, R., et al. (2012). “PRONOSTIA: An experimental platform for bearings accelerated degradation tests”. In: *IEEE International Conference on Prognostics and Health Management*. Besançon, France:pp. 1–8.
- Nguyen, C.T., Van Huynh, N., et al. (2022). “Transfer Learning for Wireless Networks: A Comprehensive Survey”. *Proceedings of the IEEE*, 110(8):1073–1115. DOI: [10.1109/JPROC.2022.3175942](https://doi.org/10.1109/JPROC.2022.3175942).
- Niu, Z., Zhong, G., et al. (2023). “Recurrent attention unit: A new gated recurrent unit for long-term memory of important parts in sequential data”. *Neurocomputing*, 517:1–9. DOI: [10.1016/j.neucom.2022.10.050](https://doi.org/10.1016/j.neucom.2022.10.050).
- Nowlan, F.S. and Heap, H.F. (Dec. 1978). *Reliability-Centered Maintenance*: tech. rep. Fort Belvoir, VA: Defense Technical Information Center. DOI: [10.21236/ADA066579](https://doi.org/10.21236/ADA066579).
- Nunes, P., Santos, J., et al. (2023). “Challenges in predictive maintenance – A review”. *CIRP Journal of Manufacturing Science and Technology*, 40:53–67. DOI: [10.1016/j.cirpj.2022.11.004](https://doi.org/10.1016/j.cirpj.2022.11.004).
- Nwankpa, C.E. (2020). “Advances in Optimisation Algorithms and Techniques for Deep Learning”. *Advances in Science, Technology and Engineering Systems Journal*, 5(5):563–577. DOI: [10.25046/aj050570](https://doi.org/10.25046/aj050570).
- Ojeda, J.C.O., De Moraes, J.G.B., et al. (2025). “Application of a Predictive Model to Reduce Unplanned Downtime in Automotive Industry Production Processes: A Sustainability Perspective”. *Sustainability*, 17(9):3926. DOI: [10.3390/su17093926](https://doi.org/10.3390/su17093926).
- Okuno, T., Takeda, A., et al. (2021). “On lp-hyperparameter learning via bilevel nonsmooth optimization”. *Journal of Machine Learning Research*, 22(245):1–47.
- Ong, K.S.H., Wang, W., et al. (2022). “Deep-Reinforcement-Learning-Based Predictive Maintenance Model for Effective Resource Management in Industrial IoT”. *IEEE Internet of Things Journal*, 9(7):5173–5188. DOI: [10.1109/JIOT.2021.3109955](https://doi.org/10.1109/JIOT.2021.3109955).

- Otović, E., Njirjak, M., et al. (2022). “Intra-domain and cross-domain transfer learning for time series data—How transferable are the features?” *Knowledge-Based Systems*, 239:107976. DOI: [10.1016/j.knosys.2021.107976](https://doi.org/10.1016/j.knosys.2021.107976).
- Özden, M.G., Liu, X., et al. (2024). “Predictive modelling of laser powder bed fusion of Fe-based nanocrystalline alloys based on experimental data using multiple linear regression analysis”. *Heliyon*, 10(15):e35047. DOI: [10.1016/j.heliyon.2024.e35047](https://doi.org/10.1016/j.heliyon.2024.e35047).
- Packer, C., Gao, K., et al. (2018). *Assessing Generalization in Deep Reinforcement Learning*. Version Number: 2. DOI: [10.48550/ARXIV.1810.12282](https://doi.org/10.48550/ARXIV.1810.12282).
- Page, M.J., McKenzie, J.E., et al. (2021). “The PRISMA 2020 statement: An updated guideline for reporting systematic reviews”. *International Journal of Surgery*, 88:105906. DOI: [10.1016/j.ijssu.2021.105906](https://doi.org/10.1016/j.ijssu.2021.105906).
- Pan, S.J. and Yang, Q. (2010). “A Survey on Transfer Learning”. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359. DOI: [10.1109/TKDE.2009.191](https://doi.org/10.1109/TKDE.2009.191).
- Pei, H. (2019). “Review of Machine Learning Based Remaining Useful Life Prediction Methods for Equipment”. *Journal of Mechanical Engineering*, 55(8):1. DOI: [10.3901/JME.2019.08.001](https://doi.org/10.3901/JME.2019.08.001).
- Peng, L., Liang, H., et al. (2021). *Rethinking Transfer Learning for Medical Image Classification*. Version Number: 8. DOI: [10.48550/ARXIV.2106.05152](https://doi.org/10.48550/ARXIV.2106.05152).
- Pfob, A., Lu, S.-C., et al. (2022). “Machine learning in medicine: a practical introduction to techniques for data pre-processing, hyperparameter tuning, and model comparison”. *BMC Medical Research Methodology*, 22(1):282. DOI: [10.1186/s12874-022-01758-8](https://doi.org/10.1186/s12874-022-01758-8).
- Poór, P. (2019). “Historical Overview of Maintenance Management Strategies: Development from Breakdown Maintenance to Predictive Maintenance in Accordance with Four Industrial Revolutions”. In: *International Conference on Industrial Engineering and Operations Management*. Pilsen, Czech Republic:pp. 495–504.
- Prapas, I., Derakhshan, B., et al. (2021). “Continuous Training and Deployment of Deep Learning Models”. *Datenbank-Spektrum*, 21(3):203–212. DOI: [10.1007/s13222-021-00386-8](https://doi.org/10.1007/s13222-021-00386-8).
- Prognostics and Health Management Society (2010). *2010 PHM Society Conference Data Challenge*. URL: https://phmsociety.org/phm_competition/2010-phm-society-conference-data-challenge/ (visited on 09/10/2024).
- Prognostics and Health Management Society (2018). *PHM DATA CHALLENGE 2018*. URL: <https://phmsociety.org/conference/annual-conference-of-the-phm-society/annual-conference-of-the-prognostics-and-health-management-society-2018-b/phm-data-challenge-6/> (visited on 09/08/2026).
- Prognostics and Health Management Society (2020). *phme20-public*. URL: <https://github.com/zakkum42/phme20-public?tab=readme-ov-file> (visited on 09/10/2024).
- Rachidi, A., Talbi, A., et al. (2013). “The new forms of industrial maintenance: which impact on the performance of the industrial companies? (Case study)”. *International Journal of Engineering and Advanced Technology*, 2:500–507.

- Radford, A., Narasimhan, K., et al. (2018). “Improving language understanding by generative pre-training”.
- Ragab, M., Chen, Z., et al. (2020). *Attention Sequence to Sequence Model for Machine Remaining Useful Life Prediction*. DOI: [10.48550/ARXIV.2007.09868](https://doi.org/10.48550/ARXIV.2007.09868).
- Ramasso, E. and Saxena, A. (2014). “Review and Analysis of Algorithmic Approaches Developed for Prognostics on CMAPSS Dataset”. In: *Annual Conference of the Prognostics and Health Management Society 2014*. Fort Worth, TX, USA., United States.
- Ratul, I.J., Zhou, Y., et al. (2025). “Accelerating Deep Learning Inference: A Comparative Analysis of Modern Acceleration Frameworks”. *Electronics*, 14(15):2977. DOI: [10.3390/electronics14152977](https://doi.org/10.3390/electronics14152977).
- Riaz, A., Nabeel, M., et al. (2020). “SBAG: A Hybrid Deep Learning Model for Large Scale Traffic Speed Prediction”. *International Journal of Advanced Computer Science and Applications*, 11(1). DOI: [10.14569/IJACSA.2020.0110135](https://doi.org/10.14569/IJACSA.2020.0110135).
- Rivas, A., Delipei, G.K., et al. (2022). “Predictions of component Remaining Useful Lifetime Using Bayesian Neural Network”. *Progress in Nuclear Energy*, 146:104143. DOI: [10.1016/j.pnucene.2022.104143](https://doi.org/10.1016/j.pnucene.2022.104143).
- Rokach, L. (2016). “Decision forest: Twenty years of research”. *Information Fusion*, 27:111–125. DOI: [10.1016/j.inffus.2015.06.005](https://doi.org/10.1016/j.inffus.2015.06.005).
- Rouzbeh, A., Bhandari, Jyoti, et al. (2016). “Developing a quantitative risk-based methodology for maintenance scheduling using bayesian network”. *Chemical Engineering Transactions*, 48:235–240. DOI: [10.3303/CET1648040](https://doi.org/10.3303/CET1648040).
- Rubin, J., Mariani, L., et al. (2022). “Ridge Regression for Functional Form Identification of Continuous Predictors of Clinical Outcomes in Glomerular Disease”. *Glomerular Diseases*, 3(1):47–55. DOI: [10.1159/000528847](https://doi.org/10.1159/000528847).
- Ruye, W. (2025). *Introduction to Machine Learning*. Cambridge university press.
- Sagar, P., Gupta, P., et al. (2021). “A forecasting method with efficient selection of variables in multivariate data sets”. *International Journal of Information Technology*, 13(3):1039–1046. DOI: [10.1007/s41870-021-00619-9](https://doi.org/10.1007/s41870-021-00619-9).
- Saha, B. and Goebel, K. (2007). *Battery Data Set*. NASA Ames Research Center, Moffett Field, CA. URL: <http://ti.arc.nasa.gov/project/prognostic-data-repository>.
- Al-Salih, O.G., Guangjian, D., et al. (2026). “Using extreme gradient boosting with Optuna hyperparameter tuning for efficient lost circulation prediction”. *Energy Geoscience*, 7(2):100540. DOI: [10.1016/j.engeos.2026.100540](https://doi.org/10.1016/j.engeos.2026.100540).
- Samanta, I.S., Panda, S., et al. (2023). “A Comprehensive Review of Deep-Learning Applications to Power Quality Analysis”. *Energies*, 16(11):4406. DOI: [10.3390/en16114406](https://doi.org/10.3390/en16114406).
- Saxena, A., Goebel, K., et al. (2008a). “CMAPSS Jet Engine Simulated Data”. In: *Damage Propagation Modeling for Aircraft Engine Run-to-Failure Simulation*. Denver CO: Prognostics Center of Excellence.
- Saxena, A., Goebel, K., et al. (Oct. 2008b). “Damage propagation modeling for aircraft engine run-to-failure simulation”. In: *2008 International Conference on Prognostics and*

- Health Management*. Denver, CO, USA: IEEE;pp. 1–9. DOI: [10.1109/PHM.2008.4711414](https://doi.org/10.1109/PHM.2008.4711414).
- Schapire, R.E. (2003). “The Boosting Approach to Machine Learning: An Overview”. In: *Nonlinear Estimation and Classification*. Ed. by P. Bickel, P. Diggle, et al. Vol. 171. Series Title: Lecture Notes in Statistics. New York, NY: Springer New York;pp. 149–171. DOI: [10.1007/978-0-387-21579-2_9](https://doi.org/10.1007/978-0-387-21579-2_9).
- scikit-learn (2025). *sklearn.preprocessing.StandardScaler*. URL: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html> (visited on 02/22/2026).
- Shankar, L., Singh, C.D., et al. (2023). “Impact of implementation of CMMS for enhancing the performance of manufacturing industries”. *International Journal of System Assurance Engineering and Management*, 14(5):1599–1620. DOI: [10.1007/s13198-021-01480-6](https://doi.org/10.1007/s13198-021-01480-6).
- Sharma, K. (2025). *Uncertainty-Aware Deep Learning Framework for Remaining Useful Life Prediction in Turbofan Engines with Learned Aleatoric Uncertainty*. Version Number: 2. DOI: [10.48550/ARXIV.2511.19124](https://doi.org/10.48550/ARXIV.2511.19124).
- Shi, J., Gao, J., et al. (2023). “Adaptively Lightweight Spatiotemporal Information-Extraction-Operator-Based DL Method for Aero-Engine RUL Prediction”. *Sensors*, 23(13):6163. DOI: [10.3390/s23136163](https://doi.org/10.3390/s23136163).
- Si, X.-S., Wang, W., et al. (2011). “Remaining useful life estimation – A review on the statistical data driven approaches”. *European Journal of Operational Research*, 213(1):1–14. DOI: [10.1016/j.ejor.2010.11.018](https://doi.org/10.1016/j.ejor.2010.11.018).
- Siarni-Namini, S., Tavakoli, N., et al. (Dec. 2019). “The Performance of LSTM and BiLSTM in Forecasting Time Series”. In: *2019 IEEE International Conference on Big Data (Big Data)*. Los Angeles, CA, USA: IEEE;pp. 3285–3292. DOI: [10.1109/BigData47090.2019.9005997](https://doi.org/10.1109/BigData47090.2019.9005997).
- Sicilia, A., Zhao, X., et al. (2023). “Domain adversarial neural networks for domain generalization: when it works and how to improve”. *Machine Learning*, 112(7):2685–2721. DOI: [10.1007/s10994-023-06324-x](https://doi.org/10.1007/s10994-023-06324-x).
- Staudemeyer, R.C. and Morris, E.R. (2019). *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. Version Number: 1. DOI: [10.48550/ARXIV.1909.09586](https://doi.org/10.48550/ARXIV.1909.09586).
- Sun, C., Ma, M., et al. (2019). “Deep Transfer Learning Based on Sparse Autoencoder for Remaining Useful Life Prediction of Tool in Manufacturing”. *IEEE Transactions on Industrial Informatics*, 15(4):2416–2425. DOI: [10.1109/TII.2018.2881543](https://doi.org/10.1109/TII.2018.2881543).
- Tai, L., Zhang, J., et al. (2016). *A Survey of Deep Network Solutions for Learning Control in Robotics: From Reinforcement to Imitation*. Version Number: 4. DOI: [10.48550/ARXIV.1612.07139](https://doi.org/10.48550/ARXIV.1612.07139).
- Tang, D., Cao, J., et al. (2019). “Remaining useful life prediction for engineering systems under dynamic operational conditions: A semi-Markov decision process-based approach”. *Chinese Journal of Aeronautics*, 32(3):627–638. DOI: [10.1016/j.cja.2018.08.015](https://doi.org/10.1016/j.cja.2018.08.015).

- Tanwar, M. and Raghavan, N. (2020). “Lubricating Oil Remaining Useful Life Prediction Using Multi-Output Gaussian Process Regression”. *IEEE Access*, 8:128897–128907. DOI: [10.1109/ACCESS.2020.3008328](https://doi.org/10.1109/ACCESS.2020.3008328).
- TensorFlow (2024). *tf.keras.layers.SpatialDropout1D*. URL: https://www.tensorflow.org/api_docs/python/tf/keras/layers/SpatialDropout1D (visited on 05/01/2026).
- Tesfay, A. (2023). “Extraction of Global Features for enhancing Machine Learning Performance”. Dissertation. School of Electrical Engineering and Computer Science.
- Thakkar, U. and Chaoui, H. (2022). “Remaining Useful Life Prediction of an Aircraft Turbofan Engine Using Deep Layer Recurrent Neural Networks”. *Actuators*, 11(3):67. DOI: [10.3390/act11030067](https://doi.org/10.3390/act11030067).
- Tompson, J., Goroshin, R., et al. (2015). “Efficient Object Localization Using Convolutional Networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*:pp. 648–656.
- Toothman, M., Braun, B., et al. (2023). “Overcoming Challenges Associated with Developing Industrial Prognostics and Health Management Solutions”. *Sensors*, 23(8):4009. DOI: [10.3390/s23084009](https://doi.org/10.3390/s23084009).
- Tran, L.P., Le, H.D., et al. (2025). “Traditional or advanced machine learning approaches: Which one is better for housing price prediction and uncertainty risk reduction?” *Risk Governance and Control: Financial Markets and Institutions*, 15(1):27. DOI: [10.22495/rgcv15i1p3](https://doi.org/10.22495/rgcv15i1p3).
- Tranmer, M. and Elliot, M. (2008). “Multiple linear regression”. *The Cathie Marsh Centre for Census and Survey Research*, 5(5):1–5.
- Tschöpe, M. (2019). “Beyond SGD: Recent improvements of Gradient Descent Methods”. DOI: [10.13140/RG.2.2.34359.88484](https://doi.org/10.13140/RG.2.2.34359.88484).
- Ullah, S., Li, S., et al. (2023). “An Investigation of Exhaust Gas Temperature of Aircraft Engine Using LSTM”. *IEEE Access*, 11:5168–5177. DOI: [10.1109/ACCESS.2023.3235619](https://doi.org/10.1109/ACCESS.2023.3235619).
- Usmani, S. and Shamsi, J.A. (2021). “News sensitive stock market prediction: literature review and suggestions”. *PeerJ Computer Science*, 7:e490. DOI: [10.7717/peerj-cs.490](https://doi.org/10.7717/peerj-cs.490).
- Van Houdt, G., Mosquera, C., et al. (2020). “A review on the long short-term memory model”. *Artificial Intelligence Review*, 53(8):5929–5955. DOI: [10.1007/s10462-020-09838-1](https://doi.org/10.1007/s10462-020-09838-1).
- Vaswani, A., Shazeer, N., et al. (2017). *Attention Is All You Need*. Version Number: 7. DOI: [10.48550/ARXIV.1706.03762](https://doi.org/10.48550/ARXIV.1706.03762).
- Vyas, U.B. and Shah, V.A. (2022). “Optimisation based 3-dimensional polynomial regression to represent lithium-ion battery’s open circuit voltage as function of state of charge and temperature”. *Journal of Energy Storage*, 50:104656. DOI: [10.1016/j.est.2022.104656](https://doi.org/10.1016/j.est.2022.104656).
- Wang, L., Chen, Y., et al. (2024). “Predictive Maintenance Scheduling for Aircraft Engines Based on Remaining Useful Life Prediction”. *IEEE Internet of Things Journal*, 11(13):23020–23031. DOI: [10.1109/JIOT.2024.3376715](https://doi.org/10.1109/JIOT.2024.3376715).

- Wang, S., Nan, B., et al. (2011). “Random lasso”. *The Annals of Applied Statistics*, 5(1). DOI: [10.1214/10-AOAS377](https://doi.org/10.1214/10-AOAS377).
- Wang, X., Jin, Y., et al. (2023). “Recent Advances in Bayesian Optimization”. *ACM Computing Surveys*, 55(13s):1–36. DOI: [10.1145/3582078](https://doi.org/10.1145/3582078).
- Wang, Y., Zhao, Y., et al. (2020). “Remaining Useful Life Prediction using Deep Learning Approaches: A Review”. *Procedia Manufacturing*, 49:81–88. DOI: [10.1016/j.promfg.2020.06.015](https://doi.org/10.1016/j.promfg.2020.06.015).
- Wang, Z., Zihang, D., et al. (2019). “Characterizing and Avoiding Negative Transfer”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*:pp. 11293–11302.
- Watanabe, S. (2023). *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. Version Number: 4. DOI: [10.48550/ARXIV.2304.11127](https://doi.org/10.48550/ARXIV.2304.11127).
- Wei, Z., Gao, J., et al. (2011). “Incipient fault diagnosis of rolling element bearing based on wavelet packet transform and energy operator”. 10(3):81–90.
- Weiss, K., Khoshgoftaar, T.M., et al. (2016). “A survey of transfer learning”. *Journal of Big Data*, 3(1):9. DOI: [10.1186/s40537-016-0043-6](https://doi.org/10.1186/s40537-016-0043-6).
- Wen, Q., Zhou, T., et al. (2022). “Transformers in Time Series: A Survey”. Version Number: 5. DOI: [10.48550/ARXIV.2202.07125](https://doi.org/10.48550/ARXIV.2202.07125).
- Werbos, P. (2002). “Backpropagation through time: what it does and how to do it.” In: *Proceedings of the IEEE*. Vol. 78. 10:pp. 1550–1560.
- Wickern, M.Z. (2019). “Challenges and Reliability of Predictive Maintenance”. DOI: [10.13140/RG.2.2.35379.89129](https://doi.org/10.13140/RG.2.2.35379.89129).
- Wickert, J.A. and Lewis, K.E. (2017). *An introduction to mechanical engineering*. Fourth edition. Australia ; Boston, MA: Cengage Learning.
- Winkel, F. and Kleinstaub, D. (2022). *Phoenix Contact Relay (PCR) Dataset*. (Visited on 03/10/2024).
- Winkel, F., Wallscheid, O., et al. (2023). “Pseudolabeling Machine Learning Algorithm for Predictive Maintenance of Relays”. *IEEE Open Journal of the Industrial Electronics Society*, 4:463–475. DOI: [10.1109/OJIES.2023.3323870](https://doi.org/10.1109/OJIES.2023.3323870).
- Wu, J., Poloczek, M., et al. (2017). “Bayesian Optimization with Gradients”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U.V. Luxburg, et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/64a08e5f1e6c39faeb90108c430eb120-Paper.pdf (visited on 06/07/2025).
- Xiang, S., Li, P., et al. (2025). “Micro Transfer Learning Mechanism for Cross-Domain Equipment RUL Prediction”. *IEEE Transactions on Automation Science and Engineering*, 22:1460–1470. DOI: [10.1109/TASE.2024.3366288](https://doi.org/10.1109/TASE.2024.3366288).
- Yang, C. and Ma, X. (2022). “Improving Stability of Fine-Tuning Pretrained Language Models via Component-Wise Gradient Norm Clipping”. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United

