A Transfer Learning Framework for Remaining Useful Life Estimation

Melanie B. Sigl[1],Klaus Meyer-Wegener[2]

[1,2] *Chair of Computer Science 6 (Data Management),*
Friedrich-Alexander-Universität Erlangen-Nürnberg, Martensstraße 3, 91058 Erlangen, Germany
melanie.sigl@fau.de
klaus.meyer-wegener@fau.de

ABSTRACT

Training a robust deep learning (DL) model for remaining useful life (RUL) estimation or fault detection typically requires a large, high-quality labeled dataset. However, such datasets are often unavailable in practice. Transfer learning is a solution for smaller labelled datasets. Yet, the effectiveness of transfer learning heavily depends on selecting an appropriate source DL model; an unsuitable choice can result in negative transfer, where model performance deteriorates significantly.

To address this challenge, we introduce REAPER (**R**eusable N**eura**l Network **P**att**e**rn **R**epository), a framework designed to assist users in selecting the most suitable DL model for reuse in transfer learning scenarios. REAPER analyzes and compares the characteristics of available datasets and employs a learned ranking model to recommend the optimal source model. This paper presents the architecture, including its dataset characterization, ranking methodology, training procedure, and practical usage guidance.

1. INTRODUCTION

Remaining useful life (RUL) estimation is a critical task in predictive maintenance, where data-driven methods – particularly deep learning (DL) models – are increasingly employed (Zheng, Ristovski, Farahat, & Gupta, 2017; Huang, Khorasgani, Gupta, Farahat, & Zheng, 2018; Das, Hussain, Yang, Habibullah, & Kumar, 2019; Yao, Yang, Liu, & Zheng, 2019; Noot, Martin, & Birmele, 2025). However, the effectiveness of DL-based RUL estimation depends heavily on the availability of large, high-quality datasets that contain labeled failure instances (Moradi & Groth, 2020). In many industrial scenarios, such datasets are scarce due to the infrequency of failures or the high cost of data labeling (Listou Ellefsen, Bjørlykhaug, Æsøy, Ushakov, & Zhang, 2019). To address this limitation, a widely adopted strategy is to apply transfer learning, which reuses a DL model pre-trained on a source dataset to improve performance in the target dataset (Pan & Yang, 2010).

By leveraging previously learned knowledge that resides in the model's architecture and learned parameters, transfer learning requires less training samples (Fan, Nowaczyk, & Rögnvaldsson, 2019; Han, Liu, Wu, & Jiang, 2021) while enhancing the model's performance (Pan & Yang, 2010; Yosinski, Clune, Bengio, & Lipson, 2014). However, it is crucial to avoid *negative* transfer learning, characterized by a reduced performance. Research found that this effect is attributed to the dissimilarity between the source and target dataset (Yosinski et al., 2014; Fawaz, Forestier, Weber, Idoumghar, & Muller, 2018).

Approach: This paper describes a framework to store DL models, datasets, and their metadata such as performance metrics to select the best DL model for a transfer learning scenario. The framework implements our recently proposed learning-to-rank approach (Sigl & Meyer-Wegener, 2025) for RUL estimation in a fleet-based setting. This approach utilizes dataset characteristics and performance improvements of known datasets and their models to generate rankings and train a ranking model. This paper describes the system behind the ranking approach for model selection.

Contributions: This paper presents REAPER (**R**eusable N**eura**l Network **P**att**e**rn **R**epository) – a framework that supports selecting the best DL models for transfer learning in low-data regimes, e.g., RUL estimation across fleets of identical or similar equipment. The core contributions are:

- A scalable architecture for managing and applying transfer learning in fleet-based prognostics and health management (PHM) settings and beyond.
- A framework for organizing datasets, DL models, and their relationships.

- An automated workflow for assessing dataset similarity and guiding model reuse.

This paper is structured as follows: Section 2 introduces relevant background and Section 3 discusses related work. Section 4 describes the framework's goal and relevant use cases, while Section 5 proposes the architecture and relevant components. Evaluation and results are discussed in Section 6. This paper concludes in Section 7 with a short summary and future work.

2. BACKGROUND

This section outlines key concepts such as transfer learning (Section 2.1) and learning to rank (Section 2.2). Following these, Section 2.3 briefly describes how learning to rank can be used to rank DL models for transfer learning.

2.1. Transfer Learning

Transfer learning is a technique that adapts a DL model trained for one dataset to a related dataset (target) (Pan & Yang, 2010). It involves two main steps. First, a DL model trained on a large dataset is modified to suit the new task – such as changing the output layer or resetting parts of the model for retraining. Second, the model is fine-tuned using data from the new dataset. During fine-tuning, some layers may be "frozen" (i.e., kept unchanged) while others are updated based on the new data (Yosinski et al., 2014). This is especially useful when limited data is available.

