Towards a Generic Framework for Mechanism-guided Deep Learning for Manufacturing Applications

Hanbo Zhang*
School of Computer Science
Fudan University
Shanghai, China
zhanghb16@fudan.edu.cn

Jiangxin Li
School of Computer Science
Fudan University
Shanghai, China
jiangxinli21@m.fudan.edu.cn

Shen Liang†
Data Intelligence Institute of Paris
Université Paris Cité
Paris, France
shen.liang@u-paris.fr

Peng Wang
Shanghai Key Laboratory of Data Science, School of Computer Science
Fudan University
Shanghai, China
pengwang5@fudan.edu.cn

Themis Palpanas
LIPADE, Université Paris Cité
& French University Institute (IUF)
Paris, France
themis@mi.parisdescartes.fr

Chen Wang
NELBDS, EIRI, School of Software
Tsinghua University
Beijing, China
wang_chen@tsinghua.edu.cn

Wei Wang
Shanghai Key Laboratory of Data Science, School of Computer Science
Fudan University
Shanghai, China
weiwang1@fudan.edu.cn

Haoxuan Zhou
School of Mechanical Engineering
Xi'an Jiaotong University
Xi'an, China
Energy Department
Politecnico di Milano
Milano, Italy
hxzhou@stu.xjtu.edu.cn

Jianwei Song
Hudong-Zhonghua Shipbuilding (Group) Co,Ltd.
Shanghai, China
13611889322@126.com

Wen Lu
Hudong-Zhonghua Shipbuilding (Group) Co,Ltd.
Shanghai, China
seyluu@163.com

ABSTRACT

Manufacturing data analytics tasks are traditionally undertaken with Mechanism Models (MMs), which are domain-specific mathematical equations modeling the underlying physical or chemical processes of the tasks. Recently, Deep Learning (DL) has been increasingly applied to manufacturing. MMs and DL have their individual pros and cons, motivating the development of Mechanism-guided Deep Learning Models (MDLMs) that combine the two. Existing MDLMs are often tailored to specific tasks or types of MMs, and can fail to effectively 1) utilize interconnections of multiple input examples, 2) adaptively self-correct prediction errors with error bounding, and 3) ensemble multiple MMs. In this work, we propose a generic, task-agnostic MDLM framework that can embed one or more MMs in deep networks, and address the 3 aforementioned issues. We present 2 diverse use cases where we experimentally demonstrate the effectiveness and efficiency of our models.

CCS CONCEPTS

• **Applied computing** → **Industry and manufacturing**; • **Computing methodologies** → *Artificial intelligence.*

KEYWORDS

manufacturing; mechanism model; deep learning

ACM Reference Format:
Hanbo Zhang, Jiangxin Li, Shen Liang, Peng Wang, Themis Palpanas, Chen Wang, Wei Wang, Haoxuan Zhou, Jianwei Song, and Wen Lu. 2023. Towards a Generic Framework for Mechanism-guided Deep Learning for Manufacturing Applications . In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August*

*Hanbo Zhang and Jiangxin Li are co-first authors with equal contributions.
†Shen Liang is the corresponding author.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599913

6–10, 2023, Long Beach, CA, USA. ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599913

1 INTRODUCTION

Manufacturing entails numerous data analytics tasks. For instance, in shipbuilding, the task of **Mechanical Operation State Prediction (MOSP)** [24] for assembly machinery is critical to safe operations, while the task of **Melt Viscosity Prediction (MVP)** [22] for alloy melts is important to marine grade steel plate production. Such tasks are traditionally undertaken with **Mechanism Models (MMs)** [49], formally defined as follows.