- Arab Emirates: Association for Computational Linguistics:pp. 4854–4859. DOI: [10.18653/v1/2022.emnlp-main.322](https://doi.org/10.18653/v1/2022.emnlp-main.322).
- Yang, J., Yin, S., et al. (2020). “A Fault Diagnosis Method of Rotating Machinery Based on One-Dimensional, Self-Normalizing Convolutional Neural Networks”. *Sensors*, 20(14):3837. DOI: [10.3390/s20143837](https://doi.org/10.3390/s20143837).
- Yang, J., Peng, Y., et al. (2022). “Remaining Useful Life Prediction Method for Bearings Based on LSTM with Uncertainty Quantification”. *Sensors*, 22(12):4549. DOI: [10.3390/s22124549](https://doi.org/10.3390/s22124549).
- Yang, M. and Wang, J. (2022). “Adaptability of Financial Time Series Prediction Based on BiLSTM”. *Procedia Computer Science*, 199:18–25. DOI: [10.1016/j.procs.2022.01.003](https://doi.org/10.1016/j.procs.2022.01.003).
- Yang, X., Wang, Y., et al. (2019). “Concepts of Artificial Intelligence for Computer-Assisted Drug Discovery”. *Chemical Reviews*, 119(18):10520–10594. DOI: [10.1021/acs.chemrev.8b00728](https://doi.org/10.1021/acs.chemrev.8b00728).
- Yang, Y. and Wang, H. (2025). “Random Forest-Based Machine Failure Prediction: A Performance Comparison”. *Applied Sciences*, 15(16):8841. DOI: [10.3390/app15168841](https://doi.org/10.3390/app15168841).
- Yates, L.A., Aandahl, Z., et al. (2023). “Cross validation for model selection: A review with examples from ecology”. *Ecological Monographs*, 93(1):e1557. DOI: [10.1002/ecm.1557](https://doi.org/10.1002/ecm.1557).
- Ye, H., Zhang, Y., et al. (2024). “Light Recurrent Unit: Towards an Interpretable Recurrent Neural Network for Modeling Long-Range Dependency”. *Electronics*, 13(16):3204. DOI: [10.3390/electronics13163204](https://doi.org/10.3390/electronics13163204).
- Yosinski, J., Clune, J., et al. (2014). “How transferable are features in deep neural networks?” Version Number: 1. DOI: [10.48550/ARXIV.1411.1792](https://doi.org/10.48550/ARXIV.1411.1792).
- Yunita, A., Pratama, M.I., et al. (2025). “Performance analysis of neural network architectures for time series forecasting: A comparative study of RNN, LSTM, GRU, and hybrid models”. *MethodsX*, 15:103462. DOI: [10.1016/j.mex.2025.103462](https://doi.org/10.1016/j.mex.2025.103462).
- Zeng, Z., Kaur, R., et al. (2023). *Financial Time Series Forecasting using CNN and Transformer*. Version Number: 1. DOI: [10.48550/ARXIV.2304.04912](https://doi.org/10.48550/ARXIV.2304.04912).
- Zhang, A., Wang, H., et al. (2018). “Transfer Learning with Deep Recurrent Neural Networks for Remaining Useful Life Estimation”. *Applied Sciences*, 8(12):2416. DOI: [10.3390/app8122416](https://doi.org/10.3390/app8122416).
- Zhang, H., Zhang, Q., et al. (2020). “Attention-Based LSTM Network for Rotatory Machine Remaining Useful Life Prediction”. *IEEE Access*, 8:132188–132199. DOI: [10.1109/ACCESS.2020.3010066](https://doi.org/10.1109/ACCESS.2020.3010066).
- Zhang, H., Wei, X., et al. (2025). “Condition-based maintenance for multi-state systems with prognostic and deep reinforcement learning”. *Reliability Engineering & System Safety*, 255:110659. DOI: [10.1016/j.ress.2024.110659](https://doi.org/10.1016/j.ress.2024.110659).
- Zhang, W., Yang, D., et al. (2019). “Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey”. *IEEE Systems Journal*, 13(3):2213–2227. DOI: [10.1109/JSYST.2019.2905565](https://doi.org/10.1109/JSYST.2019.2905565).

- Zhang, Z., Song, W., et al. (2022). “Dual-Aspect Self-Attention Based on Transformer for Remaining Useful Life Prediction”. *IEEE Transactions on Instrumentation and Measurement*, 71:1–11. DOI: [10.1109/TIM.2022.3160561](https://doi.org/10.1109/TIM.2022.3160561).
- Zhao, C., Huang, X., et al. (2020). “A Double-Channel Hybrid Deep Neural Network Based on CNN and BiLSTM for Remaining Useful Life Prediction”. *Sensors*, 20(24):7109. DOI: [10.3390/s20247109](https://doi.org/10.3390/s20247109).
- Zhao, Q. and Zhang, X. (2024). “HMDM: A hybrid GM model-based and MHSA-GRU data-driven method for remaining useful life prediction of rolling bearings”. *Journal of Vibration and Control*, 31(15-16):3295–3312. DOI: [10.1177/10775463241273080](https://doi.org/10.1177/10775463241273080).
- Zhao, R., Wang, D., et al. (2018). “Machine Health Monitoring Using Local Feature-Based Gated Recurrent Unit Networks”. *IEEE Transactions on Industrial Electronics*, 65(2):1539–1548. DOI: [10.1109/TIE.2017.2733438](https://doi.org/10.1109/TIE.2017.2733438).
- Zhuang, J., Chen, Y., et al. (2024). “A Graph-Embedded Subdomain Adaptation Approach for Remaining Useful Life Prediction of Industrial IoT Systems”. *IEEE Internet of Things Journal*, 11(13):22903–22914. DOI: [10.1109/JIOT.2024.3361533](https://doi.org/10.1109/JIOT.2024.3361533).
- Zonta, T., Da Costa, C.A., et al. (2020). “Predictive maintenance in the Industry 4.0: A systematic literature review”. *Computers & Industrial Engineering*, 150:106889. DOI: [10.1016/j.cie.2020.106889](https://doi.org/10.1016/j.cie.2020.106889).

Appendix A

SLR of techniques used for RUL Prediction

A.1 Introduction

The advent of intelligent manufacturing environments, equipped with condition monitoring sensors, has made procuring equipment health data increasingly accessible. This has allowed for rapid advancements in proactive data-driven equipment maintenance strategies. Predictive maintenance (PdM), which consists of timely corrective actions before the failure of assets, has gained significant attention in recent literature due to its ability to utilise this data. When successfully implemented, this approach reduces downtime, extends equipment lifetime and minimises maintenance costs by performing repairs only when necessary (Nunes et al., 2023). Forecasting the time equipment will function before failure occurs is a key aspect in many PdM strategies, often referred to as 'Remaining Useful Life (RUL) prediction'. However, the performance of RUL prediction approaches is highly dependent on the selection of appropriate techniques used throughout the model development pipeline.

[Appendix A](#) uses an SLR approach to summarise the commonly used datasets, feature-extraction approaches, predictive modelling techniques, optimisation and hyperparameter-tuning strategies, and evaluation metrics used in industrial PdM contexts. In addition, it examines how frequently studies report reduced-training-data experiments and synthesises the limitations and challenges identified in the literature before concluding with commonly proposed future research directions. Additional background on maintenance strategies and predictive maintenance is provided in Chapter 1; this appendix focuses specifically on the SLR methodology and findings.

A.2 Overview of the SLR

The SLR focuses on PdM using data-driven techniques applied to time series data of industrial assets and equipment. The aim is to answer the following research questions:

Q1: What are the annual trends of the publications?

Q2: What is the geographical distribution of the publications?

Q3: What articles are most prominent in the field?

Q4: What are the common challenges and limitations identified in PdM contexts by the literature?

Q5: What type of time series data is used?

Q6: What feature extraction techniques are used?

Q7: What prediction algorithms are used?

Q8: What optimisers are used?

Q9: What hyperparameter tuning techniques are used?

Q10: What performance evaluation metrics are used?

Q11: What future research opportunities are commonly proposed amongst the publications?

adapted from (Alzubi et al., 2018) To address Q1–Q3, the SLR first documents the SLR first presents the methodology used to identify and select the publications included in this review (refer to [Appendix A.3](#)). Thereafter, the common challenges observed in predictive maintenance (PdM) for industrial equipment are investigated (Q4; [Appendix A.4](#)). The remaining questions are answered by organising the synthesis according to the typical machine-learning workflow (refer to [Figure A.1](#))

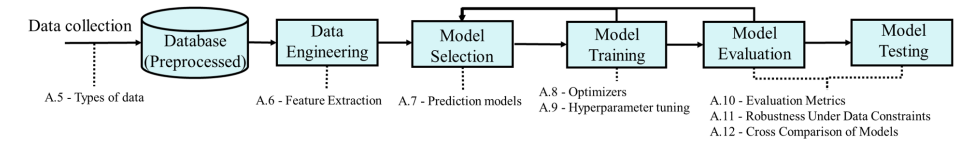


Figure A.1: The typical machine-learning workflow used to structure the SLR, adapted from, Alzubi et al. (2018).

Specifically, [section A.5](#) summarises the types of datasets and time-series data used in the selected studies (Q5). [Appendix A.6](#) reviews the data engineering approaches used,

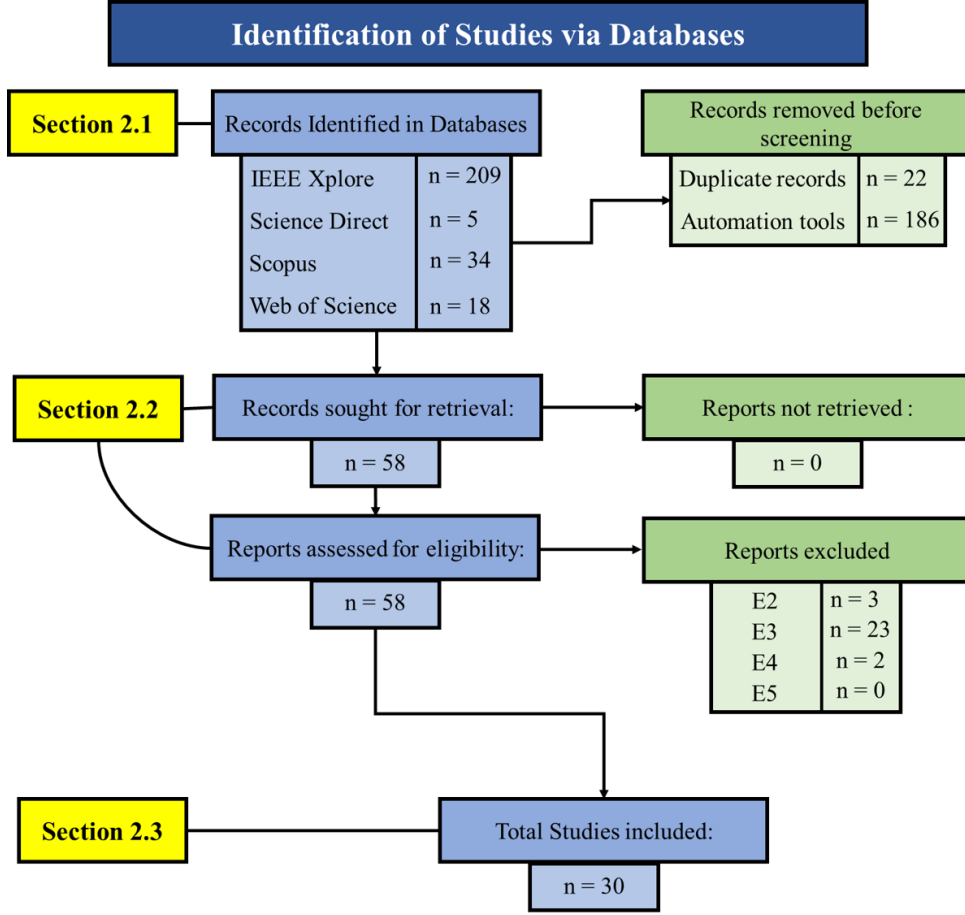


Figure A.2: PRISMA 2020 flow diagram of methodology for the systematic literature review, adapted from Page et al. (2021).

focused on **feature selection (FS)** and feature extraction (FE) techniques applied (Q6). [Appendix A.7](#) then discusses the prediction models used for RUL estimation is discussed (Q7). Next, the optimisation strategies and hyperparameter tuning methods are discussed in [Appendix A.8](#) and [Appendix A.9](#) respectively (Q8-Q9). Performance evaluation metrics are consolidated in [Appendix A.10](#) (Q10). Finally, the future research opportunities most frequently proposed in the literature are consolidated in [Appendix A.11](#) (Q11).

A.3 Systematic Literature Review Methodology

To identify relevant articles that address these questions, the PRISMA 2020 approach was utilised. Originally established as PRISMA 2009 (Liberati et al., 2009), the refined version (PRISMA 2020) is well-defined in the publication by Page et al. (2021). The approach consists of three parts: identification of records from databases, screening of the identified works, and selection of the publications for the review. Fig. X. provides a high-level overview of the process, which will be discussed in the subsequent sections.

In line with the PRISMA 2020 methodology, Section 2.1 first outlines the databases and search queries used to identify relevant records. Section 2.2 details the exclusion criteria applied to the identified publications. Finally, Section 2.3 presents the selected articles.

A.3.1 Records identified in Databases

The first step of PRISMA 2020 was to select relevant databases to retrieve the records. These databases should have extensive collections and advanced search capabilities to effectively retrieve relevant literature for the field of data-driven PdM. The subsequently identified databases were IEEE Xplore, ScienceDirect, Scopus, and Web of Science.

Each database offered advanced metadata search options. IEEE Xplore provided a meta-data search option where users could apply the 'All Metadata' command to each search term. ScienceDirect and Scopus allowed for metadata (title, abstract, keywords) search. However, ScienceDirect is confined to 8 Boolean operators. Lastly, for Web of Science, the All-fields search was used. The resulting queries and the number of unfiltered identified publications for each database are shown in [Table A.1](#).

Table A.1: List of the databases, search queries and number of results identified.

Database	Search Query	Results
IEEE Xplore	((("All Metadata": Predictive Maintenance) OR ("All Metadata": PdM)) AND ("All Metadata": time series data OR ("All Metadata": time series) OR ("All Metadata": temporal data)) AND (("All Metadata": Machine Learning) OR ("All Metadata": ML) OR ("All Metadata": data-driven)) AND (("All Metadata": remaining useful life) OR ("All Metadata": RUL))	209
ScienceDirect	("Predictive Maintenance" OR "PdM") AND ("time series" OR "temporal data") AND ("Machine Learning" OR "ML" OR "data-driven") AND ("Remaining useful life" OR "RUL")	5
Scopus	("Predictive Maintenance" OR "PdM") AND ("time series data" OR "time series" OR "temporal data") AND ("Machine Learning" OR "ML" OR "data-driven") AND ("Remaining useful life" OR "RUL")	34
Web of Science	ALL = (("Predictive Maintenance" OR "PdM") AND ("time series data" OR "time series" OR "temporal data") AND ("Machine Learning" OR "ML" OR "data-driven") AND ("Remaining useful life" OR "RUL"))	18

The search was conducted in each of these databases on September 2, 2024. IEEE Xplore identified 209 articles, ScienceDirect yielded 5, Scopus retrieved 34, and Web of Science resulted in 18 articles

A.3.2 Screening the Identified Publication

All articles were accessible and successfully retrieved. These articles were then subjected to the following exclusion criteria:

- E1.** Works that are not published, peer-reviewed journal articles written in English, including surveys, reviews, and conference proceedings.
- E2.** Works that do not include evaluation metrics.
- E3.** Works that do not explain how hyperparameters were found.
- E4.** Works that do not discuss RUL prediction techniques/algorithms used.
- E5.** Works that do not reference (or explain the origin of) the datasets used.

Only studies that provided details such as the origin of the data, algorithms/methods used for model development, hyperparameter optimisation/selection method and evaluation metrics were included to ensure the resulting studies are replicable and reliable. Furthermore, these criteria ensured that the selected articles directly addressed research questions 4, 6, 8, and 9, specifically focusing on data types, predictive modelling approaches, hyperparameter optimisation procedures, and performance evaluation methodologies. It is acknowledged that research questions 5 (FE techniques), 7 (optimisers) and 10 (future research) may not be relevant to all studies in the context of RUL prediction for PdM. Therefore, these were not included in the exclusion criteria. However, data on these areas were extracted from studies that explicitly mention them to address the applicable questions.

Figure A.3 shows the overview of how the articles were filtered according to the exclusion criterion. Using the filters provided by these databases, 186 articles were excluded in accordance with exclusion criterion E1. Additionally, among the resulting articles, 22 duplicates were found, leaving 58 articles for further screening. 3 research papers were then excluded based on E2, 23 were excluded due to E3, 2 were excluded based on E4, and no articles were excluded due to E5.

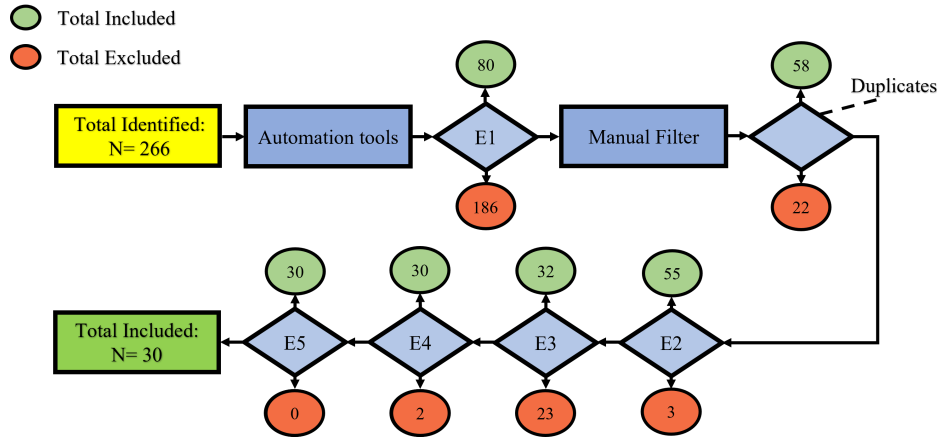


Figure A.3: Filtering process of identified publications

A.3.3 Selected Publications

A total of 30 publications passed all criteria and were included for analysis in the SLR. Figure A.4 shows a table of these publications according to year. Next to the table, a line graph illustrates the annual trend of the number of publications. A general increase over time is evident, peaking in 2022 and 2024, with 7 and 6 publications, respectively.

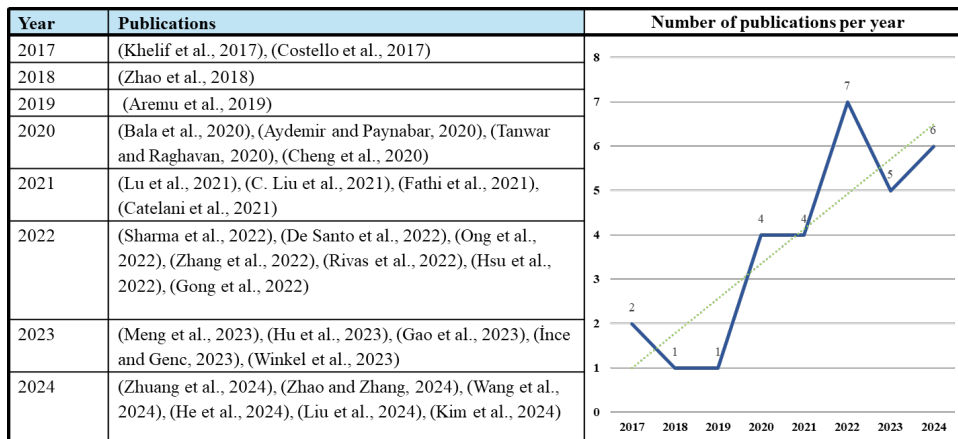


Figure A.4: Number of publications per year and corresponding references

A.3.4 Geographical distribution and prominence of the selected works

Figure A.5 presents a choropleth map that visualises the geographical distribution of studies included in the SLR. China has the highest contribution with 14 publications, while research in this field shows global interest, with smaller contributions from 13 other countries, including Australia, France, Germany, India, Italy, Malaysia, the Republic of Korea, Singapore, Switzerland, Taiwan, Turkey, UK and the USA.