2.2. Learning to Rank

Learning to rank (LTR) is a technique for ordering elements by relevance to a specific query. The goal is to place the most relevant items at the top (Järvelin & Kekäläinen, 2002). Early heuristic methods have been largely replaced by machine learning and DL models (Liu, 2011). LTR methods are typically grouped into three categories: pointwise, pairwise, and listwise, each defining the learning objective and training data structure.

Pointwise methods treat ranking as a regression or classification task, predicting relevance scores for individual items without considering their relationships or overall ranking. *Pairwise* methods compare pairs of items to learn which is more relevant, but may miss broader dependencies within a query. In contrast, *listwise* methods directly optimize the entire ranked list, often achieving better accuracy, though they require more complex computations.

2.3. Learning to Rank DL Models for Transfer Learning

In our previous work (Sigl & Meyer-Wegener, 2025), we introduced a method for selecting DL models for multivariate time-series by ranking models based on their *performance improvement* through transfer learning. This improvement is measured as the difference between the performance of a transferred model and one trained from scratch on the target dataset. Since transfer learning success depends on the similarity between source and target datasets, our approach uses dataset characteristics to guide model ranking.

The approach follows a three-step process:

1. **Ranking Ground Truth.** We established a ground truth by training a DL model on each known dataset and transferring it to all others. Performance improvements were calculated to rank DL models per target dataset.
2. **Dataset Characteristics.** We extracted three characteristics from each dataset: statistical, shape-based, and a combination of both.
3. **Ranking Model.** We trained a pairwise LTR model, RankNet (Burges et al., 2005), to learn how to rank DL models based on these characteristics, with the goal of maximizing performance improvement. Using a pairwise LTR method allows to order any number of datasets.

The proposed approach (Sigl & Meyer-Wegener, 2025) is applicable to multivariate and univariate time-series datasets.

3. RELATED WORK

Transfer Learning and Model Selection. The problem of selecting the most appropriate DL model for transfer is rarely addressed – particularly for (multivariate) time-series data. Most existing studies on transfer learning focus on evaluating how well a specific DL architecture transfers knowledge from a source to a target dataset (Mao, He, & Zuo, 2020; Tang, Ma, Yan, Zhu, & Khoo, 2024). Existing approaches that study the selection problem often target univariate time series (Fawaz et al., 2018) or simplify multivariate data by transforming it into a univariate form prior to training (Ye & Dai, 2021). Research addressing model selection typically focuses on image data (Istrate et al., 2019) and text (Ruder & Plank, 2017). For example, Istrate et al. (2019) propose a system that predicts the performance of DL models to recommend the best. However, their approach relies on a large data basis for accurate performance predictions and the system's initialization reportedly took 18 months. Recently, we (Sigl & Meyer-Wegener, 2025) proposed a method for multivariate time-series datasets that robustly recommends the best DL model even with a small data basis.

Model Management. Platforms like TensorFlow Hub[1], ModelZoo[2], and Hugging Face[3] allow sharing DL models, with Hugging Face also supporting datasets. Systems such as MLflow[4], ModelDB (Vartak et al., 2016), and ModelHub (Miao, Li, Davis, & Deshpande, 2017) track models, hyperparameters, and training data, with basic keyword-based

[1] https://www.tensorflow.org/hub
[2] https://modelzoo.co/
[3] https://huggingface.co/
[4] https://mlflow.org/

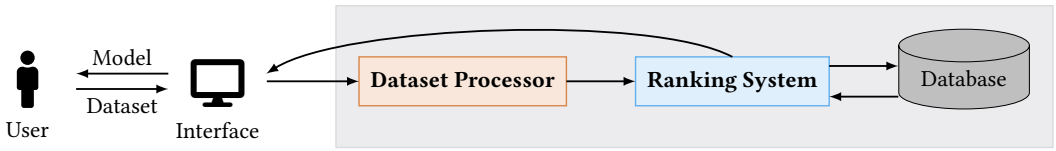


Figure 1. General workflow how a data scientist interacts with the system.

search capabilities. While helpful, keyword search depends on quality and user familiarity, making it difficult to identify suitable DL models for positive transfer learning. These systems do not consider dataset similarity in the search process.

Dataset Management: Store and Capture. Systems like SciDB (Stonebraker, Brown, Poliakov, & Raman, 2011), DataHub (Bhardwaj et al., 2015), Decibel (Maddox et al., 2016), and DataLab (Zhang et al., 2016, 2017) support dataset versioning and querying, but none offer similarity-based search for datasets, especially in the context of transfer learning workflows.

Dataset Management: Discover and Share. GOODS (Halevy et al., 2016), AURUM (Castro Fernandez et al., 2018), and Google Dataset Search (Noy, Burgess, & Brickley, 2019) extract metadata from distributed storage to enable dataset search. While AURUM uniquely supports similarity queries across relational tables using content and schema profiles, these systems do not address transfer-learning-specific needs.

4. Overview and Goals

The goal of the framework, REAPER, is to help users in selecting the best possible DL model for transfer learning. Figure 1 gives a general overview on how users interact with REAPER. A user interacts via an interface and presents a target datasest, so it serves as a query (dataset) in our system. Presenting a dataset triggers the extraction of characteristics (e.g., descriptive statistics, distributional statistics, time-series characteristics) via the *Dataset Processor* (Sigl & Meyer-Wegener, 2025). These characteristics are used in the *Ranking System*, a component that employs a trained ranking model (e.g., RankNet) to create an order of all database-stored datasets and models for the presented query. REAPER returns the DL model that is ranked highest, i.e. the best model.

To support users effectively, this section outlines the framework's requirements (4.1) and essential use cases (4.2).

4.1. Requirements

Our proposed framework satisfies pre-specified requirements to ensure core functionality to its users:

- **Usability & User Interaction.** An easy to use, yet simple and functional interface. Interfaces include a web user interface (UI) for quick exploration, and a software development kit (SDK) to be used from within the code, e.g., Python[5] or Jupyter Notebook[6].
- **Model Recommendation.** Suggest the best pre-trained DL models based on dataset characteristics to increase model performance.
- **Asset Analyzer.** Assets such as datasets and DL models are processed to ensure quick analysis in the web UI and fast DL model recommendation.
- **Data Management.** Support structured and unstructured data formats like (multivariate) time-series datasets, DL models, and their metadata.
- **Modular & Extensible.** System components are modularly designed and extensible: It is easy to extract and store new dataset characteristics or add a new ranking model.

4.2. Use Cases

In addition to its core requirements, REAPER is designed to support four key use cases: importing artifacts, constructing the ranking ground truth, training the ranking model, and responding to user queries by recommending the most suitable DL model.

Use Case ①: Import Artifacts. Our ranking approach (Sigl & Meyer-Wegener, 2025) depends on a populated repository and a trained ranking model, making artifact import the initial step. As shown in Figure 2, users can import datasets, DL models, and metadata either automatically via a CI pipeline (e.g., GitHub Actions[7], GitLab CI/CD[8]) using the SDK, or manually through the web UI.

REAPER automatically identifies and processes artifacts based on their data type. Multivariate time-series datasets, typically stored as comma-separated values (CSV) files, are handled by the *Dataset Processor*, which verifies data format compatibility, file integrity, and sufficient observation count. It also extracts relevant characteristics, including descriptive statistics, distribution measures, and time-series features. Similarly, the *Model Processor* handles DL models with tailored validation steps. Once processed, all artifacts are stored in REAPER's database.

[5] https://www.python.org/
[6] https://jupyter.org/
[7] https://github.com/features/actions
[8] https://docs.gitlab.com/ci/pipelines/

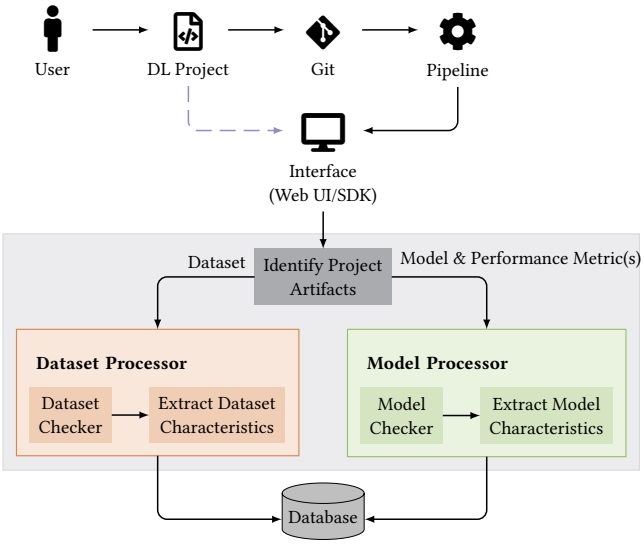


Figure 2. Importing artifacts.

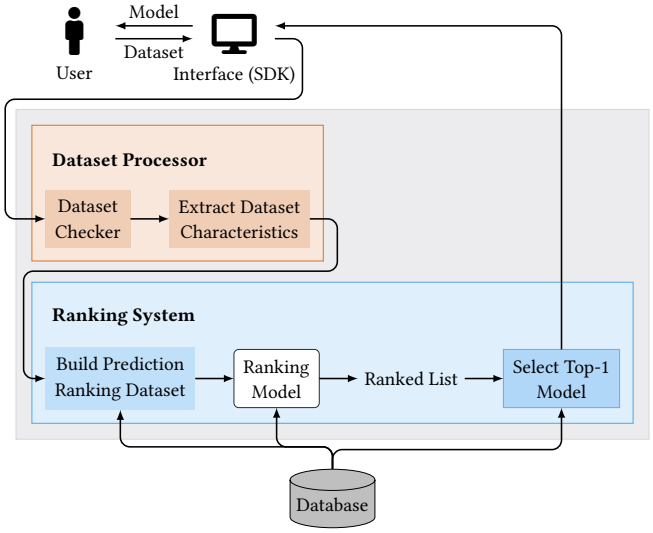


Figure 3. Using REAPER to retrieve the best DL model.