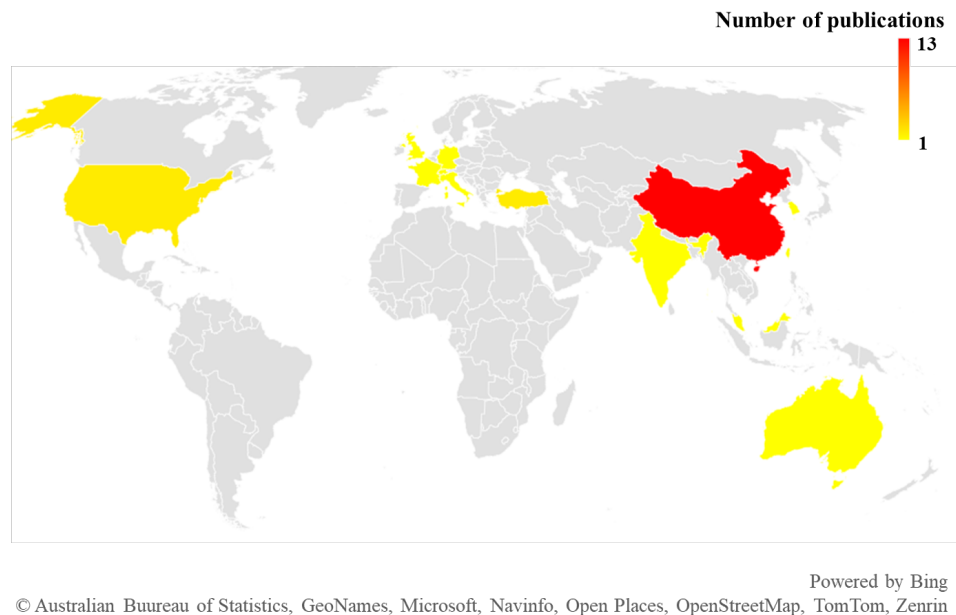


Figure A.5: The choropleth map that represents the geographical distribution of the identified articles

The publications varied in their impact on the field of PdM. Analysing citation counts helped to identify the most influential studies and assess their significance in advancing research. Table A.2 presents the citation counts for the selected works as of October 25, 2024. In addition, Figure A.6 presents the distribution of citations across the studies.

Table A.2: Citation distribution of the selected works.

Reference	Respective Countries	Citations
Zhao et al. (2018)	China	808
Khelif et al. (2017)	France	414
Zhang et al. (2022)	China	173
Cheng et al. (2021)	China	120
Catelani et al. (2021)	Italy	118
De Santo et al. (2022)	USA	84
Hu et al. (2023); Ong et al. (2022)	China, Singapore	50–59
Costello et al. (2017); Aydemir and Paynabar (2020)	UK, Turkey	40–49
Hsu et al. (2022); Lu et al. (2021); Bala et al. (2020); Aremu et al. (2019); Liu et al. (2021); Meng et al. (2023)	Taiwan, China, Malaysia, Australia, China, China	30–39

Kumar Sharma et al. (2022); Tanwar and Raghavan (2020); Gong et al. (2022)	India, Singapore, China	20–29
Fathi et al. (2021); Gao et al. (2023); Rivas et al. (2022)	Switzerland, China, USA	10–19
He et al. (2024)	China	5–9
Kim et al. (2024); İnce and Genc (2023); Wang et al. (2024); Winkel et al. (2023); Liu et al. (2024); Zhuang et al. (2024)	Republic of Korea, Turkey, China, Germany, China, China	1–5
Zhao and Zhang (2024)	China	0

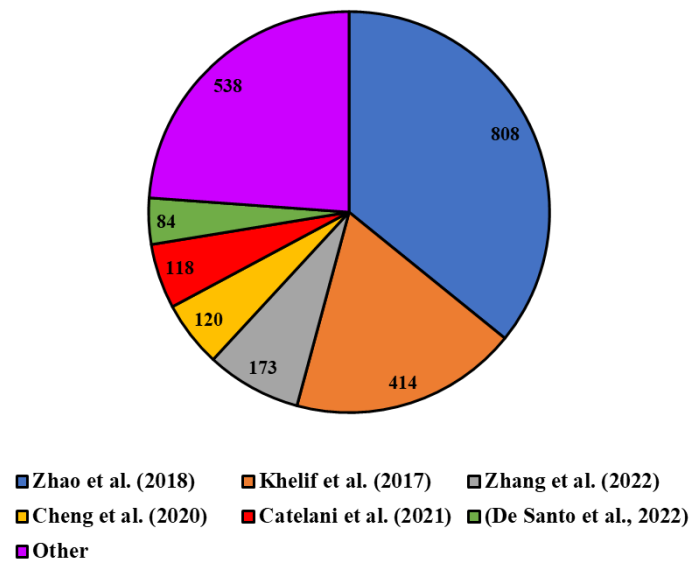


Figure A.6: Pie chart showing the proportion of citations across selected studies

Zhao et al. (2018) and Khelif et al. (2017) rank among the most highly cited contributions in the corpus, with 808 and 414 citations, respectively. In addition, several relatively recent works—Zhang et al. (2022), Cheng et al. (2021), Catelani et al. (2021), and De Santo et al. (2022) have also shown a substantial influence on the field, with 173, 120, 118, and 84 citations, respectively. The majority of the remaining publications have accrued between 20 and 40 citations. Publications with fewer than five citations are predominantly very recent (2023 or later), which is expected given the limited time available for newer works to accumulate citations by the search date.

A.4 Limitations and Challenges Commonly Found in Predictive Data Maintenance (PdM)

Three common types of limitations/ challenges have been found to affect predictive data maintenance (PdMs) that use data driven methods for predicting remaining useful life (RUL). The first challenge is that labelled degradation and run-to-failure data is very difficult to find. This is made even harder when the data contains class imbalances; it has insufficient sensors available; and/or there is insufficient measurement data available to extract representative degradation information from the data (İnce and Genc, 2023; Fathi et al., 2021; Kim et al., 2024; He et al., 2024; Gong et al., 2022).

The second challenge is that each piece of industrial equipment operates under variable conditions. Each unit within an industry may operate differently due to variable working environments; or failure modes may vary over time. Additionally, degradation behaviour may be non- monotonic. All of these factors can lead to different distribution patterns in both the training set and the test set, making it difficult for a model to generalise its knowledge to other operating condition-based testing scenarios (Winkel et al., 2023; Tanwar and Raghavan, 2020; Wang et al., 2020; Cheng et al., 2021).

Third, another type of challenge was identified as being related to the predictability and uncertainty of the predictions produced by RUL models. RUL models typically produce a single point estimate value as their output. They do not attempt to represent the degree of uncertainty that exists with respect to their predictions. This lack of representation of the uncertainty associated with the predictions limits their utility when used in risk informed decision-making processes for maintenance (Tanwar and Raghavan, 2020; Rivas et al., 2022; He et al., 2024).

These findings address Q4 of the SLR. A more detailed discussion of these identified limitations and challenges, together with their relevance to the present study, is provided in [section 2.5](#) in the main body of the dissertation.

A.5 Types of data used

Given the data availability, labelling, sensor, and operating-condition challenges outlined in [section A.4](#), it is informative to first characterise the datasets used in the selected studies. This section summarises whether the data are real-world or synthetic and identifies the industrial equipment from which these datasets are sourced across the 30 selected publications. Advances in sensing technologies, wireless communications, and computing systems have greatly expanded the amount of historical data available from manufacturing systems (Zhao et al., 2018). Data-driven machine health monitoring systems can utilise

this historical data to train predictive models and make real-time decisions based on sensor inputs from monitored equipment.

Data can either be real, meaning it is collected from actual equipment working in real-world environments, or synthetic, meaning it is generated artificially through simulations or algorithms to mimic these real-world environments. [Figure A.7](#) illustrates the proportion of studies that used synthetic and real-world data. Each study was tallied based on the presence of real or synthetic datasets. Regardless of the number of datasets of the same type, each study contributed only one count per type.

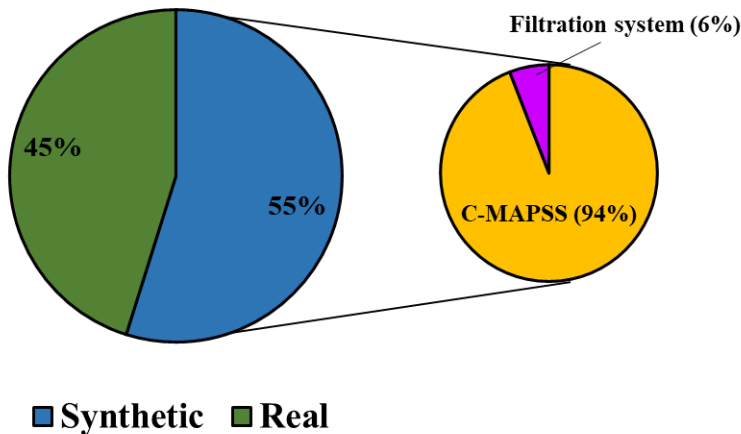


Figure A.7: Proportion of studies using synthetic vs real-world data

The results show that more publications made use of synthetic data (55%) than real-world data (45%). However, the majority of synthetic datasets used were from the C-MAPSS dataset (94%), which is sourced from simulated aircraft turbofan engines. In contrast, the real-world data in these studies comes from diverse sources. [Figure A.8](#) highlights this by presenting a breakdown of the industrial equipment from which the data in the studies were sourced.

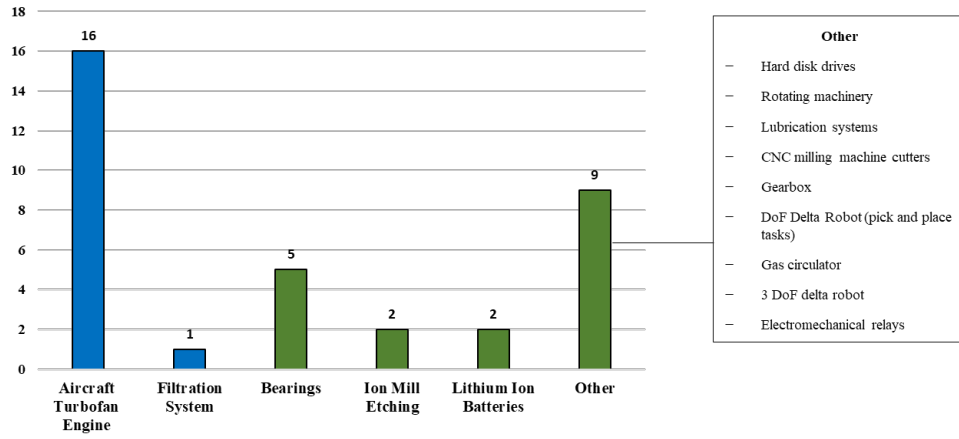


Figure A.8: Distribution of industrial equipment sources represented in the reviewed studies

The range of real-world data sources included 5 studies using bearing data, 2 using data from ion mill etching tools and 2 sourced from lithium-ion batteries. The rest of the sources were from various other types of industrial equipment, as represented in the ‘other’ category in Figure A.8.

One reason studies tend to favour real data over synthetic data is that synthetic data often fails to capture real-world complexities, such as waveform source signatures, realistic noise, and accurate reflectivity. This can result in poor performance of trained neural networks during inference (Alkhalifah et al., 2022). Despite this the majority of publications extracted for the SLR made use of a synthetic dataset known as C-MAPSS. C-MAPSS stands for Commercial Modular Aero-Propulsion System Simulation. It is a software simulation tool (developed by NASA’s Prognostics Center of Excellence) that models the behaviour of large commercial turbofan engines with a control system that closely mirrors actual engine operations (Frederick et al., 2013). NASA’s Prognostics Center of Excellence have used this simulation environment to create a set of datasets that are well-established benchmark for PdM-related research (Wang et al., 2020). These datasets, referred to as FD001, FD002, FD003, and FD004, simulate different engine degradation scenarios and operating conditions.

Table A.3. lists the references for all publicly available or external datasets used in the studies (proprietary datasets have been excluded).

Table A.3: Non-proprietary datasets

Dataset Name	Data Source	Reference
C-MAPSS	Aircraft Turbofan Engine	(Saxena et al., 2008b)
IEEE PHM 2012 Prognostic Challenge	Bearings	(Nectoux et al., 2012)

Dataset1_BatteryAgingARC-FY08Q4	Lithium-Ion Batteries	(Saha and Goebel, 2007)
XJTU-SY bearing datasets	Bearings	(Wang et al., 2020)
Incipient Fault Diagnosis of Rolling Element Bearings	Bearings	(Wei et al., 2011)
2010 PHM data challenge	CNC milling machine cutters	(Prognostics and Health Management Society, 2010)
PHME20	Filtration system	(Prognostics and Health Management Society, 2020)
Phoenix Contact Relay Dataset	Electromechanical relays	(Winkel and Kleinstauber, 2022)
Backblaze dataset	Hard disk drives	(Bauer, 2018)
S.M.A.R.T dataset	Hard disk drives	(Baidu, 2013)
2018 PHM Data Challenge	Ion mill etching tool	(Prognostics and Health Management Society, 2018)
Oxford Battery Degradation Dataset	Lithium-ion batteries	(Birkel, 2017)
WDC data	Lubrication systems	(Fan et al., 2017)

When sensor-equipped systems generate datasets for industrial assets, data-driven models for PdM can be developed to predict machine failures. The following sections discuss some of the key components found amongst the papers involved in the development of PdM models. Specifically, feature engineering, RUL-related prediction algorithms, model optimisation, and the metrics employed for model evaluation.

A.6 Feature extraction techniques

FE is often loosely defined in the literature. Sharma (2025) describes it as a methodology used to minimise the number of features to improve computation efficiency. Zhao et al. (2018) defines FE as a method to transform original features into a new informative feature space. Some studies consider certain data preprocessing methods to be a form of FE. For example, in Fathi et al. (2021) research, the technique of time windowing was referred to as FE. However, time windowing doesn't involve creating new features from raw data but rather segmenting data to make it easier for models to process temporal dependencies.

To answer Q5, FE is defined here as a technique that derives new helpful information

from data and feeds it into a separate model for further FE or prediction. Therefore, models used to make the final RUL prediction, which may inherently extract features, are not included here but are discussed in Section 3.4, focusing on RUL prediction models. This section only tallies studies that apply methods fitting the above FE definition to identify the most commonly used techniques in the field. Fig. 9. shows the proportion of algorithms and techniques used across selected publications.

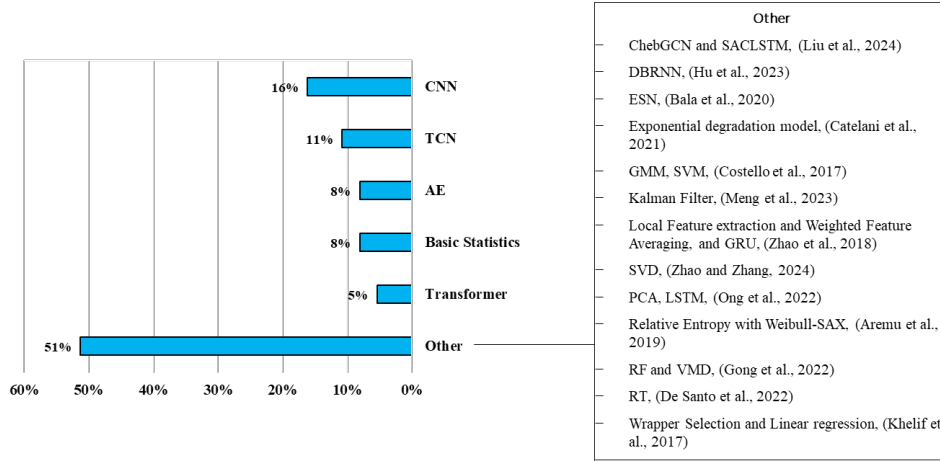


Figure A.9: Distribution of algorithms/techniques found in the literature for feature extraction

Across the 30 studies, a total of 38 FE techniques were tallied, comprising 24 distinct techniques. Since some studies used multiple techniques, the percentages are calculated based on the total number of techniques rather than the number of studies. Of these, 51% were unique to individual studies. The most used technique was Convolutional Neural Network (CNN)-based algorithms, which appeared in 16% of the techniques used. TCN made up 11% of the techniques used. Basic statistical and Auto Encoder (AE) techniques were applied 8% of the time for FE. Another algorithm that appeared in more than one instance included Transformers, representing 5% of the techniques used. In sections 3.3.1 to 3.3.5 the FE techniques that were employed multiple times across the studies are discussed. For techniques that were only applied in a single study, readers can refer to the 'other' category in Fig. 9. where both the algorithm name and the corresponding reference are provided for further details.

A.6.1 Convolutional neural networks

CNNs are frequently employed to extract complex degradation features from condition monitoring data (Liu et al., 2024). These algorithms are, according to Zhang et al. (2020), one of the most successfully applied ML algorithms that are known for their ability to learn and express features. It involves the use of multiple layers of filters to slide over

input sequences to extract hierarchical features through localised data patterns (Zhang et al., 2020). One key advantage CNNs have is that it has local FE capabilities and fast computational speed (Gao et al., 2023). In addition, they are useful in extracting the spatial features of data.

He et al. (2024) used a cascaded multiscale CNN module. This involved combining depth-wise separable convolution and dilated causal convolution to extract spatial and temporal degradation features respectively from veering vibration data at multiple scales. To focus on the most important features the study used a convolutional block attention module. In the research done by Wang et al. (2020) a 1D-CNN was used to extract local spatial features from a high-dimensional dataset (C-MAPSS). The CNN features were then used as inputs for a bidirectional-Long Short-Term Memory (bi-LSTM) predictor, where the Long Short-Term Memory (LSTM) networks could then identify the temporal dependencies and perform the prediction. Gao et al. (2023) also utilised CNN layers in a dual-channel feature attention approach that combined CNN and Bi-LSTM layers to extract spatial and temporal features, respectively. The first CNN layer extracted intermediate features along the time axis, and then pooling techniques were used to recognise the significance of different sensors in the spatial attention module. Additionally, a multilayer CNN was used to extract deeper spatial features from the weighted sensor data. These extracted spatial features were then fused with temporal features extracted using a bidirectional LSTM. Aydemir and Paynabar (2020) also used CNNs in a CNN-LSTM pipeline. Their model involved extracting spatial features from infrared image stream images collected from rotating machinery. After two convolutional layers, pooling techniques were applied to reduce the dimensionality of the extracted spatial features. Then those were passed through an LSTM layer to capture the temporal dependencies and make the time-to-failure prediction. In the research done by Fathi et al. (2021), CNN layers were used within an autoencoder architecture to capture spatial information in data from a 3 DoF (degrees of freedom) Delta Robot. Notably, the authors decided not to use pooling layers in the CNN, as they argued that the convolutional layers alone captured sufficient spatial information, and pooling would unnecessarily complicate their model. Liu et al. (2021) used a 1-D CNN to extract high-level features from operating condition data, while Temporal Convolutional Networks (TCN) were used to extract the temporal features of the data. After FE, a dual attention mechanism was used to enhance the feature representations by capturing global dependencies, which helped the model focus on the most important degradation patterns.

A.6.2 Temporal convolutional networks

TCNs have emerged as a powerful model for sequence modelling. These models are found to significantly outperform traditional recurrent architectures like LSTM and Gated Recurrent Unit (GRU) in some applications (Liu et al., 2021; Hsu et al., 2022). A key

component of TCNs is the use of causal convolutions. Unlike general convolutions, causal convolutions prevent information from the future from influencing the past (Liu et al., 2021). This greater receptive field gives the model advanced long-term memory capacity (Hsu et al., 2022). TCNs are also flexible enough to handle input sequences of varying lengths while producing an output sequence that matches the length of the input, which makes it advantageous for applications where sequences vary (Liu et al., 2021).

Apart from Liu et al. (2021), three other studies made use of TCNs for FE. Zhuang et al. (2024) used TCNs for manifold learning, which involved dividing degradation features into local manifolds to represent regions of the feature space. These manifolds were then aligned across different machine domains to minimise discrepancies between similar subdomains, which allowed for better generalisation for the RUL model. Their framework consisted of a temporal subdomain network that relies on TCN with a normalisation-based attention module to extract deep degradation features from vibration signals of bearing data. Hsu et al. (2022) used TCNs to extract temporal features from ion mill etching tool sensor data. Their approach involved integrating an attention mechanism after the TCN to prioritise the most important temporal features for RUL prediction. Gong et al. (2022) used TCNs with a Bi-GRU model to extract temporal features. Random Forest (RF) was used to weigh the features according to importance and filter the noise. Following this, Variational Model Decomposition (VMD) was applied to the weighted raw sensor data to decompose the signals into intrinsic mode functions. The TCNs then helped capture the key temporal patterns in the data to aid in prediction further down the pipeline.

A.6.3 Autoencoders

Another FE algorithm used by three of the selected publications is autoencoders (AE). AE can be defined as a specific kind of neural network that is primarily intended to encode information into a condensed and meaningful representation and then decode it back to produce a reconstruction of the input that is as close to the original as feasible. Aydemir and Paynabar (2020) describe AE as a nonlinear extension of Principle Component Analysis (PCA). PCA is a linear technique used to reduce the dimensionality of data, but only effectively in linear relationships. On the other hand, AE approaches often use nonlinear activation functions for both the encoder and decoder, which can capture more complex nonlinear patterns (Aydemir and Paynabar, 2020).