Use Case ②: Building the Ground Truth for Ranking. After importing artifacts into a newly initialized system, the *ranking ground truth* must be established before training a ranking model. This ground truth is essential for accurately ranking DL models for a given query dataset.

The creation of the ranking ground truth is a two-step process (Sigl & Meyer-Wegener, 2025). First, a selected dataset from the database is treated as the target, and DL models trained on other datasets are adapted according to a user-defined modification specification – e.g., by reinitializing the output layer. Second, the modified models are fine-tuned on the target dataset.

Following this brute-force transfer learning, we compute the *performance improvement* to assess whether each transfer was beneficial. The performance improvement is calculated as (Sigl & Meyer-Wegener, 2025):

$$\Delta\text{perf}(\psi, \psi^{transfer}) = -\frac{\psi - \psi^{transfer}}{\psi},$$

where $\psi^{transfer}$ is the performance of the transferred model, and ψ is the performance of a model trained from scratch on the target dataset, using the same performance metric (e.g., mean squared error (MSE)). For metrics where higher values indicate better performance (e.g., accuracy), the fraction is multiplied by $+1$ instead of -1.

Use Case ③: Train the Ranking Model. Once the ranking ground truth is established, users can initiate the training of the ranking model. REAPER currently supports RankNet, a pairwise LTR model, though other models can be integrated.

To train RankNet, the system generates training, validation, and test *rankings* by selecting each dataset once as a query D_Q and forming all possible pairs D_k, D_l from the remaining datasets. For each pair, the performance improvements Δperf_k and Δperf_l of D_k, D_l, respectively, are compared with respect to D_Q. A binary label is assigned: 1 if $\Delta\text{perf}_k \leq \Delta\text{perf}_l$, and 0 otherwise. The model is trained using the dataset characteristics of the triple (D_Q, D_k, D_l) as input and the binary label as output.

Users can configure how database-stored elements are split into training, validation, and test sets – either randomly or by explicitly selecting specific datasets. Prior work (Sigl & Meyer-Wegener, 2025) has shown that even small rankings of four datasets are sufficient to train an effective ranking model. The trained model is stored in REAPER's database for future use.

Use Case ④: Retrieve the best DL model for Transfer Learning. Completing Use Cases 1–3 prepares the system for use. Once set up, users can submit a query dataset to retrieve the most suitable DL model. The full workflow is illustrated in Figure 3. As shown, the system first processes the query dataset by performing a quick validation and extracting its characteristics. These characteristics are then passed to the *Ranking System*, which generates prediction rankings in the same format as the training and validation data, but without labels. The trained ranking model uses this input to produce a ranked list of DL models, from which the highest-ranked model is returned to the user.

Execution Order of the Use Cases. All use cases must be executed in the specific order as presented to complete the system setup. Once the setup is complete, a ranking model becomes available, and users are free to import artifacts and retrieve DL models in any order they choose (Use Cases 1 and 4). Similarly, users can train a new ranking model at any time by executing Use Cases 2 and 3 together.

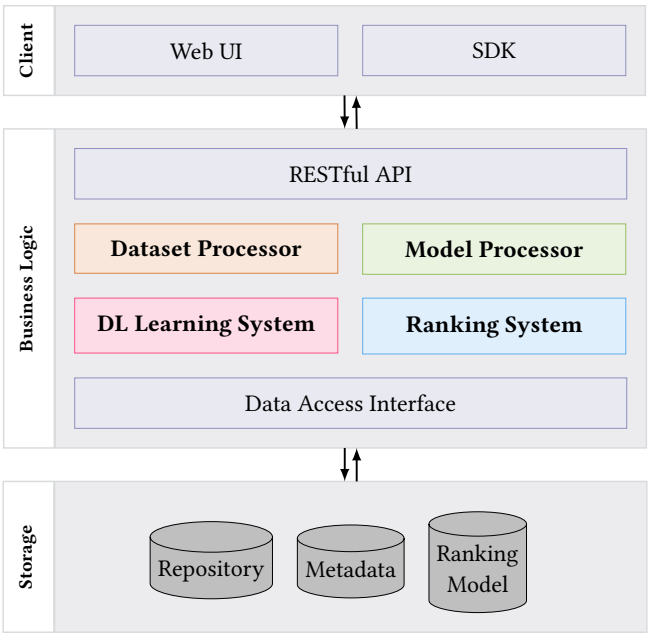


Figure 4. Architecture.

Listing 1. Python script to find and download a DL model.

```python
from reaper.backend import Backend
from reaper.dataset import Characteristics
from reaper.model import Model
from reaper.ranking import RankingModel

# Initialize and connect with REAPER's
# backend via a configuration file
backend = Backend.from_config()

# Extract characteristics from a local dataset
characteristics = Characteristics.from_file(
    path="./dataset/train.csv"
)

# Find and fetch a DL model using REAPER's
# default ranking model
model = backend.find_model_for_transfer(
    characteristics=characteristics
)

# Find and fetch a DL model using a specific
# ranking model
ranking_model = RankingModel.get_by_id(
    backend=backend, id=42
)
model = backend.find_model_for_transfer(
    characteristics=characteristics,
    ranking_model=ranking_model
)
model.download(path="./model/")
```

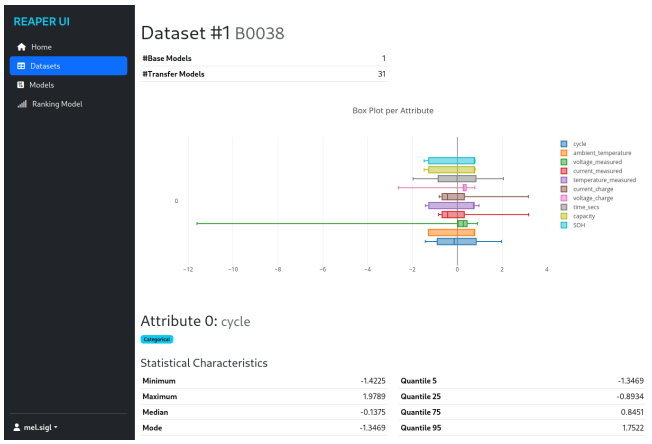


Figure 5. Web interface to analyse datasets and their metadata.

5. Architecture & Components

REAPER is designed as a modular three-tier architecture consisting of a client layer, a business logic layer, and a storage layer (see Figure 4). Each layer exposes well-defined interfaces, allowing for independent development and potential interchangeability. Figure 4 also names the key components within each layer.

All layers are currently implemented in Python to ensure seamless integration, reduce language-related overhead, and simplify development and maintenance.

5.1. Client Layer

The client layer implements two interfaces: a web UI and an SDK.

The **web UI** provides a graphical interface to REAPER, allowing users to browse available datasets and DL models, inspect their attributes, and view reuse history of models. As shown in Figure 5, users can also access detailed metadata for each stored dataset. In addition, users can initiate re-training of the ranking model (see Use Cases 2 and 3).

The **SDK** offers a programmatic interface to REAPER, enabling similar functionality through code. As illustrated in Listing 1, users can interact with REAPER from within Python to search for and download the most suitable DL model for transfer learning.

5.2. Business Logic Layer

The business logic layer, also referred to as the backend, provides interfaces to both the client and storage layers. It encapsulates key components shown in Figures 2 and 4, including the *Dataset Processor*, *Model Processor*, and *Ranking System*. Additionally, it contains the *DL Learning System*, which is responsible for generating the ranking ground truth and training the ranking model. Communication with the client layer is handled through a *RESTful API*, offering a stateless and uniform interface (Fielding, 2000), while interaction with the storage layer is managed via a dedicated *Data Access Interface*.

5.3. Storage Layer

The storage layer manages all artifacts – datasets, DL models, and their metadata – handling both structured and unstructured data. It comprises three components: a repository for datasets and DL models, a datastore for trained ranking models, and a relational database for metadata.

Datasets, DL models, and ranking models are stored in their original file formats on the file system, while metadata is maintained in PostgreSQL[9], a relational database. This metadata database organizes information including dataset characteristics, model details, performance metrics, and training relationships. Specifically, it supports:

- **Provenance and Ownership:** Record ownership, contact, origin, and file system paths for each database-stored artifact pair, linked to a single DL project. This supports traceability and accountability for datasets and models.
- **Dataset Characteristics:** Store extracted dataset features to support ranking and enable dataset exploration via the web UI.
- **DL Model Metadata:** Include model type, architecture, and output activation function to support task-based filtering (e.g., classification, forecasting).
- **Dataset–Model Relationships:** Capture links between datasets and DL models, including project affiliation and model performance.
- **Model Lineage:** Track DL model provenance to identify common origins of transferred models.
- **File Paths:** Maintain file system paths for all artifacts and ranking models to enable retrieval and reproducibility.
- **Ranking Model Metadata:** Identify DL projects used to build the ranking dataset, specify the active ranking model, and store its file path.