İnce and Genc (2023) used a convolutional AE. This involved an encoder consisting of three convolutional layers to extract spatial information and then three deconvolutional layers to decode the signals. These features were then passed to an LSTM network for temporal modelling and prediction. Lu et al. (2021) used an AE as part of a deep adversarial learning model. The AE was used to compress the data into lower-dimensional features, which were then used by the discriminator to assess the generated predictions. Recall that

Aydemir and Paynabar (2020) used a CNN-LSTM approach for FE. However, in addition to this, they also investigated an AE-LSTM approach. This involved two encoding layers and one decoding layer to reduce the dimensionality of the degradation images of rotating machinery. Notably, the AE-LSTM performed better in nonlinear scenarios compared to the CNN-LSTM and performed competitively in linear scenarios where the time to failure was modelled as a direct linear relationship.

A.6.4 Traditional Statistics

While deep learning-based approaches have been proven successful for FE. Several studies have found success in utilising more traditional methods by extracting basic statistics such as Root Mean Square (RMS), standard deviation (SD) and the mean from data. Zhao and Zhang (2024) used RMS for FE from vibration signals of rolling bearings. The RMS was utilised as a Health Indicator (HI) to reflect the magnitude of a varying signal. It was calculated for each time window of the vibration signals and provided a straightforward yet effective way to extract features that correlate with the bearing's health degradation. Lu et al. (2021) also only used one statistic instead of multiple statistical features (such as kurtosis, mean value, etc.) since they argued that it would unnecessarily increase the complexity of tasks in their approach. They therefore decided to infer the RUL using only the RMS. On the other hand, in the research done by Costello et al. (2017), multiple simple statistical features were used, including mean, SD and RMS from vibration signals of gas circulators. The statistics were calculated for each segment of the vibration data and served as indicators of the condition of the machine. Although these FE techniques can be effective and are simple to implement, they often fail to capture deeper relationships in the data. Consequently, studies often use other FE techniques in conjunction. For example, recall Lu et al. (2021) extracted features through traditional statistics and utilised AE as part of their FE approach.

A.6.5 Transformers

Transformers are a recently introduced sequence modelling architecture that has demonstrated superior performance in tasks involving complex data for prediction (Zhang et al., 2022; Kim et al., 2024). It utilises self-attention mechanisms to capture long-term dependencies between elements in a sequence regardless of their distance. This makes it less impacted when increasing sequence length compared to traditional methods like Recurrent Neural Networks (RNNs) and LSTMs (Zhang et al., 2022). The basic components of a transformer model include the embedding/positional encoding, attention, and prediction layers (Kim et al., 2024)

Zhang et al. (2022) utilised a transformer architecture with a dual-aspect self-attention

mechanism. This involved an encoder-decoder structure that includes encoder, feature fusion, and decoder layers. Their approach focuses on capturing long-term dependencies in the input-output sequence using self-attention to ensure the importance of each work cycle is kept regardless of time step length. For FE, a dual-aspect encoding mechanism with parallel sensor and time-step encoders was used to capture weighted features across the sensors and times steps. The features obtained from both the sensor encoder and time-step encoder were then combined in a feature fusion layer, which creates a new feature map that contains the important information. Kim et al. (2024) also made use of a transformer encoder architecture but integrated multi-head-self-attention modules. Moreover, embedding layers were used to transform the raw data into readable vector representations for the model. They note that traditionally, transformer-based approaches use sinusoidal encoding which is fixed and non-learnable. Therefore, they opted to use 'Time2Vec' encoding which allowed for learnable vector representations and flexible embedding. The FE layers of their model consisted of multiple attention blocks and a global pooling layer that focused on capturing key features from both sensor data and time steps simultaneously. Multi-head attention modules were used to attend to the different aspects of the data at various timestamps during training.

A.7 Prediction models

Selecting an appropriate model for training will directly impact the performance of the final RUL prediction. Typically, RUL prediction models involve regression approaches since the goal is to predict a continuous value that represents the time remaining before a machine or component fails (Si et al., 2011). This section focuses on the algorithms used by the selected works that function to output the actual RUL predictions. Additionally, in Section 3.4.6 the most common architecture pipeline (FE and RUL prediction model) combinations are shown. Fig. 10. shows the proportion of algorithms found in the literature for the actual RUL-related predictions.

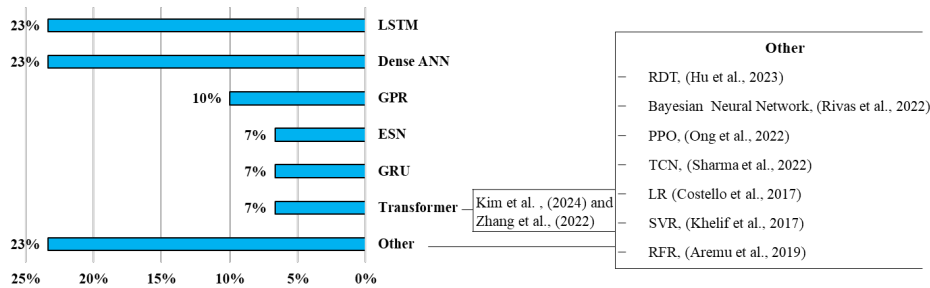


Figure A.10: Distribution of algorithms/techniques found in the literature for RUL prediction

The most used approach found for RUL-related predictions was LSTM and dense Artificial Neural Networks (ANN) based models (both found in 23% of the papers). This is followed the Gaussian Process Regression (GPR) which was used in 10% of the studies. Echo State Networks (ESN), GRUs and Transformers each appeared in 7% of the studies. 21% make up the ‘other’ category (consisting of prediction algorithms that occur only once among the literature). For more information on how the algorithms, in the ‘other’ category, were used refer to the references provided in Fig. 10. The subsequent sections discuss how LSTM, dense ANNs, GPRs, ESN and GRU were used for RUL prediction in the studies.

A.7.1 LSTM for RUL prediction

LSTM is a variant of RNN-based structures. RNNs are limited to short-term memory, whereas LSTM networks consist of gate control mechanisms that can manage both short-term and long-term memory (Chen et al., 2021). The canonical architecture of a Long Short-Term Memory (LSTM) network comprises three primary gating mechanisms: the forget gate, the input gate, and the output gate. The forget gate decides which information from the previous cell to discard. LSTM is known for addressing the training difficulties that are associated with long sequence data. For example a key benefit of LSTM is that they are designed to address the vanishing gradient problems that often occur in traditional RNNs (Ullah et al., 2023). This is because LSTM has cells that allow the network to learn long-range temporal dependencies while having a forget gate to remove hidden states (Lu et al., 2021). Therefore this model can better handle long-term sequential data and address the problem of long-term dependency through its memory function (Chen et al., 2021).

Lu et al. (2021) addresses the prediction error superposition problem by proposing a deep adversarial LSTM framework. They used a generative adversarial network architecture that combined LSTM and AE models for bearing health degradation prediction. While the AE, as previously mentioned (Section 3.3.3), was used as a discriminator, the LSTM was used as the generator to predict the degradation of the bearing’s health. The adversarial feedback loop between the generator and discriminator ensured that the predictions aligned with the actual data and any errors that were identified were corrected before they accumulated over multiple steps. This approach showed promising results, outperforming other state-of-the-art ML approaches for vibration-based monitoring of bearing degradation.

Aydemir and Paynabar (2020) proposed two methods for RUL prediction of image data from rotating machinery, both utilising LSTM networks for the actual prediction. As mentioned in sections 3.3.1 and 3.3.3 the two approaches included AE-LSTM and CNN-LSTM frameworks, where the AEs and CNNs were used for FE. The LSTM layers functioned to extract the temporal information from the features generated by those algorithms. Then the models were trained using backpropagation through time algorithm. They opted for

using a weighted average of the squared error, instead of the commonly preferred MSE, as the cost function prioritises accurate predictions as the system approaches failure.

Cheng et al. (2021) proposed an ensemble of LSTM neural networks to predict RUL. This involved training multiple LSTM models from different identical equipment. The outputs of these LSTM neural networks were then combined using a Bayesian inference algorithm to select the optimal RUL from the multiple predictions. The proposed ensemble model was able to minimise the influence of varying equipment life spans on RUL prediction accuracy and increase the model's ability to adapt and generalise across different prediction scenarios. Their approach was found to competitively perform in comparison to other state-of-the-art approaches.

Hsu et al. (2022) used LSTM to capture temporal dependencies from the outputs of TCNs. To address the limitations of LSTM, which traditionally only uses hidden state information from the final step, the model incorporated all hidden states from the previous time steps. To do this the study used an attention layer to selectively focus on the relevant parts of the input sequence. Then two fully connected layers were used to map the LSTM representations into RUL predictions.

De Santo et al. (2022) proposed an LSTM-based model to predict the health status of hard disk drives. The model leveraged Self-Monitoring Analysis and Reporting Technology (SMART) attributes to monitor hard disk drive health over time. Their results showed that the approach outperformed state-of-the-art methods, with the model being able to predict hard drive health status up to 45 days before failure.

İnce and Genc (2023) proposed a joint autoencoder-regressor model which makes use of a convolutional autoencoder (as mentioned previously in section 3.3.3) and an LSTM-based regressor. The RUL regression layers contain two LSTM layers which extract temporal information from features extracted by the encoder. Subsequently, dense layers were used to generate predictions for RUL, at specific time steps, based on this temporal information. They note that using more than one output in neural networks (known as multitask learning) is anticipated to improve the generalisation of their approach.

Wang et al. (2024) introduced a bidirectional LSTM network with an attention mechanism for predicting the RUL of turbofan engines using the C-MAPSS dataset. To improve the model's prediction performance Bayesian optimisation was used to optimise the hyperparameters. Additionally, the study proposed a scheduling framework using mixed-integer linear programming to optimise maintenance scheduling which was aimed at minimising the total maintenance time. The results demonstrated that the proposed model and scheduling system effectively monitored aircraft engine health in real-time and significantly reduced maintenance time.

A.7.2 Dense feed-forward neural network for remaining useful life prediction

Feed-forward networks (FFNs) are known by many interchangeable names, such as ANN and Multilayer perceptions (MLPs). The general structure of an FFN comprises of an input layer, one or multiple hidden layers and an output layer. A key characteristic of FFNs is the absence of recurrent connections, meaning that data flows in only one direction. All the studies that used FFNs for prediction made use of fully connected FFNs. A fully connected FFN, also referred to as a dense-ANN, is one where all the neurons in each layer are connected to all the neurons in the next layer.

Zhao et al. (2018) proposed a local feature-based GRU. It involved generated sequences of local features that passed through a bidirectional GRU network to realise the data representation. These representations were then passed through a fully connected feed-forward network and linear regression layer to predict the machine condition. Liu et al. (2021) used a dual attention-based TCN to extract degradation features which were passed through fully connected layers that served as the regression machine to predict the RUL. The primary loss function used to train the regressor was MSE. Additionally, they used the multi-kernel maximum mean discrepancy (MK-MMD) loss function to minimise discrepancy between source and target domains which improved the model's ability to perform under differing operating conditions. Winkel et al. (2023) used an RUL prediction technique based on MLP which was used as part of their approach that focused on a Maintenance Algorithm for Unlabeled Data (MAUD). The MAUD framework involves pseudo-labelling, and the MLP was trained on both labelled and unlabeled data (semi-supervised) to predict the failure risk of electromechanical relays.

In the research done by Gao et al. (2023), a feature fusion network and fully connected layers were used to estimate the RUL. The fusion network was used to combine the features generated by the dual channel feature attention framework to learn a higher representation of the data. The output was then passed through fully connected layers for prediction. To improve the reliability of the RUL predictions (Gao et al., 2023), used the Monte Carlo dropout method, which characterised the uncertainty of predictions by simulating different dropout scenarios during testing. This enabled them to provide a confidence interval for the RUL estimates.

He et al. (2024) employed a different uncertainty quantification strategy. Their framework integrated; depth-wise separable convolution, dilated causal convolution, and a convolutional block attention module to extract multiscale degradation features in spatial and temporal dimensions. These features were then passed to a dense ANN for RUL prediction. A staged-optimisation layer was then used to quantify the uncertainty of the prediction. Their approach avoided overconfident point predictions (compared to traditional CNNs that often overlook uncertainty caused by noise) and thus provided a more

realistic confidence interval without changing the network structure.

After extracting spatial and temporal features via their proposed SGIAEC framework, Liu et al. (2024) used a mapping module consisting of several fully connected layers (dense ANN) to predict the RUL. The model's training process was guided by the MSE loss function. Their proposed approach incorporated dropout regularisation to help prevent overfitting and improve generalisation.

Zhuang et al. (2024) introduced a graph-embedded subdomain adaptation network to forecast the RUL across various machines in an Industrial Internet of Things (IIoT) environment. Their approach incorporated manifold graph construction, similarity evaluation, and discrepancy learning to refine the model. A regressor with four fully connected layers was then used, utilizing the Sigmoid activation function in the final layer for RUL prediction. The model was trained in an end-to-end manner using the MSE loss function.

A.7.3 GPR for remaining useful life prediction

GPR is a non-parametric kernel-based supervised ML approach (Meng et al., 2023). Their nonparametric characteristic means that it does not assume a predefined form for the underlying functions. It instead functions by assigning greater probabilities to functions that are more consistent with the observed data (Fathi et al., 2021). Fathi et al. (2021) describes GPs as an infinite-dimensional multivariable distribution where any finite collection of variables follows a Gaussian distribution. Meng et al. (2023) points out that, because GPR is a non-parametric and probabilistic method, it can tackle the extrapolation issue while also quantifying the related uncertainty, i.e., the model's confidence in its predictions. Another advantage described by Fathi et al. (2021) is that trained GP models do not require numerous data points to predict anomalous values and do not require a long time to be re-trained.

Tanwar and Raghavan (2020) did research on the RUL prediction of equipment lubrication systems. They stressed that various factors make their RUL prediction task difficult such as various environmental conditions, oil replenishments, oil loss, chemical breakdowns etc. As a result, they argued that several standard statistical approaches are not suitable due to the intractability of these time-series degradation trends. To address this problem, they proposed a GPR model, taking advantage of its nonparametric Bayesian characteristic. Specifically, they used a multi-output GPR allowing them to define correlations between historical degradation trends for similar lubrication systems and the real-time degradation pattern of the system being monitored. The robustness and applicability of their approach were verified by investigating three different oil replenishment strategies under the multi-output GPR framework. Fathi et al. (2021) used the GPR algorithm for RUL prediction. The study extracted HIs from autoencoder-based anomaly detection, where a sigmoid

function was applied to predict the probabilities based on CIs. The GPR model was trained on 20 past HI data points to forecast 15 future HI values. The RUL was estimated by determining the difference between the current timestamp and the timestamp at which the HI exceeds the threshold. Meng et al. (2023) conducted research on predicting the end-of-life of batteries while incorporating uncertainty quantification. Their approach first applies a Kalman filter (KF)-based, empirical-model-free state tracking phase to process partial battery degradation data. Instead of directly forecasting the future degradation curve, this approach converts the problem into predicting two parameters (virtual degradation rate and virtual acceleration). This conversion simplified the task by reducing the complexity of the degradation trend. A GPR model was then used to predict these two parameters.

A.7.4 ESN for remaining useful life prediction

Bala et al. (2020) describe ESNs as a modern variant of RNNs, distinguished by using a dynamic reservoir that replaces the trainable hidden layers of traditional RNNs. The general structure of an ESN comprises an input layer, a reservoir (which can contain hundreds to thousands of sparsely connected neurons depending on task complexity), and an output layer (Catelani et al., 2021; Bala et al., 2020). Due to ESN's inherent simplicity in both training and architecture, Catelani et al. (2021) describe it as a fast and efficient type of RNN, recognised for its inherent simplicity in both training and architecture. A key feature of the ESN is the echo state property, which ensures that the influence of past states and initial inputs diminishes over time, rather than continuously increasing (Bala et al., 2020). This implies that ESNs help address the vanishing and exploding gradient problems that traditional RNNs often face. Bala et al. (2020) note, however, that designing the parameters and topology of an ESN presents a significant challenge.

Bala et al. (2020) focused on optimising an ESN to predict the RUL of turbofan engines (C-MAPSS) using an "Improved Grasshopper Optimisation Algorithm" (GOA). The improved GOA optimised several parameter settings including reservoir size, spectral radius, and leaky rate. It was found that their optimisation approach improved the accuracy and convergence speed of the ESN, outperforming other techniques such as Cuckoo Search and Particle Swarm Optimisation. Catelani et al. (2021) on the other hand used ESNs to predict the RUL of lithium-ion batteries. They generated training data for the ESN using both double and single exponential models. To improve prediction accuracy, the ESN was integrated with a state-space estimation technique. Additionally, they introduced a deep layer within the ESN to help capture the inherent nonlinear trends of the battery data.

A.7.5 GRU for remaining useful life prediction

The GRU is a variant of LSTM which comprises of more simplified gating mechanisms compared to LSTM and consists of reset and update gates that control the flow of information. Its structure is shown to improve the efficiency of the model in many cases (Gong et al., 2022)

In the study by Gong et al. (2022), a bidirectional GRU-based model was proposed. Specifically, two GRUs (allowing forward and backward information flow) were used to capture past and future dependencies in the data. This bidirectional architecture enabled the model to capture temporal features more effectively. Zhao and Zhang (2024) integrated the GRU with a Multi-Head Self Attention (MHSA) mechanism to predict the RUL of rolling bearings. The GRU was able to avoid issues such as vanishing and exploding gradients. The MHSA mechanism focused on different parts of the sequence to assign attention to time steps identified as important. This structure improved the model to learn from long data sequences

A.7.6 FE-LSTM and FE-Dense ANN Architectures

The studies reviewed demonstrate a range of RUL prediction algorithms, each with unique combinations of feature extraction (FE) and prediction algorithms. However, studies utilising LSTM and Dense ANN predictors tend to adopt similar architectural combinations. Figure 11 illustrates the proportion of FE techniques employed alongside LSTM and Dense ANN models.

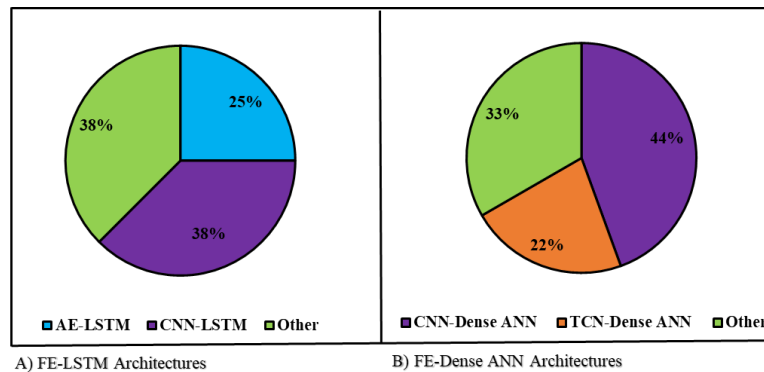


Figure A.11: Distribution of different feature extraction (FE) and LSTM (A) and Dense ANN (B) prediction model architecture combinations.

Figure A.11 A. highlights the distribution of different FE-LSTM-based architectures, with AE-LSTM making up 25%, CNN-LSTM at 38%, and other LSTM-based architectures accounting for 38%. Figure A.11 B. shows the FE-Dense ANN-based architectures, where

CNN-Dense ANN represents 44%, TCN-Dense ANN accounts for 22%, and other unique combinations make up 33%. These distributions indicate a strong preference for CNN-based feature extraction when paired with both LSTM and Dense ANN models

A.8 Optimisers

Optimisation is central to many ML approaches. These algorithms revolve around minimising or maximising certain functions to improve model performance (Ruye, 2025). In neural network-based models, optimisers play the role in training the models by converging towards optimal parameters, while balancing trade-offs such as stability and efficiency (Nwankpa, 2020).

22 studies explicitly mentioned the use of optimisers. In Fig. 12. the proportion of optimisers used in the selected articles is presented. Based on the findings, it is evident that Adam Optimiser is the most used optimiser, used in 62% of the relevant papers. This is followed by Stochastic Gradient Descent (SGD) which was used in 15% of the applicable studies

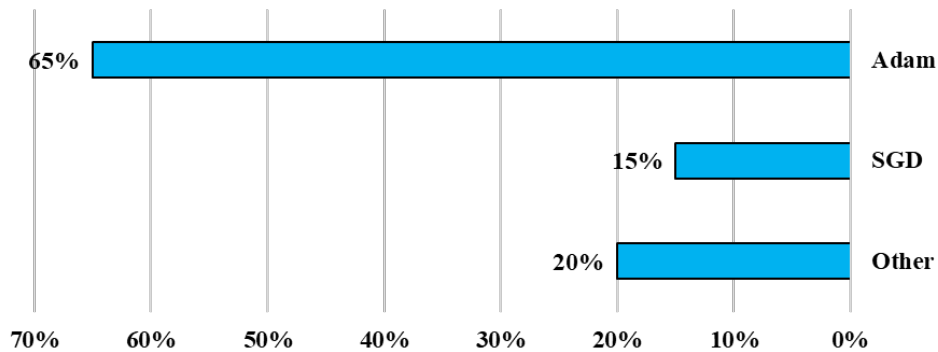


Figure A.12: Distribution of optimiser algorithms found in the literature.