5.4. Scalability To Support Use Cases 2 and 3

The business logic layer is modular and supports transfer learning and ranking model training, which may require GPU-equipped machines depending on stored DL models. Creating the ranking ground truth involves fine-tuning models on multiple datasets, which can be time-consuming if done sequentially. To address this, the backend operates in two modes (see Figure 6): a *server* – simply called REAPER in the figure – providing a REST API and managing storage, and a *worker* designed to run on a compute cluster. When training is triggered ①, tasks are created and sent via RabbitMQ[10] to worker nodes ②, which process them in parallel and return results through the backend ④. This workflow

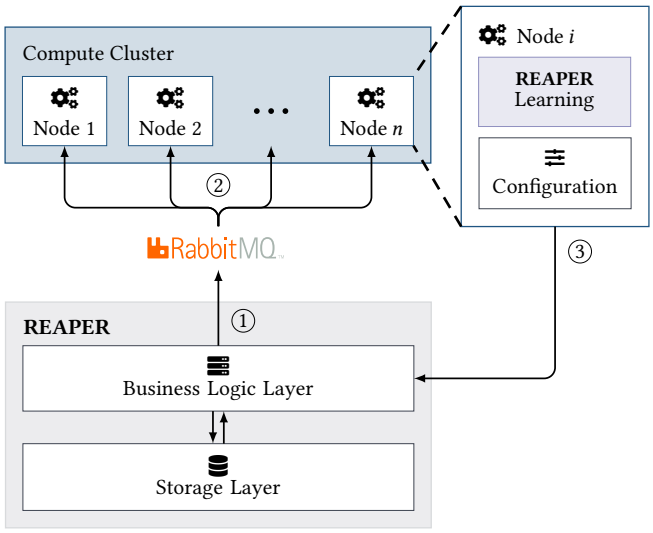


Figure 6. REAPER's two execution modes to scale for Use Cases 2 and 3.

ensures the business logic layer remains responsive and supports a loosely coupled, scalable system architecture.

Note that Figure 6 omits the client layer for simplicity.

6. Evaluation and Results

The model ranking component integrated into REAPER is based on our previously proposed LTR approach for selecting DL models in transfer learning scenarios (Sigl & Meyer-Wegener, 2025). This method was empirically validated using NASA's publicly available battery dataset (Saha & Goebel, 2007), which contains sensory information of 34 lithium-ion cells that have been aged in a controlled environment to show degradations.

In that evaluation, long short-term memory (LSTM) models trained on selected source datasets were transferred to target datasets representing different battery usage profiles. As can be seen in Figure 7, the model selection process, guided by dataset characteristics and performance history, consistently identified models suitable for positive transfer learning (up to 95% using statistical dataset characteristics). The evaluation metric used in this analysis is the NDCG at position 1. The

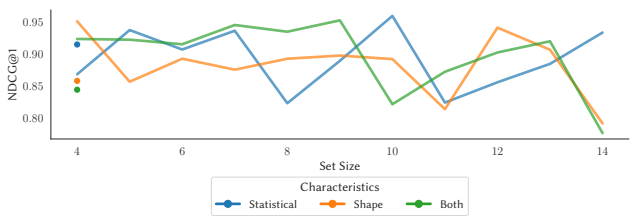
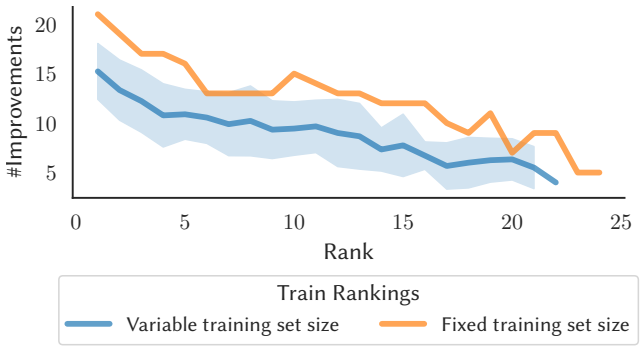


Figure 7. Normalised discounted cumulative gain (NDCG)@1 of RankNet training with varying train rankings.

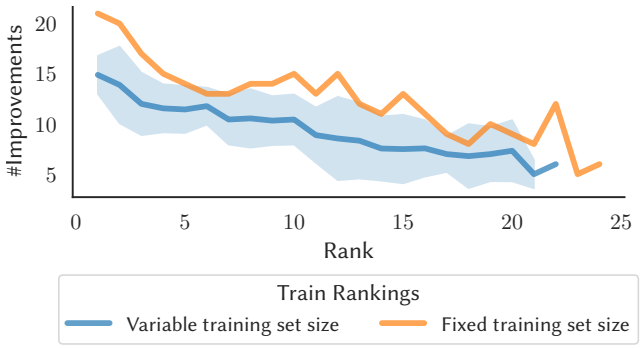
[9] https://www.postgresql.org/
[10] https://www.rabbitmq.com/

dots in this figure correspond to the fix-sized train rankings, while the lines correspond to the varying-sized train ranking. As shown, experiments with varying ranking set sizes demonstrate that even small training rankings consisting of only four datasets confirm the robustness and generalization capability of the learned ranking model.

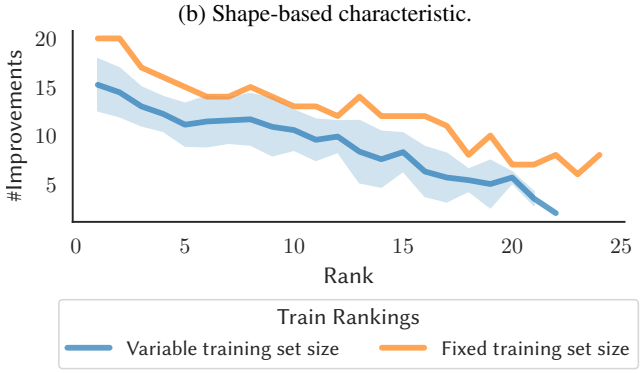
In addition, Figure 8 shows that the trained ranking model consistently places the best-performing DL model for positive transfer learning at rank 1, while assigning lower ranks to models with poorer performance or negative transfer learning values across all characteristics – statistical, shape-based, and combined. The error band around the blue line represents the standard deviation.



(a) Statistical characteristic.



(b) Shape-based characteristic.



(c) Combination of statistical and shape-based characteristics.

Figure 8. Number of improvements per rank.

These results demonstrate that the ranking model employed within REAPER is capable of identifying highly transferable models based solely on dataset-level features. While the experimental details and benchmarking are fully described in our earlier work, their inclusion in REAPER enables the system to provide practical, data-driven support for model reuse in fleet-based PHM applications. Practical use cases include ion mill fault diagnosis (TV, Gupta, Malhotra, Vig, & Shroff, 2018), as well as integration into digital twin environments aimed at optimizing product quality in large-scale industrial systems such as air separation units (Blum et al., 2021).

7. Conclusion and Future Work

This paper presented REAPER, a framework designed to support the selection and reuse of DL models for transfer learning, particularly in fleet-based PHM applications. Built upon our previously proposed learning-to-rank approach, REAPER leverages dataset characteristics and historical performance improvements to train a ranking model that guides DL model selection. REAPER introduces a scalable architecture, an automated workflow for assessing dataset similarity and enabling model reuse, and stores datasets, DL models, their metadata, and trained ranking models. We described the key use cases and requirements that shape REAPER, and detailed its architecture and core components to facilitate the selection of DL models for transfer learning. By supporting selecting the best DL models for transfer learning, REAPER offers a practical and scalable solution in real-world PHM settings. Future work will focus on expanding the dataset characteristics and evaluating REAPER across additional datasets.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. Melanie B. Sigl also thanks Shofiyati Nur Karimah for her helpful advice in the early stages of this work.

References

Bhardwaj, A. P., Bhattacherjee, S., Chavan, A., Deshpande, A., Elmore, A. J., Madden, S., & Parameswaran, A. G. (2015). DataHub: Collaborative Data Science & Dataset Version Management at Scale. In *CIDR*.

Blum, N., Krespach, V., Zapp, G., Oehse, C., Rehfeldt, S., & Klein, H. (2021, 11). Investigation of a Model-Based Deep Reinforcement Learning Controller Applied to an Air Separation Unit in a Production Environment. *Chemie Ingenieur Technik*, *93*(12), 1937–1948.

Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., & Hullender, G. N. (2005). Learning to Rank Using Gradient Descent. In L. D. Raedt & S. Wrobel (Eds.), *ICML* (Vol. 119, pp. 89–96). ACM.

Castro Fernandez, R., Abedjan, Z., Koko, F., Yuan, G., Mad-

den, S., & Stonebraker, M. (2018, 4). Aurum: A Data Discovery System. *ICDE*.

Das, A., Hussain, S., Yang, F., Habibullah, M. S., & Kumar, A. (2019). Deep Recurrent Architecture with Attention for Remaining Useful Life Estimation. *IEEE Region 10 Conference (TENCON)*.

Fan, Y., Nowaczyk, S., & Rögnvaldsson, T. S. (2019). Transfer learning for Remaining Useful Life Prediction Based on Consensus Self-Organizing Models. *CoRR*, *abs/1909.07053*.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. (2018). Transfer Learning for Time Series Classification. In *IEEE international conference on big data* (pp. 1367–1376).

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures* (Unpublished doctoral dissertation). University of California, Irvine.

Halevy, A., Korn, F., Noy, N. F., Olston, C., Polyzotis, N., Roy, S., & Whang, S. E. (2016). Goods: Organizing Google's Datasets. *SIGMOD*.

Han, T., Liu, C., Wu, R., & Jiang, D. (2021). Deep Transfer Learning with Limited Data for Machinery Fault Diagnosis. *Applied Soft Computing*, *103*, 107150.

Huang, W., Khorasgani, H., Gupta, C., Farahat, A., & Zheng, S. (2018). Remaining Useful Life Estimation for Systems with Abrupt Failures. *Proceedings of the Annual Conference of the PHM Society*, *10*(1).