A.8.1 Adam optimiser

Adam optimiser proposed by Kingma and Ba (2014) is a stochastic optimisation approach that relies only on first-order gradients and has low memory requirements. It involves calculating individual adaptive learning rates for different parameters based on estimates of the first and second moments of the gradients (Kingma and Ba, 2014). The Adam optimiser combines the parts of RMSProp and AdaGrad (two well-known optimisers) to estimate the mean (first moment) and uncentered variance (second moment) of the gradients (Nwankpa, 2020).

Adam optimiser emerged as the most used optimiser for training in the studies. Hu et al.

(2023) opted for its use because of its ability to independently adapt the learning rates for different parameters by analysing both the first and second-moment estimations of the gradient. This ensures convergence stability without the requirement of storing global gradients making it particularly useful for large datasets. Kumar Sharma et al. (2022) and Zhao and Zhang (2024) employed the Adam optimisation algorithm in conjunction with early stopping and dropout regularisation to mitigate overfitting and enhance the generalisation capability of their models. Gao et al. (2023) used a unique dropout approach that involved stochastic forward passes, using Adam in conjunction with Monte Carlo dropout to estimate uncertainty during training. Ong et al. (2022) applied Adam in their Proximal Policy Optimisation (PPO) model, integrated within a PPO-LSTM architecture. This combination addressed challenges such as the retrieval of older state-action sequence information due to varying equipment degradation rates. The incorporation of an LSTM layer enhanced the optimisation process by enabling Adam to more effectively capture the temporal dependencies. Rivas et al. (2022) combined Adam with Bayes by backprop (a variational inference algorithm) to update the distribution parameters of a Bayesian Neural Network. Adam specifically functioned to update the distribution parameters for the weights and biases in the network and Bayes by backprop applied the variational inference. Liu et al. (2021) used Adam with a regression loss. The main optimisation objective was to minimise the source regression loss which measured the accuracy of the RUL predictions. Notably, all the recent studies that used Adam used MSE as the loss function during optimisation including, Wang et al. (2024), He et al. (2024), Kim et al. (2024), and Zhao and Zhang (2024). Other works that utilised Adam include Winkel et al. (2023), De Santo et al. (2022), Gong et al. (2022), and Aydemir and Paynabar (2020).

Two studies used other variants of Adam. Rectified Adam was used by Zhang et al. (2022), which incorporated a rectification mechanism to stabilise the adaptive learning rate. Hsu et al. (2022) used Nadam (Nesterov-accelerated Adaptive Moment Estimation). Their study favoured Nadam due to its faster convergence in loss compared to Adam and RMSprop during experimentation. The authors argue that in some cases the integration of Nesterov momentum in Nadam allows for quicker and more stable optimisation compared to the standard Adam optimiser.

A.8.2 Stochastic gradient descent

SGD builds on the gradient descent algorithm. However, instead of using the entire dataset to compute gradients and update model parameters, SGD, also referred to as random GD, randomly picks small samples (Tschöpe, 2019). This significantly reduces computational requirements and improves performance on large datasets compared to more advanced methods that use detailed second-order calculations (Ruye, 2025).

Fathi et al. (2021) was the only study to use non-random GD. They used a GD-based learn-

ing algorithm to adjust the delta robot's system parameters during motion by minimising the tracking error function. Lu et al. (2021) used SGD as their optimiser to minimise the MSE of the generative (LSTM) and discriminative (autoencoder) in their proposed adversarial learning architecture. The MSE was also used as a loss function in the study by Cheng et al. (2021) where SGD was used to optimise both the weight coefficients and biases in their ensemble LSTM model, with parallel training of multiple networks. Zhuang et al. (2024) used SGD to optimise the cross-entropy loss for local manifold distributions in their graph-embedded subdomain adaptation network.

A.9 Hyperparameter tuning techniques

The next critical step in training an neural network is hyperparameter tuning. Unlike internal parameters learned from data during training, hyperparameters are external configurations which effects how an algorithm behaves during the training process. The majority of ML algorithms are set up by a collection of hyperparameters, the values of which frequently have a significant impact on performance and must be carefully selected (Bischi et al., 2023). Some of these can include the number of layers of the model, type of activation functions used, batch sizes of data, optimisers used, and learning rates, among others. Essentially, hyperparameters are settings or configurations that are determined before training begins and remain constant throughout the training process (Arnold et al., 2024). In Fig. 13. the proportion of different hyperparameter tuning techniques used in the selected studies is presented.

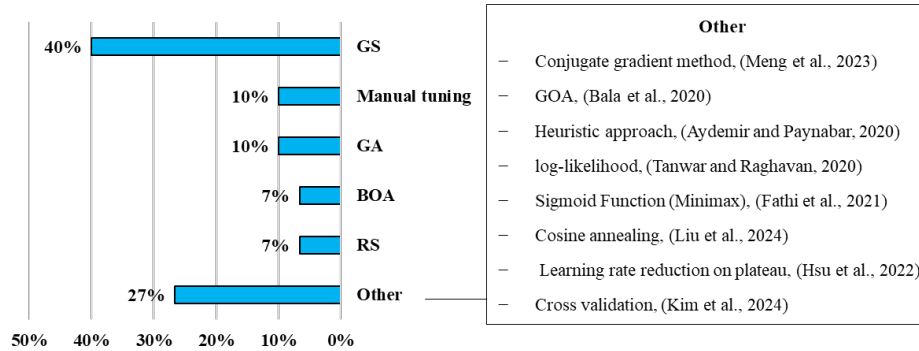


Figure A.13: of algorithms/techniques found in the literature for hyperparameter optimisation.

A.9.1 Grid Search

GS involves a thorough search over a manually selected subset of the hyperparameter space used by the ML model (Belete and Huchaiah, 2022). It works by dividing the range of each parameter into smaller intervals, calculating the target value, which is usually

an error, for every parameter combination, and selecting the best one to minimise the target value (Hu et al., 2023). Belete and Huchaiah (2022) contend that this method’s ease of use, parallelisation capabilities, and resilience in low-dimensional areas make it a good approach for hypermeter optimisation. However, traditional GS methods are time-consuming which limits its application involving large datasets (Beltrami and Da Silva, 2020).

Despite the limitations of GS, recent studies have still effectively used it to optimise the hyperparameters of their models. For example, Hu et al. (2023) used GS to optimise multiple hyperparameters of their deep bidirectional RNN ensemble. These included the number of hidden cell layers and neurons, as well as parameters related to the regression Decision Tree (DT) in the ensemble layer. Gao et al. (2023) utilised GS to optimise hyperparameters in their dual-channel feature attention-based approach, including the number of epochs and batch sizes. The study by He et al. (2024) adopted GS for optimizing hyperparameters in their cascaded multiscale convolutional neural network (CMS-CNN) framework. The GS was used to fine-tune hyperparameters such as the number of CMS-CNN modules and the size of convolution kernels. Zhao and Zhang (2024) applied GS to systematically search for the best combination of hyperparameters in their hybrid-based model. In the research done by Zhuang et al. (2024), GS was used to optimise sensitive hyperparameters, specifically the trade-off coefficients in their graph-embedded subdomain adaptation network model. Other studies that used GS include Costello et al. (2017), Khelif et al. (2017), Cheng et al. (2021), De Santo et al. (2022), Rivas et al. (2022), and Zhang et al. (2022).

A.9.2 Genetic Algorithm

Conversely, GAs are particularly suited for high-dimensional hyperparameter spaces (Mehdary et al., 2024). GAs are randomised optimisation techniques inspired by natural selection and evolutionary processes (Beg and Islam, 2016). Another notable advantage GAs have over local methods (such as GS) is that they do not remain trapped in local maximums or minimum (Aszemi and Dominic, 2019).

Three studies in the selected works have used GAs to optimise their model’s hyperparameters. Catelani et al. (2021) used a GA for their ESN model. It specifically optimised parameters such as reservoir size, spectral radius and leaky rate. Kumar Sharma et al. (2022) utilised it to tune the hyperparameters of their LSTM network. The search space was defined with options for parameters such as number of units, dropout rates, batch sizes and activation functions. Ince and Genc (2023) used GAs to optimise their joint autoencoder-regressor deep neural network model. The parameters included window size, learning rate, number of filters in convolutional layers, kernel size, activation functions, and LSTM units. All the studies followed the same evolutionary process in their GA

approach, involving population initialisation, crossover, mutation, and elitist selection.

A.9.3 Bayesian optimisation algorithm

According to Wang et al. (2024), the Bayesian optimisation algorithm (BOA) is a method that has been proven to be effective for hyperparameter optimisation. This approach involves combining the GPR model with an acquisition function (Wang et al., 2024). The GPR model estimates posterior probability distributions based on observed data, while the acquisition function selects the next point to evaluate.

Wang et al. (2024) used Bayesian Optimisation (BO) to tune hyperparameters such as the number of neurons in the CNN and Bi-LSTM layers, as well as the dropout rate, activation function, and learning rate in their deep learning ensemble model. In the research done by C. Liu et al. (2021), BOA was utilised to fine-tune hyperparameters such as the learning rate, number of convolution filters, and kernel sizes in their dual attention-based TCN model. They selected those parameters by utilising cross-validation with a tree Parzen estimator-based algorithm using the HyperOpt library.

A.9.4 Random Search

Random Search (RS) is an optimisation algorithm that works by randomly selecting a predefined range of hyperparameter values. The performance of the randomly chosen combination of hyperparameter values is then evaluated and the best is selected. This is a more efficient way of searching for hyperparameters compared to more brute-forced methods such as GS as it does not search through all possible combinations.

Aremu et al. (2019) utilised RS to optimise the hyperparameters of their Relative Entropy Weibull-SAX framework. The RS method was applied to fine-tune the parameters of a Random Forest Regression (RFR) model, which was trained using a standard 70%-30% split for training and testing, respectively. Winkel et al. (2023) used RS to optimise their MLP-based prediction model. They conducted 20 random draws to tune parameters such as the number of neurons per layer, activation functions and dropout rates. The hyperparameter values were sampled from a uniform probability distribution and to evaluate the effectiveness of each randomly selected combination of hyperparameters, the accuracy of the model during the first training iteration was used as the performance metric.

A.9.5 Manual Tuning

In some cases, the hyperparameter space is small and/or researchers have domain expertise or intuition about the necessary hyperparameters to use for their models. In these cases,

they may opt for manual hyperparameter tuning which involves systematically experimenting with different hyperparameter configurations. Zhao et al. (2018) used manual tuning to adjust the hyperparameters of their local feature-based GRU network. They systematically experimented with various hyperparameters such as the learning rate, the number of recurrent layers, and the number of hidden units and validated their approach with different datasets. In the research by Ong et al. (2022), manual tuning was performed on their deep reinforcement learning-based model. They experimented with different learning rates, LSTM layers and optimiser settings.

A.10 Performance Evaluation Metrics

After PdM models are trained and optimised, they are typically evaluated using a set of metrics. These metrics are useful for comparing less-optimised versions of the same approach, internal architectural variants, and sometimes alternative models reported in external literature. However, most of the reviewed papers primarily aim to demonstrate the performance of an end-to-end pipeline on a selected dataset. As a result, performance numbers are not always directly comparable across studies, because differences in preprocessing, feature/window construction, label definitions, tuning budgets, and evaluation protocols can materially affect the reported metrics (Kapoor et al., 2024). Furthermore, across the reviewed papers, key protocol details (e.g., split approach, window construction, and target definition such as RUL capping) are not always stated explicitly, which further limits the comparability of reported metric values.

Within the reviewed studies, comparisons are therefore commonly made between variants within a single modelling family (e.g., alternative neural architectures within deep learning), rather than as broad cross-family benchmarks. For instance, Kumar Sharma et al. (2022) focuses on comparing alternative deep-learning configurations of an LSTM-based architecture, and Gong et al. (2022) compares several neural sub-architectures within the deep-learning family. However, cross-family comparisons are valuable because, when they are run on a common testbed under the same experimental protocol, they support fair comparisons and help attribute performance differences to the algorithms rather than inconsistent evaluation setups (Packer et al., 2018). Studies such as Hsu et al. (2022) cross-evaluated their model across a wider range of model families, spanning tree-based methods (e.g., Random Forest, XGBoost) and deep-learning models (e.g., LSTM and CNN).

This section synthesises the evaluation metrics used across the selected studies. These metrics align with either regression-based or classification-based tasks. RUL prediction is traditionally treated as a regression task because RUL is a continuous quantity. However, some studies instead formulate the task as a classification problem or rely on classification metrics to define distinct health states of equipment. Figure A.14 shows the proportion of studies that used classification metrics or regression metrics.

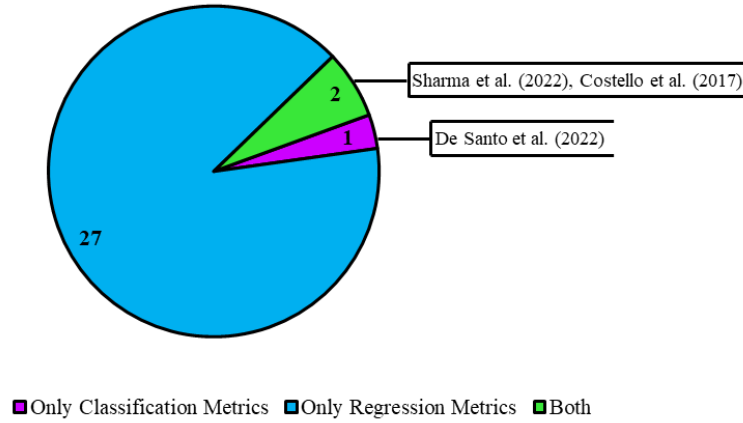


Figure A.14: Proportion of studies that used classification metrics or regression metrics.

Only 3 studies made use of classification metrics, with one focusing purely on a classification task. Costello et al. (2017) used a test accuracy metric to evaluate their L2 and L1 regularised SVM classifier models in classifying label states (Online, Upper, Lower). Additionally, a cross-validation (CV) score was used to evaluate the generalisation performance of the classifiers by averaging the models' performances during training. The classified states were then used as inputs to a logistic regression (LR) model which predicted the RUL. The RUL prediction performance was evaluated using Pearson's correlation coefficient which served as a regression-type metric that measured the relationship between the estimated state membership probabilities and the actual RUL. In the research done by Sharma (2025), their TCN-based models, which predicted the RUL, were evaluated using commonly used RUL regression metrics including root mean square error (RMSE), mean square error (MSE), mean absolute error (MAE) and a scoring function (SF). The study further focused on classifying failure events within specific time windows. To do this, binary classification was used to predict whether the engine would fail within the next defined number of cycles. Additionally, multi-label classification was performed where each label corresponds to different time intervals until failure. To evaluate how the classifier correctly identified the labels, metrics such as accuracy, precision, recall and F1-score were used. De Santo et al. (2022) reframed the RUL estimation, traditionally a regression problem, as a classification task. This enabled them to use classification metrics such as accuracy, precision and recall. Specifically, they defined multiple health statuses for hard drives as discrete levels based on proximity to failure, rather than predicting an exact RUL value. They did this by using a regression tree (RT) to divide the health statuses into different classes, such as good, very fair, fair, soft warning, warning and red alert, based on how close the hard drives were to failure. Accuracy measured the overall correctness in classifying health states, while precision focused on the accuracy of failure predictions, and recall assessed the model's ability to identify all failing drives. Moreover, De Santo et al. (2022) utilised a failure detection rate (FDR) and a false alarm rate (FAR) met-

ric. FDR quantified the proportion of correctly classified actual failed sequences. While FAR indicated the frequency with which the model triggered alerts that were not actually needed.

The other 27 studies purely made use of regression-related metrics to evaluate their regression models' RUL prediction performance. The works made use of a variety of different evaluation metrics. To organise them, they were categorised in the following areas: accuracy/error, computational efficiency, reliability, fit and generalisation. Figure A.15 presents the distribution of evaluation metrics used in the literature

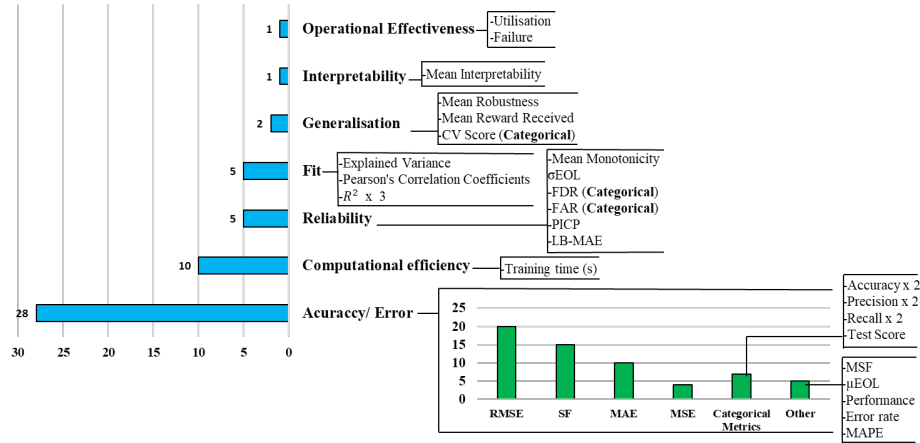


Figure A.15: Distribution of evaluation metrics used for RUL prediction models.

A.10.1 Accuracy/ Error metrics

28 out of the 30 studies used at least one accuracy/error performance metric. Accuracy includes metrics that determine how close the models' predictions are to the actual outcomes or labels. Various metrics were used for this including several point prediction metrics such as root mean squared error (RMSE) and mean absolute error (MAE). Where RMSE was the most common, it was utilised by 20 of the studies, and 4 studies made use of the non-squared (MSE) variant. Moreover, the MAE was used by 10 of the studies. Other commonly used accuracy-related metrics include the use of score function (SF), which was used by 15 studies, and categorical metrics. To assess the accuracy of models, the following SF was used by the works (taken from Gao et al. (2023)):

$$\text{Score} = \begin{cases} \sum_{j=1}^n \left(e^{\frac{h}{a_1}} - 1 \right), & h \geq 0, \\ \sum_{j=1}^n \left(e^{\frac{h}{a_2}} - 1 \right), & h < 0, \end{cases} \quad h = \hat{y}_j - y_j \quad (\text{A.1})$$

Where $\hat{y}_j$ is the predicted RUL, and what y_j is the actual RUL of the j th sample, and a_1 and a_2 are constants that control the sensitivity of the score depending on whether it h is positive or negative. In other words, the scoring function (SF) weights the impact of under- and over-predictions, where smaller errors yield smaller score values, indicating better model performance.

There were several other unique accuracy/error-related metrics used by the studies. Cheng et al. (2021) used a mean score function, which is like SF, where it applies different weights to overestimations and underestimations. However, the MSF averages individual scores and overall predictions. Khelif et al. (2017) made use of a unique metric referred to as a performance metric. Specifically, it counts how many predictions fall within a defined acceptable range. If the prediction error is within the specified range, the prediction is considered correct. This allows for a binary assessment of the model's performance as opposed to the SF that outputs a continuous penalty based on how far off the prediction is. The Mean Absolute Percentage Error (MAPE) metric was used by Aydemir and Paynabar (2020). This metric measures the average of the absolute percentage differences between the actual and predicted values. Intrinsically, it puts equal weight on all errors, regardless of the magnitude of over- or underestimations. Meng et al. (2023) used a Mean End of Life Cycle Prediction (EOL) metric which calculates the average predicted End of Life (EOL) cycle across multiple test instances and compares it to the actual observed EOL cycles. In Zhao and Zhang (2024), prediction accuracy was evaluated using a percentage error rate (relative deviation between predicted and true RUL), which normalises errors across different RUL magnitudes. Furthermore, they implemented a scoring system based on this metric and introduced a weighting mechanism to account for the significance of under- and over-predictions to assess the practicality of their model.

A.10.2 Computational Efficiency

Training time is the second most used metric (used by 10 studies). This directly assesses the computational efficiency of the model. A model that can quickly be trained is desired in real-time environments when models need to be frequently retrained and deployed (Prapas et al., 2021). Additionally, training time is important to measure since a model with a shorter training time can scale better with larger datasets (Prapas et al., 2021). While other efficiency metrics can also be reported, such as “inference accuracy, inference time, throughput, memory usage, and power consumption” (Ratul et al., 2025), training time was the only computational-efficiency metric used in the selected studies for this category.

A.10.3 Reliability Metrics

Reliability metrics focus on evaluating the stability and consistency of the model. 4 studies incorporated reliability-related metrics. Meng et al. (2023) used an SD of EOL Cycle (EOL) metric. The metric functioned to assess the model's ability to consistently predict the EOL predictions by measuring the indicated variability/spread of the predictions. Rivas et al. (2022) used LB-MAE (Lower Bound Mean Absolute Error) as a metric. LB-MAE assesses the difference between the mean prediction and the lower bound for uncertainty. This enables the assessment of how conservative the model was in predicting the time that remains before a failure occurs. A recent study by He et al. (2024) used prediction interval coverage probability (PICP), a metric that quantifies the proportion of predictions where the true RUL falls within the predicted interval, given a specified confidence level. Catelani et al. (2021) used mean monotonicity to measure the consistency of the trend of the predictions over time. A high score indicated that the predicted values decrease consistently as the system (in this case lithium-ion batteries) approaches its EOL. This helps see if the prediction follows the expected real-world smooth downward trend and identifies artificial/unreliable prediction fluctuations.

A.10.4 Fit Metrics

The evaluation of the fit assesses how well the model explains the variance between the models' outputs and the actual data. Catelani et al. (2021), Sharma (2025), and Zhuang et al. (2024) used R^2 the coefficient of determination. This metric measures how much of the variance in the RUL variable is explained by the predictions of the model. The closer the R^2 value, the better the indication is that the model is capturing the variation of the data. Fathi et al. (2021) used a similar metric, explained variance, which also measures how much of the data's variability is captured by the model. However, explained variance does this without directly comparing and penalising it against a baseline (mean prediction) like R^2 . Costello et al. (2017) took a different approach by opting for Pearson's correlation coefficients to evaluate the strength of the linear relationship between the predicted RUL and actual RUL. This approach focused on the correlation between the two variables rather than the variance explained by the model.

A.10.5 Generalisation Metrics

It is vital for models to generalise well, meaning they perform well on unseen data. Generalisation metrics evaluate how well a model can maintain its accuracy on new unseen data and ensure that the model will consistently make accurate predictions on data beyond the sets it was trained on. Costello et al. (2017) utilised the Cross-Validation (CV) Score as a metric for generalisation. This method involved partitioning the dataset into multiple

subsets. The model is then trained on some of these subsets and tested on the remaining ones. By tracking performance across all iterations, the final CV score could be obtained. Ong et al. (2022) applied the mean reward received to evaluate their deep reinforcement learning-based model. The reward function was defined as part of the agent's policy optimisation process. Specifically, the rewards were accumulated for actions that resulted in successful maintenance actions, and penalties were accumulated for failed actions. The average cumulative performance allowed the authors to assess how well the trained policy performed across a variety of scenarios that were not encountered during training.

Notably, out of all the reviewed works, Tanwar and Raghavan (2020) was the only study to evaluate their models' performance across multiple reduced training data fractions. Specifically they trained on 20%, 40%, 60% and 80% fractions of lubricant condition monitoring (LCM) time-series data, where degradation is represented by wear debris concentration (WDC, ppm) over operating time. This enabled the quantification of how sensitive the model performance was to limited data and demonstrated the effectiveness of their approach even under limited data experiments. Although the reduced-data analysis was not used for cross-family benchmarking, the proposed multi-output Gaussian process regression (GPR) model was found to outperform the baseline GPR variants across the training set fractions.

A.10.6 Interpretability and Operational Effectiveness Metrics

Interpretability and operational effectiveness-based metrics were each used by only one study in this review. Catelani et al. (2021) used a mean interpretability metric in addition to the more common accuracy, reliability, fit, and generalisation metrics as previously discussed. Specifically, their model could be interpreted in terms of representing the battery's health stages. A high score indicated that the model can effectively divide the system degradation into stages that are meaningful and easy to understand. Winkel et al. (2023) also used a unique metric that focused on operational effectiveness. They used two metrics which measured the utilisation and failures of the system. The utilisation metric measured the percentage of the lifetime of electromechanical relays that were utilised before maintenance was performed. This implies that a higher utilisation indicates that a relay was used for a larger fraction of its potential lifespan. A higher utilisation is preferable since this means that unnecessary maintenance actions are reduced. On the other hand, the failure metric measured the percentage of relays that would fail without prior maintenance. This metric was aimed to keep as low as possible since these failures potentially led to downtime and higher maintenance costs.

A.11 Future recommendations of studies

Future work sections in papers identify potential areas for improvement or further research in the field. Of the 30 studies 26 studies proposed a total of 46 future works. The future works were categorised into eight different groups based on the nature of the research focus. The eight categories made up 64% of the mentioned future works and included subdomain adaptation, application to new domains, hyperparameter tuning, model uncertainty, practical application, transfer learning, anomaly detection, and equipment-specific research. Figure A.16 shows the distribution of the proposed future works

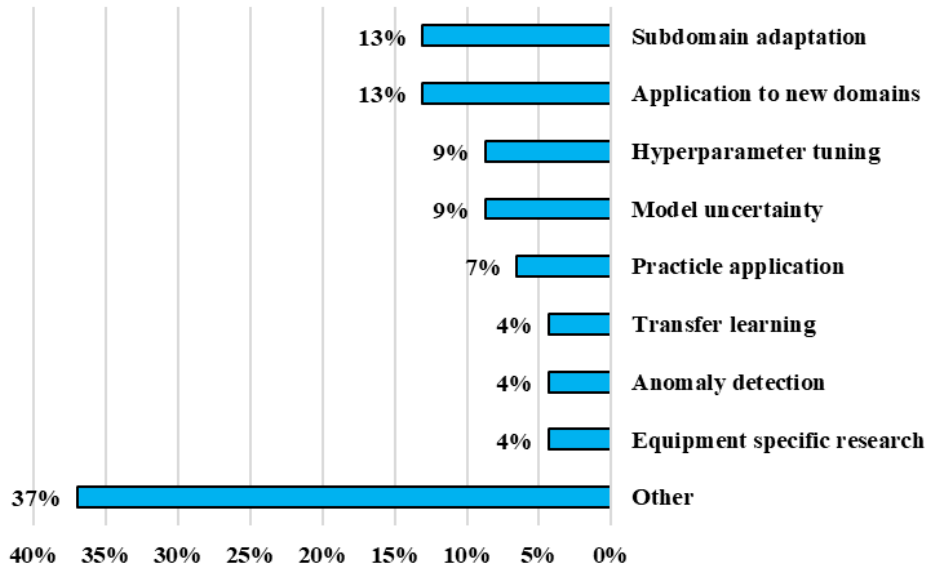


Figure A.16: Distribution of the proposed future works.

A.11.1 Subdomain Adaptation

Subdomain adaptation was one of the most proposed future works, mentioned in 13% of the literature. It is concerned with adapting models within the same domain but under varying conditions, datasets, or environments. In the future work proposed by Costello et al. (2017), for example, the development of a general model for gas circulator refuelling vibration response, and broader circulator data, including both normal and degradation behaviour, are suggested. They stress that although thorough testing and cross-validation were conducted, the proposed classification models used in the study were restricted to a specific machine example. Tanwar and Raghavan (2020), focused on predicting the RUL of lubricating oil and recommended that future work should account for other factors such as the effect of imperfect maintenance, loss of oil, operating conditions, and environmental changes, among others. Rivas et al. (2022) used the FD001 and FD003 datasets (CMAPSS datasets containing one operating condition) but highlighted that future work

should focus on expanding their model to datasets that present multiple operating conditions such as FD002 and FD004. Hu et al. (2023) suggested investigating distributed adaptive methods that consider the limited historical monitoring data across various operating conditions within the C- MAPSS dataset further. They argue that this will allow the proposed method to be adaptively adjusted, enhancing its ability to generalise across different application scenarios. He et al. (2024) suggested incorporating working condition information into the RUL prediction task to better quantify bearing RUL uncertainty across different working conditions. Zhuang et al. (2024) proposed investigating an adversarial subdomain adaptation approach for machine health monitoring to address and eliminate discrepancies between subdomains by utilising adversarial learning.

A.11.2 Application to New Domains

The other most proposed future work, also mentioned in 13% of the future works, is referred to as application to new domains. While subdomain adaptation focuses on applying the model to data sourced from the same equipment type (however under different operating conditions) this category suggests applying/adapting existing models or techniques to entirely different domains. For example, Gao et al. (2023) propose the development of a cloud platform that incorporates their dual-channel feature attention-based approach for RUL detection that can generalise well to various types of engines. In addition, Tanwar and Raghavan (2020) suggest that future research should apply their approach to real condition monitoring datasets from various other industrial systems to further validate and confirm the robustness of their model. Ince and Genc (2023) suggested that their joint autoencoder-regression model approach should be applied to other domains that benefit from time-series data. Furthermore, Cheng et al. (2021) recommended further investigation of their ensemble LSTM network architectures for different equipment with multi-sensor monitoring. Kim et al. (2024) suggested exploring how their transformer architecture could be improved to generalise to other domains, including electronics and industrial equipment.

A.11.3 Hyperparameter tuning

In 9% of the future works, the expansion on optimising the hyperparameters of the proposed RUL prediction models, was proposed. Khelif et al. (2017) suggested that future work should focus on improving their SVR model by exploring hyperparameter tuning to analytically determine the parameters that yield the most accurate RUL estimate. Aydemir and Paynabar (2020) argued that their approach is fit for employment in industry settings due to their model's ease of implementation and rational mathematical reasoning. However, they acknowledge that an improved systematic process for optimising network parameters, such as kernel sizes, number of layers, and neurons, is required.

Kumar Sharma et al. (2022) recommended integrating more novel optimisation methods with predictive models to improve hyperparameter search and develop resource-intensive methods. Rivas et al. (2022) suggest the exploration of different hyperparameter tuning algorithms such as the GA or simulated annealing to find the optimal hyperparameters of their approach.

A.11.4 Model Uncertainty

Addressing the uncertainty of a model is vital for verifying or improving the generalisation and reliability of RUL prediction models. Several studies (making up 9% of the proposed future works) recognised the need to incorporate uncertainty estimation into their models. Hsu et al. (2022) recognise that uncertainties arising from input data, model structure, parameters, and data calibration can undermine the reliability of RUL point predictions. Consequently, they suggest that RUL interval estimation should be explored to provide better support for maintenance decisions. He et al. (2024) recognises that their proposed method assumes that the RUL follows a normal distribution. In future work, they plan to relax this assumption and develop a new uncertainty quantification loss function to address the limitation. Kim et al. (2024) proposed integrating uncertainty modelling into both the training and prediction phases of their model since it will offer several benefits to their model. They add that by incorporating uncertainty estimation techniques, the model could generate a predictive distribution rather than a single RUL point estimate which would make more informative and reliable predictions. Liu et al. (2024) assert that addressing the uncertainty in RUL prediction is a valuable area of exploration to enhance the practical application of models in engineering.

A.11.5 Practical Application

Practical application refers to future work that addresses the need to apply the models in real-world environments. This category was also mentioned in 7% of the future works. Meng et al. (2023) suggest exploring how to integrate EOL prediction results into the design of battery energy management systems. De Santo et al. (2022) suggests exploring how data centre technicians and managers can effectively use the more detailed predictions generated by their method to improve long-term maintenance planning. Rivas et al. (2022) propose creating a PdM framework that can detect anomalies and be integrated with the reactor training simulator at the North Carolina State University nuclear simulation lab. This approach would showcase the framework's effectiveness in maintaining a full reactor system.

A.11.6 Additional Future Works

Other suggested future works, each mentioned in 4% of the literature, included transfer learning, anomaly detection and equipment-specific research. Liu et al. (2021) suggests examining more efficient transfer learning approaches that can solve the fault prognosis issue across diverse operating conditions. Ong et al. (2022) seek to investigate how transfer learning techniques could be applied to increase the learning efficiency of their proposed model. The anomaly detection category focused on suggested methods to identify unusual patterns or deviations in system behaviour that may indicate faults or failures. Kumar Sharma et al. (2022) seeks to investigate anomalies and injected out-of-distribution data to address false triggers in ML-based models. Ince and Genc (2023) suggest the investigation of incorporating metric learning into the fault detection algorithm for RUL labelling based on fault detection. Equipment-specific research focuses on researching the characteristics of the data extracted from the equipment to make improvements to the RUL prediction frameworks. Wang et al. (2024) state that their future research will focus on PdM scheduling for aircraft engines, with particular emphasis on examining how aircraft engine mission cycles influence maintenance scheduling. Zhao and Zhang (2024) recommended studying the degradation characteristics of rolling bearings to better define their health condition, which would enhance the accuracy of prediction results.

A.12 Discussion

The research presented an SLR that investigated published literature in the domain of RUL prediction for PdM. The review aimed to answer 11 questions (refer to [section A.2](#)). To filter and identify the relevant studies that serve to answer these questions, the PRISMA 2020 methodology was used. A total of 30 studies were identified to be suitable for the review.

Q1 was answered through [subsection A.3.3](#) where [Figure A.4](#) showed that the number of publications increased over the years indicating a growing interest in the field of RUL prediction. Section 3.1 answered Q2 and Q3 by presenting the geographical and citation distribution of selected works. It was found that RUL prediction for PdM is a global interest, finding contributions from 14 countries, with China ranking the highest, contributing 14 out of the 30 publications. Zhao et al. (2018) and Khelif et al. (2017) had considerably more citations than the rest of the publications, with 808 and 414 citations respectively. Most of the studies had citation counts that fall between 20 and 40.

The common challenges and limitations identified in PdM (Q4) were investigated in [section A.4](#). Two prevalent limitations have appeared repeatedly across the reviewed PdM/RUL studies. First, many works point to a data and labelling constraints where

labelled run-to-failure histories are scarce, labels can be uncertain, and datasets are often disproportionately abundant with "healthy" operation data with relatively few failure-progression samples. Second, the literature frequently raises generalisation as a challenge, since changing operating conditions and unit-to-unit variability can introduce domain shift and reduce performance outside the training distribution.

In [section A.5](#) the types of data used in the selected works were discussed to answer Q5. Most of the datasets used in the studies were synthetic (55%). However, the use of synthetic data was largely homogeneous, with all but one study utilising the C-MAPSS turbofan engine dataset. In contrast, the datasets sourced from real-world environments were diverse.

To answer Q6, [section A.6](#) investigated the FE techniques used by the selected research papers. The section identified some of the most frequently used algorithms including CNNs and TCNs making up 16% and 11% of the FE techniques used respectively. Notably, the use of FE techniques was quite varied, with 51% of the techniques being unique (appearing in only one study). This indicates that there is no standardised approach for applying these techniques in the field. Possibly suggesting that FE is quite task-specific and requires custom approaches for specific problems and datasets. However, this also highlights a potential research area to explore more generalised FE approaches.

Q7 involved investigating which prediction models were used in the studies. [Figure A.10](#) (refer to [section A.7](#)) presents the distribution of algorithms and techniques found in the literature for RUL prediction. The results indicate that the range of algorithms used for prediction is not as diverse, with only 23% of studies using unique algorithms. The most widely utilised were found to be LSTM (23%), dense feedforward networks (23%), and GPR (10

Q8 focused on identifying the optimisers used by the studies. In [section A.8](#) the distribution of optimiser algorithms found in the literature was presented [Figure A.12](#). The Adam optimiser was the most favoured in the studies, being used 65% of the time, followed by SGD, which was used in 15% of the cases. These findings suggest that Adam is a versatile and reliable choice for RUL-related neural network training. The SLR also investigated the hyperparameter optimisation techniques used by the studies to answer Q9 (refer to [section A.9](#)). The GS method is found to be the most popular (40%) despite its exhaustive and resource-intensive approach. This was followed by GA and manual hyperparameter tuning, each appearing in 10% of the studies.

Q10, which examines the performance evaluation metrics used across the studies, was addressed in [section A.10](#). The results show that 28 out of 30 studies used accuracy/error as the primary metric to evaluate their models' performance. RMSE stood out as the most used metric for this evaluation (used by 20 of the studies). Despite the prevalent use of accuracy metrics, the review reveals a notable underutilisation of other metrics. This in-

cludes those related to the evaluation of model efficiency (training time), which appeared in only 11 studies. Additionally, metrics assessing reliability and fit were featured in just 4 studies each, while those involving generalisation, interpretability, and operational effectiveness were used in 2 or fewer papers. The minimal use of these metrics suggests a gap in the comprehensive evaluation of RUL prediction models that extend beyond accuracy and ensure that models are robust, reliable and suitable for real-world applications. In addition, even when the same metric is reported (e.g., RMSE), values are not necessarily comparable across studies if key experiment and evaluation protocols differ or are not stated explicitly.

As highlighted, metric distribution needs to be interpreted in light of protocol sensitivity, even when two studies report the same metric (e.g., RMSE), values are not necessarily comparable if the underlying preprocessing, windowing, RUL capping, and evaluation procedure differ. Consequently, “state-of-the-art” comparisons based purely on published numbers can be suggestive rather than definitive unless baselines are re-implemented and evaluated under a shared protocol. This reinforces the value of controlled benchmarking studies that align the full experimental pipeline when comparing alternative approaches.

The last SLR research question, Q11, was analysed in [section A.11](#). Out of the 30 papers reviewed 26 studies proposed future research directions. Of these most suggested exploring new application domains and subdomain adaptation (13% each). This emphasises the importance and interest in researching the development of robust models that generalise well to intra- and inter-domain scenarios. Additionally, model uncertainty quantification and practical application have also been a commonly proposed area of investigation, both appearing in 9% and 7% of future works respectively. These findings further underscore the need for standardised evaluation metrics that fall beyond the scope of prediction accuracy, such as generalisation, operational effectiveness, uncertainty, and interpretability. Hyperparameter tuning techniques are also an area of interest, proposed by 9% of the studies. Other commonly suggested areas for future work include transfer learning (further substantiating the need for generalised approaches), anomaly detection and equipment-specific research.

A.13 Conclusion

From the results above, RUL prediction from PdM is a relatively new field, with the first publication among the selected works being in 2017. Among the selected studies, China accounted for 47% of the publications and included the most highly cited authors. The review highlighted the use of synthetic data, which were used in 55% of the publications compared to real-world data used in 45%. 94% of the synthetic data originated from the C-MAPSS dataset. This is also highlighted in the suggested future works where 7% of studies (where 50% of them used CMAPSS) suggest that practical application of the

models in real-world scenarios should be investigated. FE techniques were dominated by deep learning neural network techniques (specifically CNN, TCN and AE) used in 35% of the publications. However, 51% of the publications made use of unique FE techniques, suggesting the application of various FE techniques could be explored in more depth. The applied prediction models were a bit more balanced, with LSTM and feedforward networks utilised by 23% of the literature each. Notably, there was a strong preference for CNN-based feature extraction techniques, particularly in combination with dense-ANN and LSTM-based predictors.

The optimiser techniques used in the publications were, however, dominated by the Adam optimiser, making up 65% of the applicable works, and the SGD optimiser, another 15%. This suggests that the Adam optimiser is generally successful in PdM applications, however there is a gap in exploring alternatives. Furthermore, hyperparameter tuning techniques made use of the basic techniques like grid search, random search and manual tuning in 40%, 10% and 7% of the studies, respectively. However, 27% of other hyperparameter tuning techniques were also utilised and could make for interesting future comparisons. This trend was also picked up in the suggested future works, where 9% of the literature suggested investigating hyperparameter tuning in more detail.

As expected, most of the studies used regression-based metrics since RUL prediction is inherently a regression task. Only one study converted the problem into a classification task and only two studies used classification metrics to evaluate their classification health status models. Furthermore, accuracy/error-based metrics were found to be the most widely used type of metric, where only 2 studies didn't use it. Among these types of metrics, RMSE and SF have emerged as the prominent evaluation metrics, used by 20 and 15 studies respectively. Notably, other metrics such as interpretability, reliability and generalisation have been underutilised amongst the studies despite their importance in evaluating the ability for models to be used in real-world applications. Further, substantiation for the need for more extensive metrics can be seen in the proposed future works. Future recommendations of model uncertainty (9%) and practical application (7%) suggest the need for models to be evaluated in areas beyond the scope of accuracy. Additionally, those of subdomain adaptations (13%), application of new areas (13%) and other suggestions like the use of TL (4%) highlight the need for more generalised models that can be adapted to various datasets.