Istrate, R., Scheidegger, F., Mariani, G., Nikolopoulos, D. S., Bekas, C., & Malossi, A. C. I. (2019). TAPAS: Train-Less Accuracy Predictor for Architecture Search. In *AAAI* (pp. 3927–3934).

Järvelin, K., & Kekäläinen, J. (2002). Cumulated Gain-Based Evaluation of IR Techniques. *ACM Transactions on Information Systems*, *20*(4), 422–446.

Listou Ellefsen, A., Bjørlykhaug, E., Æsøy, V., Ushakov, S., & Zhang, H. (2019, 3). Remaining Useful Life Predictions for Turbofan Engine Degradation Using Semi-supervised Deep Architecture. , *183*, 240–251.

Liu, T. (2011). *Learning to Rank for Information Retrieval*. Springer.

Maddox, M., Goehring, D., Elmore, A. J., Madden, S., Parameswaran, A., & Deshpande, A. (2016, 5). Decibel: The Relational Dataset Branching System. *Proceedings of the VLDB Endowment*, *9*(9), 624–635.

Mao, W., He, J., & Zuo, M. J. (2020, 4). Predicting Remaining Useful Life of Rolling Bearings Based on Deep Feature Representation and Transfer Learning. *IEEE Transactions on Instrumentation and Measurement*, *69*(4), 1594–1608.

Miao, H., Li, A., Davis, L. S., & Deshpande, A. (2017, 4). ModelHub: Deep Learning Lifecycle Management. *ICDE*.

Moradi, R., & Groth, K. M. (2020). On the Application of Transfer Learning in Prognostics and Health Management. *CoRR*, *abs/2007.01965*.

Noot, J.-P., Martin, M., & Birmele, E. (2025). LSTM and Transformers based methods for Remaining Useful Life Prediction considering Censored Data. *IJPHM*, *16*(2).

Noy, N. F., Burgess, M., & Brickley, D. (2019). Google Dataset Search: Building a search engine for datasets in an open Web ecosystem. In *The world wide web conference* (pp. 1365–1375).

Pan, S. J., & Yang, Q. (2010). A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359.

Ruder, S., & Plank, B. (2017). Learning to select data for transfer learning with Bayesian Optimization. In M. Palmer, R. Hwa, & S. Riedel (Eds.), *EMNLP* (pp. 372–382). Association for Computational Linguistics.

Saha, B., & Goebel, K. (2007). *Battery Data Set.* Retrieved 2022-05-09, from `https://www.nasa.gov/intelligent-systems-division/discovery-and-systems-health/pcoe/pcoe-data-set-repository/`

Sigl, M. B., & Meyer-Wegener, K. (2025). Towards Learning to Rank Deep-Learning Models for Multivariate Time-Series Transfer Learning. In *DEEM* (pp. 2:1–2:9).

Stonebraker, M., Brown, P., Poliakov, A., & Raman, S. (2011). The Architecture of SciDB. In *SSDBM* (pp. 1–16).

Tang, S., Ma, J., Yan, Z., Zhu, Y., & Khoo, B. C. (2024). Deep Transfer Learning Strategy in Intelligent Fault Diagnosis of Rotating Machinery. *Engineering Applications of Artificial Intelligence*, *134*, 108678.

TV, V., Gupta, P., Malhotra, P., Vig, L., & Shroff, G. (2018, 9). Recurrent Neural Networks for Online Remaining Useful Life Estimation in Ion Mill Etching System. *Proceedings of the Annual Conference of the PHM Society*, *10*(1).

Vartak, M., Subramanyam, H., Lee, W.-E., Viswanathan, S., Husnoo, S., Madden, S., & Zaharia, M. (2016). ModelDB: A System for Machine Learning Model Management. In *Proceedings of the workshop on human-in-the-loop data analytics* (pp. 14:1–14:3).

Yao, Q., Yang, T., Liu, Z., & Zheng, Z. (2019). Remaining Useful Life Estimation by Empirical Mode Decomposition and Ensemble Deep Convolution Neural Networks. *ICPHM*.

Ye, R., & Dai, Q. (2021). Implementing transfer learning across different datasets for time series forecasting. *Pattern Recognition*, *109*, 107617.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (pp. 3320–3328). Curran Associates, Inc.

Zhang, Y., Xu, F., Frise, E., Wu, S., Yu, B., & Xu, W. (2016). DataLab: A Version Data Management and Analytics System. In *BIGDSE* (pp. 12–18). ACM.

Zhang, Y., Zhang, T., Jia, Y., Sun, J., Xu, F., & Xu, W. (2017). DataLab: Introducing Software Engineering Thinking into Data Science Education at Scale. In *ICSE-SEET* (pp. 47–56). IEEE Computer Society.

Zheng, S., Ristovski, K., Farahat, A., & Gupta, C. (2017). Long Short-Term Memory Network for Remaining Useful Life Estimation. *ICPHM*.