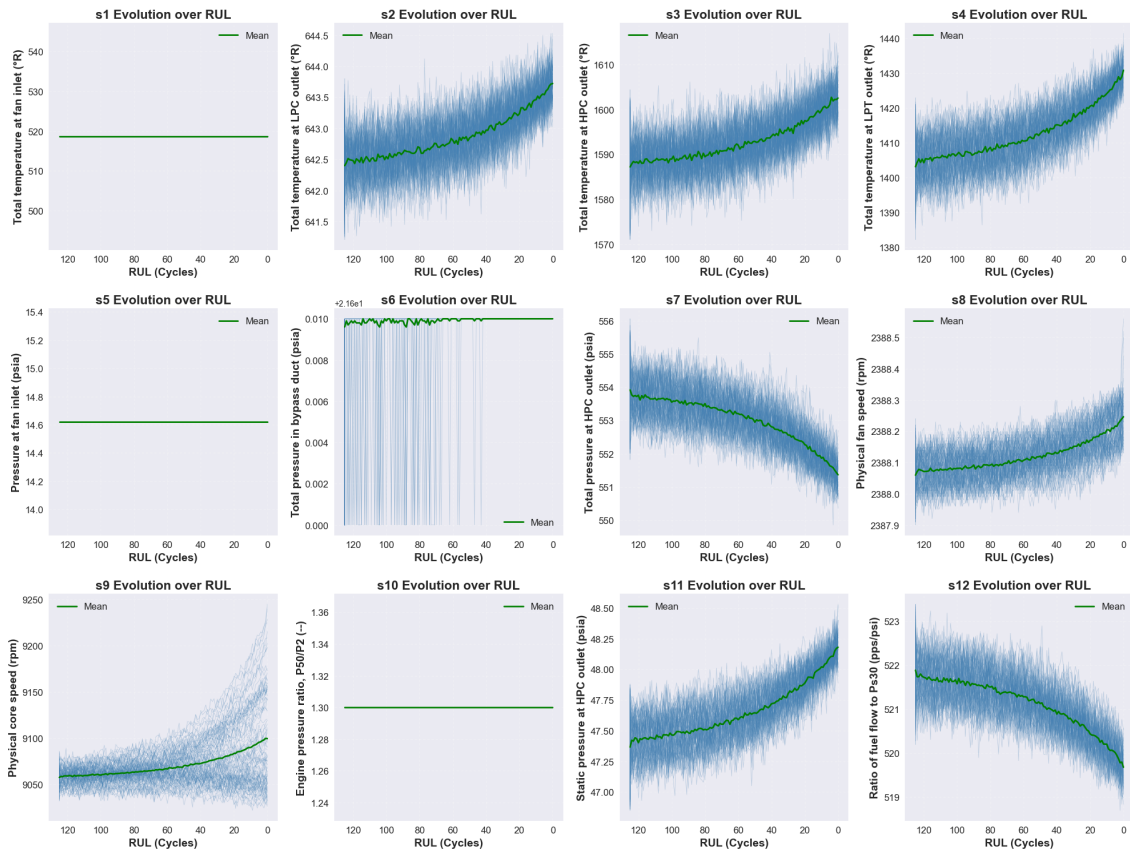
The reviewed literature consistently reports that practical PdM/RUL work is constrained by data realities. Specifically, labelled run-to-failure histories are often limited, labels can be uncertain, and datasets are frequently imbalanced toward healthy operation, sometimes compounded by sensor availability (İnce and Genc (2023), Fathi et al. (2021), Kim et al. (2024), He et al. (2024), and Gong et al. (2022)). In addition, equipment variability—changing operating conditions, unit-to-unit differences, and non-monotonic or multi-mode degradation—makes it difficult for a single model to generalise reliably across scenarios

(Wang et al. (2020), Cheng et al. (2021), Winkel et al. (2023), Tanwar and Raghavan (2020), and Liu et al. (2024)). These limitation themes recur across multiple studies and motivate continued attention to data availability, robustness, and generalisation in PdM research.

Appendix B

Exploratory Data Analysis

B.1 Evolution of the RUL (Cycles) of all CMAPSS FD001 sensors



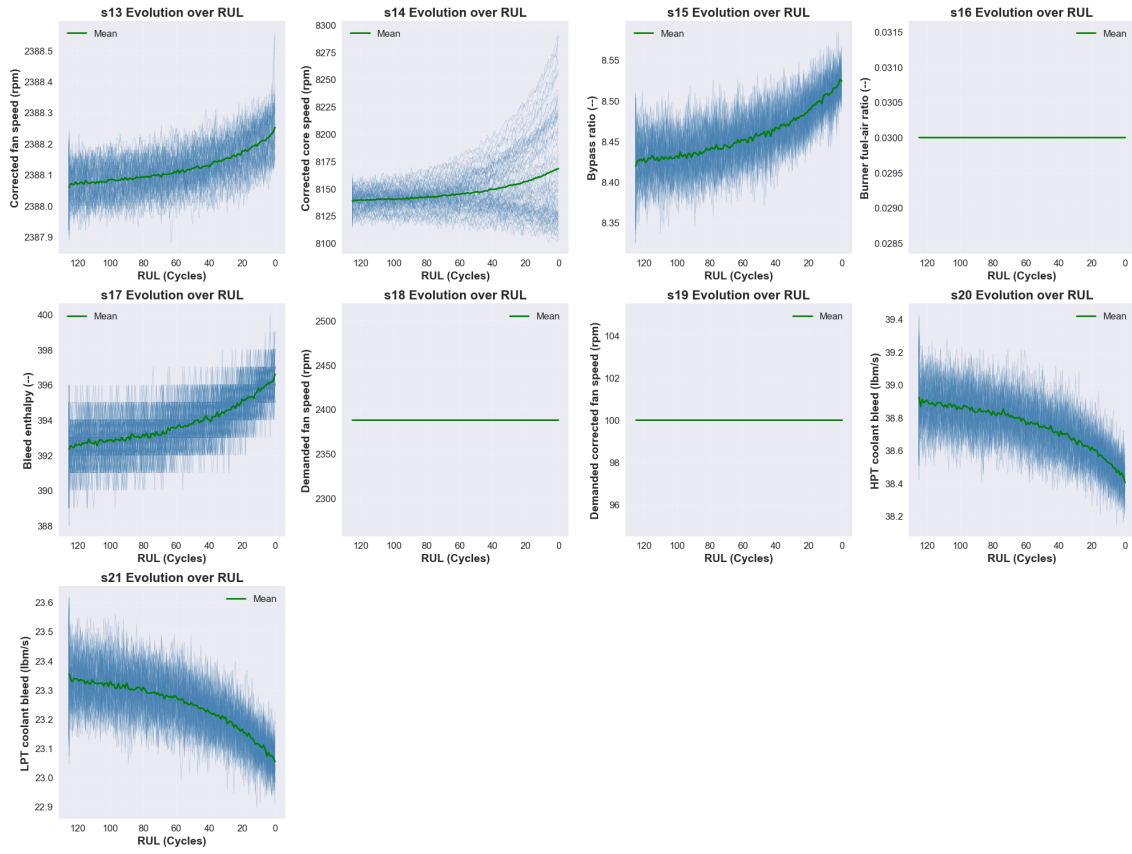


Figure B.1: Evolution of the RUL (Cycles) of all CMAPSS FD001 sensors.

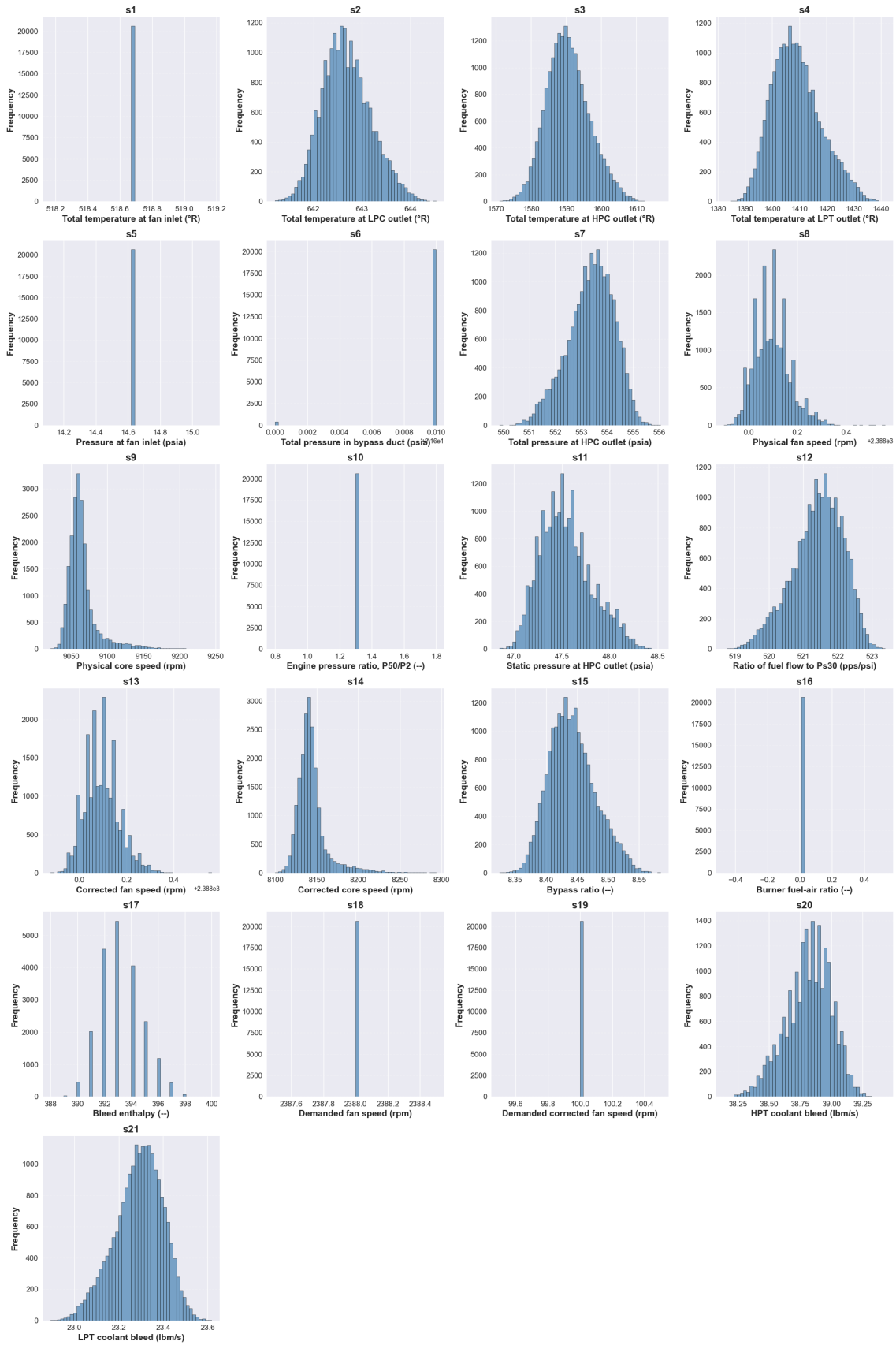


Figure B.2: Distributions of sensor features in C-MAPSS FD001.

Appendix C

Implementation and Validation

C.1 Hyperparameter search spaces used for the baseline models

Table C.1: Hyperparameter search spaces used for the baseline models.

Model	Hyperparameter	Search Space
MLR, Poly2, Poly3	–	Not tuned
Ridge, Lasso	alpha	$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2, 10^3\}$
ElasticNet	alpha	$\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$
	l1_ratio	$\{0.1, 0.3, 0.5, 0.7, 0.9\}$
Random Forest	n_estimators	[100, 1000]
	max_depth	[5, 20]
	min_samples_split	[2, 20]
	min_samples_leaf	[1, 10]
	max_features	{sqrt, log2, None}
XGBoost	n_estimators	[100, 1000]
	max_depth	[3, 8]
	learning_rate	$[10^{-2}, 3 \times 10^{-1}]$ (log-scale)
	subsample	[0.6, 1.0]
	colsample_bytree	[0.6, 1.0]
	min_child_weight	[1, 7]
	gamma	[0, 5]
LightGBM	n_estimators	[100, 1000]
	num_leaves	[20, 100]
	learning_rate	$[10^{-2}, 3 \times 10^{-1}]$ (log-scale)
	subsample	[0.6, 1.0]

Model	Hyperparameter	Search Space
	colsample_bytree	[0.6, 1.0]
	min_child_samples	[5, 50]
ANN	n_hidden_layers	{1, 2}
	units_layer1	{64, 128, 256}
	units_layer2	{32, 64, 128}
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}
RNN	n_recurrent_layers	{1, 2}
	units_layer1	{64, 128, 256}
	units_layer2	{32, 64, 128}
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}
LSTM	n_lstm_layers	{1, 2}
	units_layer1	{64, 128, 256}
	units_layer2	{32, 64, 128}
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}
GRU	n_gru_layers	{1, 2}
	units_layer1	{64, 128, 256}
	units_layer2	{32, 64, 128}
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}
BiLSTM	n_bilstm_layers	{1, 2}
	units_layer1	{64, 128, 256}
	units_layer2	{32, 64, 128}
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}
CNN	n_conv_layers	{2, 3}
	n_filters	{32, 64, 128}
	kernel_size	{3, 5}

Model	Hyperparameter	Search Space
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}
TCN	n_filters	{32, 64, 128}
	n_blocks	{2, 3}
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}
Transformer	num_transformer_blocks	{1, 2}
	d_model	{64, 128}
	num_heads	{2, 4, 8}
	ff_dim	{128, 256}
	dropout_rate	{0.1, 0.2, 0.3, 0.4, 0.5}
	learning_rate	$\{10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-2}, 5 \times 10^{-2}\}$
	batch_size	{32, 64, 128, 256}

C.2 Classical Regression Baseline Selected Hyperparameters (Grid Search)

Table C.2: Selected best hyperparameters for the classical regression baseline models.

Model	Selected Hyperparameters
Multilinear Regression (MLR)	n/a
Lasso Regression	$\alpha = 0.001$
Ridge Regression	$\alpha = 0.001$
ElasticNet	$\alpha = 0.001$ $\text{l1_ratio} = 0.9$

C.3 Selected Baseline Hyperparameters from Optuna

Table C.3: Selected best hyperparameters for the remaining baseline models.

Model	Selected Hyperparameters
Random Forest	<code>n_estimators = 500</code> <code>max_depth = 20</code> <code>min_samples_split = 14</code> <code>min_samples_leaf = 3</code> <code>max_features = sqrt</code>
XGBoost	<code>n_estimators = 966</code> <code>max_depth = 5</code> <code>learning_rate = 0.010171</code> <code>subsample = 0.890795</code> <code>colsample_bytree = 0.823410</code> <code>min_child_weight = 1</code> <code>gamma = 1.633698</code>
LightGBM	<code>n_estimators = 409</code> <code>num_leaves = 61</code> <code>learning_rate = 0.016729</code> <code>subsample = 0.778232</code> <code>colsample_bytree = 0.628778</code> <code>min_child_samples = 12</code>
ANN	<code>n_hidden_layers = 2</code> <code>units_layer1 = 256</code> <code>units_layer2 = 32</code> <code>dropout_rate = 0.2</code> <code>learning_rate = 0.01</code> <code>batch_size = 128</code>
RNN	<code>n_recurrent_layers = 1</code> <code>units_layer1 = 256</code> <code>units_layer2 = 128</code> <code>dropout_rate = 0.5</code> <code>learning_rate = 0.0001</code> <code>batch_size = 32</code>
LSTM	<code>n_lstm_layers = 1</code> <code>units_layer1 = 128</code> <code>units_layer2 = 64</code> <code>dropout_rate = 0.1</code> <code>learning_rate = 0.001</code> <code>batch_size = 32</code>
BiLSTM	<code>n_bilstm_layers = 1</code> <code>units_layer1 = 64</code> <code>units_layer2 = 32</code> <code>dropout_rate = 0.5</code> <code>learning_rate = 0.001</code> <code>batch_size = 64</code>

Model	Selected Hyperparameters
GRU	<code>n_gru_layers = 1</code> <code>units_layer1 = 256</code> <code>units_layer2 = 64</code> <code>dropout_rate = 0.3</code> <code>learning_rate = 0.0005</code> <code>batch_size = 32</code>
CNN	<code>n_conv_layers = 2</code> <code>n_filters = 64</code> <code>kernel_size = 5</code> <code>dropout_rate = 0.1</code> <code>learning_rate = 0.0005</code> <code>batch_size = 128</code>
TCN	<code>n_filters = 32</code> <code>n_blocks = 3</code> <code>dropout_rate = 0.4</code> <code>learning_rate = 0.01</code> <code>batch_size = 32</code>
Transformer	<code>num_transformer_blocks = 2</code> <code>d_model = 128</code> <code>num_heads = 2</code> <code>ff_dim = 256</code> <code>dropout_rate = 0.1</code> <code>learning_rate = 0.001</code> <code>batch_size = 256</code>

C.4 Hybrid FE Hyperparameter Search Spaces

Table C.4: Hyperparameter search spaces used for the hybrid feature extraction models.

Architecture	Hyperparameter	Search Space
CNN-GRU	<code>conv_layers</code>	<code>{1, 2}</code>
	<code>filters</code>	<code>{32, 64, 128}</code>
	<code>kernel_size</code>	<code>{3, 5}</code>
	<code>pooling</code>	<code>{True, False}</code>
TCN-GRU	<code>num_blocks</code>	<code>{1, 2, 3}</code>
	<code>filters</code>	<code>{32, 64, 128}</code>
	<code>kernel_size</code>	<code>{3, 5}</code>
	<code>dropout</code>	<code>{0.1, 0.2, 0.3}</code>
AE-GRU	<code>encoder_layers</code>	<code>{1, 2}</code>
	<code>encoder_units</code>	<code>{32, 64, 128}</code>
	<code>bottleneck_dim</code>	<code>{4, 8, 12}</code>

Architecture	Hyperparameter	Search Space
	activation	{relu}
	lambda_recon	{0.01, 0.05, 0.1}

C.5 Selected Hybrid Feature Extraction Hyperparameters

The selected hybrid feature extraction hyperparameters reported in [Table C.5](#) were obtained using 30 Optuna trials for each architecture, with validation RMSE used as the tuning objective. For all hybrid models, the GRU prediction backend was kept fixed using the best Phase 1 GRU configuration (refer to [Table C.3](#) for the selected GRU hyperparameters).

Table C.5: Selected best hyperparameters for the hybrid feature extraction models.

Architecture	Selected Feature Extractor Hyperparameters
CNN-GRU	conv_layers = 1 filters = 32 kernel_size = 3 pooling = False
TCN-GRU	num_blocks = 3 filters = 32 kernel_size = 5 dropout = 0.3
AE-GRU	encoder_layers = 1 encoder_units = 32 bottleneck_dim = 4 activation = relu lambda_recon = 0.1

Appendix D

DANN Implementation

D.1 DANN Training Curves.

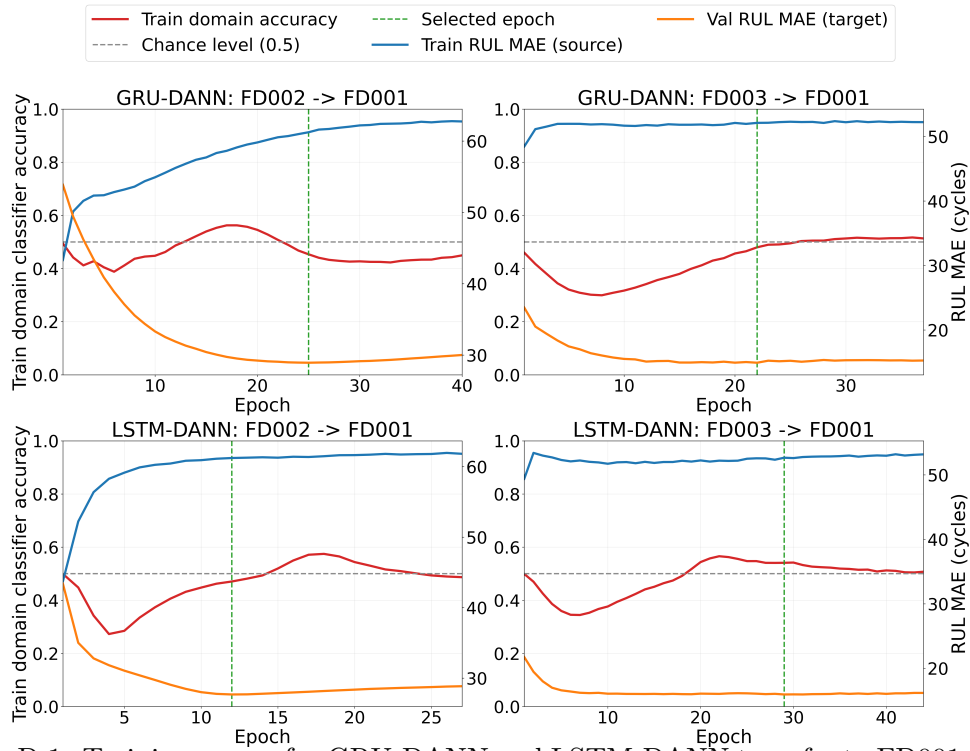


Figure D.1: Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 80 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).

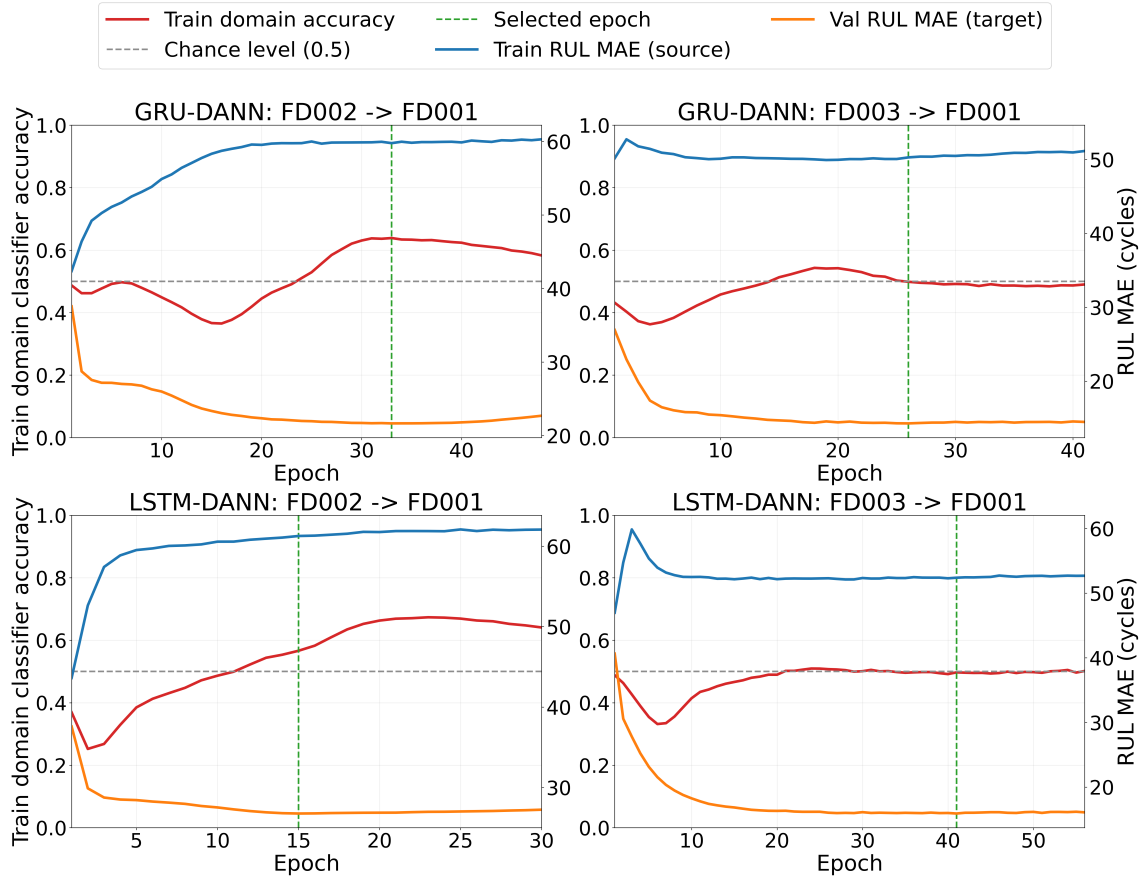


Figure D.2: Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 40 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).

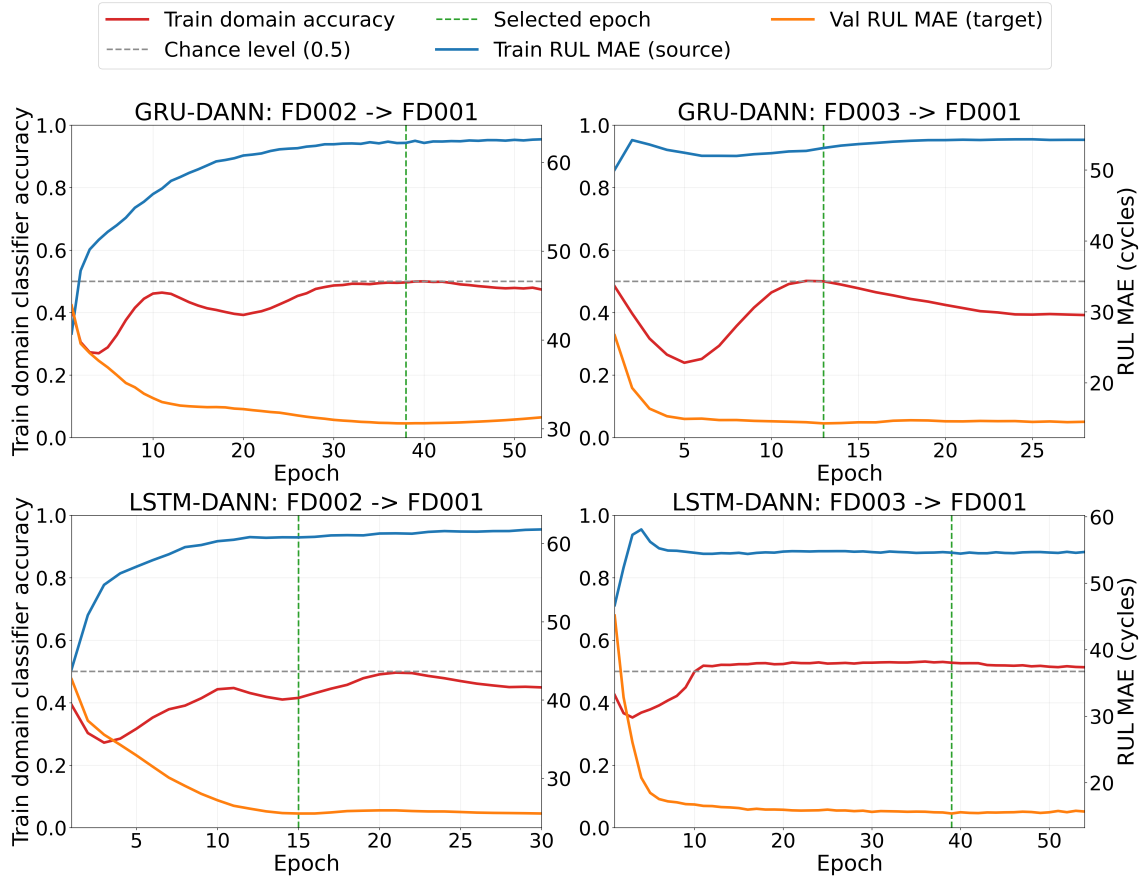


Figure D.3: Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 10 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).

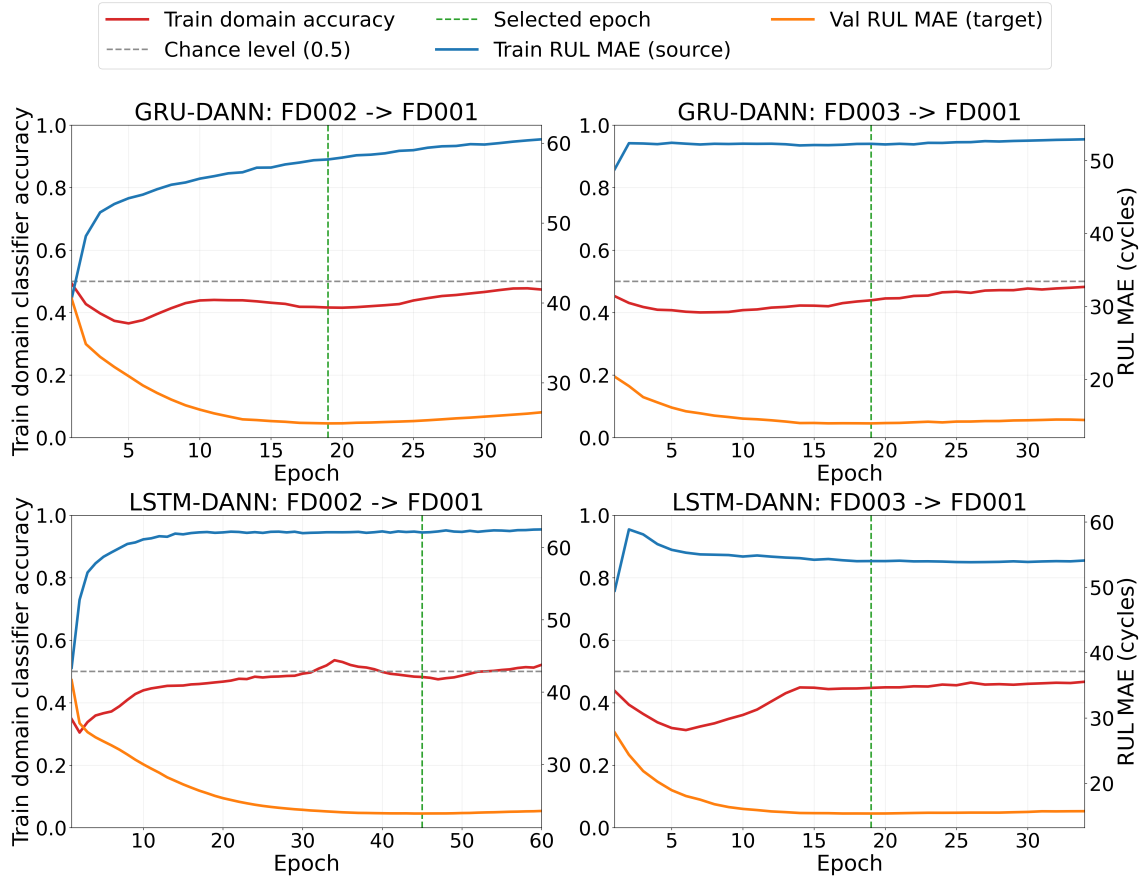


Figure D.4: Training curves for GRU-DANN and LSTM-DANN transfer to FD001 with 5 unlabeled target engines. Domain-classifier accuracy is on the left y-axis; source training RUL MAE and target validation RUL MAE (cycles) are on the right y-axis. The dashed grey line indicates chance domain accuracy (0.5); the green dashed line shows the selected epoch (minimum validation RUL loss).

D.2 DANN Batch Generator Implementation

```

1 def create_domain_batches(
2     source_data,
3     target_data,
4     batch_size=512,
5     oversample_smaller=True
6 ):
7     """
8     Create mixed-domain mini-batches for DANN training.
9
10    During training, mixed-domain mini-batches are constructed using
11    this generator function, which randomly samples from the source and
    target

```

```

12     training sets.
13
14     The source samples retain their RUL labels for supervised learning,
        while
15     the target samples are treated as unlabelled in the DANN setting.
        Therefore,
16     target samples are assigned dummy RUL labels with zero sample
        weights so
17     that they do not contribute to the RUL prediction loss.
18
19     Domain labels are assigned to all samples:
20         0 = source domain
21         1 = target domain
22
23     All source and target samples contribute to the domain-
        classification loss.
24     """
25
26     X_source = source_data['X_train']
27     y_source = source_data['y_train']
28     X_target = target_data['X_train']
29
30     n_source = len(X_source)
31     n_target = len(X_target)
32
33     # Calculate the number of batches in the source and target training
        sets.
34     # The maximum is used so that the smaller dataset can be reused
        when
35     # constructing the same number of batches as the larger dataset.
36     n_batches_source = int(math.ceil(n_source / batch_size))
37     n_batches_target = int(math.ceil(n_target / batch_size))
38     n_batches = max(n_batches_source, n_batches_target)
39
40     print(f"\nBatch configuration:")
41     print(f"    Source: {n_source} samples -> {n_batches_source} batches"
        )
42     print(f"    Target: {n_target} samples -> {n_batches_target} batches"
        )
43     print(f"    Total batches per epoch: {n_batches}")
44     print(f"    Batch size: {batch_size}")
45
46     while True:
47
48         # Randomly shuffle the source and target training indices at
            the start

```

```

49     # of each generator cycle, so that mini-batches are randomly
        sampled
50     # from both domains.
51     source_indices = np.random.permutation(n_source)
52     target_indices = np.random.permutation(n_target)
53
54     for batch_idx in range(n_batches):
55
56         # Source-domain batch
57         # Select a mini-batch from the source training set. The
            source domain
58         # provides both the input windows and the corresponding RUL
            labels.
59         start_source = (batch_idx * batch_size) % n_source
60         end_source = min(start_source + batch_size, n_source)
61
62         if end_source - start_source < batch_size and
            oversample_smaller:
63             # Oversampling is applied where necessary by wrapping
                around to
64             # the beginning of the shuffled indices. This allows
                the smaller
65             # dataset to be reused when constructing the same
                number of
66             # batches as the larger dataset.
67             indices_source = np.concatenate([
68                 source_indices[start_source:],
69                 source_indices[:batch_size - (n_source -
                    start_source)]
70             ])
71         else:
72             indices_source = source_indices[start_source:end_source
73             ]
74
75         X_source_batch = X_source[indices_source]
76         y_source_batch = y_source[indices_source]
77
78         # Target-domain batch
79         # Select a mini-batch from the target training set. In the
            DANN
80         # setting used here, the target samples are treated as
            unlabelled
81         # for the RUL prediction task.
82         start_target = (batch_idx * batch_size) % n_target
83         end_target = min(start_target + batch_size, n_target)

```

```

84         if end_target - start_target < batch_size and
            oversample_smaller:
85             # Oversampling is applied where necessary by wrapping
                around to
86             # the beginning of the shuffled indices. This balances
                the
87             # number of source and target batches used during
                training.
88             indices_target = np.concatenate([
89                 target_indices[start_target:],
90                 target_indices[:batch_size - (n_target -
                    start_target)]
91             ])
92         else:
93             indices_target = target_indices[start_target:end_target
                ]
94
95         X_target_batch = X_target[indices_target]
96
97         # Mixed-domain mini-batch
98         # Construct a mixed batch containing both source-domain and
99         # target-domain samples.
100        X_batch = np.concatenate([X_source_batch, X_target_batch],
            axis=0)
101
102        # RUL labels
103        # The source samples retain their true RUL labels for
            supervised
104        # learning. Since Keras expects a target array for each
            model output
105        # with the same batch dimension, the target-domain samples
            are
106        # assigned dummy RUL labels.
107        y_rul_batch = np.concatenate([
108            y_source_batch,
109            np.zeros(len(X_target_batch), dtype=np.float32)
110        ])
111
112        # Domain labels
113        # Each sample is assigned a domain label for the domain
            classifier:
114        # 0 denotes the source domain and 1 denotes the target
            domain.
115        y_domain_batch = np.concatenate([
116            np.zeros(len(X_source_batch), dtype=np.float32),
117            np.ones(len(X_target_batch), dtype=np.float32)
118        ])

```

```

119
120     # RUL sample weights
121     # The dummy RUL labels assigned to the target samples are
        given zero
122     # sample weights. This ensures that target-domain samples
        do not
123     # contribute to the RUL prediction loss.
124     sample_weights_rul = np.concatenate([
125         np.ones(len(X_source_batch), dtype=np.float32),
126         np.zeros(len(X_target_batch), dtype=np.float32)
127     ])
128
129     # Domain sample weights
130     # All source and target samples contribute to the domain-
        classification
131     # loss, since both domains are required for adversarial
        domain learning.
132     sample_weights_domain = np.ones(len(X_batch), dtype=np.
        float32)
133
134     yield (
135         X_batch,
136         {
137             'rul_output': y_rul_batch,
138             'domain_output': y_domain_batch
139         },
140         {
141             'rul_output': sample_weights_rul,
142             'domain_output': sample_weights_domain
143         }
144     )

```

Listing D.1: Mixed-domain mini-batch generator used for DANN training.

D.3 Build DANN Model

```

1 from tensorflow import keras
2 from gradient_reversal import GradientReversalLayer #This is a custom
    script that implements the Gradient Reversal Layer
3
4 def build_costa_dann(input_shape=(30, 14), lstm_units=(64,),
    lstm_dropout=0.1,
5                         feature_units=64, rul_units=(32,), rul_dropout
                        =0.0,
6                         domain_units=(16, 16), domain_dropout=0.1,

```

```

7         l2_reg=0.01, lambda_adversarial=1.0):
8     """
9     Build DANN model with three main components:
10    1. Shared LSTM feature extractor
11    2. RUL predictor branch
12    3. Domain classifier branch with gradient reversal layer (GRL)
13
14    Reference: Costa et al. (2020)
15    """
16
17    regularizer = keras.regularizers.l2(l2_reg)
18
19    # =====
20    # 1. SHARED LSTM FEATURE EXTRACTOR
21    # =====
22    # Processes windowed sensor data from both domains
23    # Learns domain-invariant representations through adversarial
24    # training
25
26    input_layer = keras.layers.Input(shape=input_shape, name='
27        sensor_input')
28
29    # LSTM layer(s) - configuration dependent on source-target pair
30    x = input_layer
31    for i, units in enumerate(lstm_units):
32        return_sequences = (i < len(lstm_units) - 1)
33        x = keras.layers.LSTM(
34            units=units,
35            dropout=lstm_dropout,
36            return_sequences=return_sequences,
37            kernel_regularizer=regularizer,
38            name=f'lstm_layer_{i+1}'
39        )(x)
40    lstm_out = x
41
42    # Feature extractor layer - creates shared domain-invariant
43    # features
44    features = keras.layers.Dense(
45        units=feature_units,
46        activation='relu',
47        kernel_regularizer=regularizer,
48        name='feature_extractor'
49    )(lstm_out)
50
51    # =====
52    # 2. RUL PREDICTOR BRANCH
53    # =====

```

```

51 # Predicts RUL using SOURCE domain labels only
52 # Loss: Mean Absolute Error (MAE)
53
54 rul_hidden = features
55 for i, units in enumerate(rul_units):
56     rul_hidden = keras.layers.Dense(
57         units=units,
58         activation='relu',
59         kernel_regularizer=regularizer,
60         name=f'rul_hidden_{i+1}'
61     )(rul_hidden)
62     if rul_dropout > 0.0:
63         rul_hidden = keras.layers.Dropout(
64             rul_dropout, name=f'rul_dropout_{i+1}'
65         )(rul_hidden)
66
67 # RUL output - linear activation for regression
68 rul_output = keras.layers.Dense(
69     units=1,
70     activation='linear',
71     kernel_regularizer=regularizer,
72     name='rul_output'
73 )(rul_hidden)
74
75 # =====
76 # 3. DOMAIN CLASSIFIER BRANCH WITH GRADIENT REVERSAL LAYER (GRL)
77 # =====
78 # GRL reverses gradients (multiplies by -) during backpropagation
79 # Forces feature extractor to learn domain-invariant
80 # representations
81 # Uses BOTH source and target domain samples
82
83 # Gradient Reversal Layer
84 grl_features = GradientReversalLayer(
85     lambda_=lambda_adversarial,
86     name='gradient_reversal'
87 )(features)
88
89 # Domain classifier hidden layers
90 domain_hidden = grl_features
91 for i, units in enumerate(domain_units):
92     domain_hidden = keras.layers.Dense(
93         units=units,
94         activation='relu',
95         kernel_regularizer=regularizer,
96         name=f'domain_hidden_{i+1}'
97     )(domain_hidden)

```

```

97         domain_hidden = keras.layers.Dropout(
98             domain_dropout, name=f'domain_dropout_{i+1}'
99         )(domain_hidden)
100
101     # Domain output as in the study sigmoid for binary classification
102     # is used(source=0, target=1)
103     # Loss: Binary Cross-Entropy
104     domain_output = keras.layers.Dense(
105         units=1,
106         activation='sigmoid',
107         kernel_regularizer=regularizer,
108         name='domain_output'
109     )(domain_hidden)
110
111     # =====
112     # MULTI-OUTPUT MODEL
113     # =====
114     # The model has two outputs; RUL prediction and the domain
115     # classification
116
117     model = keras.Model(
118         inputs=input_layer,
119         outputs=[rul_output, domain_output],
120         name='Costa_DANN'
121     )
122
123     return model

```

Listing D.2: Mixed-domain mini-batch generator used for DANN training.

D.4 The Gradient Reversal Layer

```

1  """
2  Gradient Reversal Layer (GRL) Implementation
3  Based on Ganin et al. 2016 - Domain-Adversarial Training of Neural
4      Networks
5
6  This layer acts as an identity function during forward pass,
7  but reverses the gradient sign during backpropagation (multiplies by -
8      lambda).
9  """
10
11  import tensorflow as tf
12  from tensorflow import keras

```



```

12 #https://www.tensorflow.org/api_docs/python/tf/custom_gradient
13 @tf.custom_gradient #this is a tensorflow function enables us to define
    how we want the gradient to be computed
14 def gradient_reversal(x, lambda_):
15     """
16     Gradient reversal function.
17
18     Forward pass: y = x (identity)
19     Backward pass: dy/dx = -lambda * grad
20
21     """
22     def grad(dy):
23         return -lambda_ * dy, None
24
25     return x, grad
26
27
28 class GradientReversalLayer(keras.layers.Layer):
29     """
30     Gradient Reversal Layer (GRL).
31
32     This layer implements the gradient reversal operation as described
        in Ganin et al. (2016)
33     During forward propagation the output is the same as the input
        however in back propagation the sign
34     is flipped by multiplying lambda with -1. This essentially flips the
        gradient. Through this adversarial training takes place by
35     having an objective of the domain classifier set to minimise the
        loss while having the feature extractor minimising the loss.
36
37     """
38     #define lambda
39     def __init__(self, lambda_=1.0, **kwargs):
40
41         super(GradientReversalLayer, self).__init__(**kwargs) #parent
            class to make sure it behaves like normal keras layer
42
43         self.lambda_ = lambda_
44
45     #this is the forward pass with the gradient reversal
46     def call(self, x):
47
48         return gradient_reversal(x, self.lambda_)
49
50     #this is the serialization of the layer which so that the layer can
        be saved
51     def get_config(self):

```

```
52     """Enable serialization."""
53     config = super(GradientReversalLayer, self).get_config()
54     config.update({'lambda_': self.lambda_})
55     return config
```

Listing D.3: Mixed-domain mini-batch generator used for DANN training.