Definition 1. ***Mechanism Model (MM):*** *For a manufacturing task with input and output vectors* $\mathbf{x}$ *and* $\mathbf{y}$*, an MM is an equation*

$$\hat{\mathbf{y}}^M = f(\mathbf{x}, \theta) \tag{1}$$

which is created by domain experts to model the underlying physical or chemical processes of the task, with $\hat{\mathbf{y}}^M$ *being the MM's prediction for* $\mathbf{y}$*, and* θ *being unknown parameters to estimate for specific datasets. The process of estimating* θ *is called* ***Parameter Identification****.*

For instance, an MM for MOSP is the Wiener-process MM [24] (detailed in Section 5), while an MM for MVP is the VFT MM [42]:

$$\hat{y} = A_{VFT} + \frac{(12 - A_{VFT})(T_g - C_{VFT})}{x - C_{VFT}} \tag{2}$$

where x is the temperature, $\hat{y}$ is the base-10 logarithm of the predicted viscosity, and A_{VFT}, C_{VFT}, T_g are parameters to identify.

MMs are highly interpretable and are generally robust against small data, as they are rooted in domain knowledge and the number of unknown parameters is usually small. Thus, they remain widely used to this day [22–24, 44, 49]. However, their performance can be limited by 2 factors: 1) MMs may not be accurate enough to model complex real-world processes due to the limitations of the domain knowledge they are built on. 2) Parameter identification for MMs traditionally relies on classic numerical optimization methods (e.g. least squares [49]), which often yield sub-optimal parameters.

Recently, **Deep Learning (DL)** has been increasingly applied to manufacturing [12, 25]. Unlike MMs which use explicit domain knowledge, DL models directly learn from raw data, and are often more accurate on large datasets. However, they still widely suffer from low interpretability and sensitivity to small data, despite recent advances in trying to solve these problems [26, 46].

Faced with both the pros and cons of MMs and DL, researchers are now bringing them together with **Mechanism-guided Deep Learning Models (MDLMs)**, which embed MMs in deep networks. Most existing MDLMs follow one of 2 paths: 1) DL for MM parameter identification [11, 20, 22, 34, 43, 49], replacing traditional numerical optimization methods. 2) DL for approximation of unsolvable/undifferentiable MMs [11, 20, 34, 43]. Existing MDLMs are often tailored to specific tasks or types of MMs. Also, they are often unable to fulfill the following 3 *requirements* of accurate mechanism-guided DL.

R1: Utilization of multi-example interconnections. Manufacturing often entails interconnected data examples. For instance, under the Wiener-process MM [24] (detailed in Section 5), the MOSP task is formulated as a single-in-single-out problem: the MM takes in the Operation State Value (OSV) at the current timestamp and predicts the next OSV, both of which are scalars. However, the OSVs are not isolated from each other. Rather, they are interconnected data examples that jointly form a high-level data example: a time series. Existing MDLMs are often restricted by the rigid input-output format of the MM. In this case, the Wiener-process MM only considers the pair of input-output scalars, ignoring their connections with OSVs in their vicinity, which can negatively impact the performance of the model as potentially vital context information (e.g. the shape of the OSV time series) is missing. In response, we need MDLMs that can capture the interconnections of multiple low-level data examples (e.g. individual OSVs) by learning from the high-level data example they form (e.g. the OSV time series).

R2: Adaptive and bounded error correction. MMs are often (over-)simplified approximations of complex processes, thus requiring additional error correction. However, most MMs either do not have built-in error terms, or make rigid assumptions on the error distribution (e.g. that being Gaussian [24]), lacking adaptiveness to real-world data. While there are some efforts to address errors in MMs [8, 49], existing methods tend to lack the ability to adaptively bound the error terms so that the users can easily observe or control the amount of error.

R3: Multi-MM Ensembling. Many manufacturing tasks (e.g. the MVP task [22]) can be tackled by multiple MMs. Intuitively, an MM Ensemble (MME) can boost prediction accuracy. While the idea of MMEs has been explored both in [13] and out of [21, 30, 31] manufacturing, such studies remain limited in the context of task-agnostic end-to-end MDLMs. Building an MME is especially challenging in that unlike in conventional ensembles where the ensemble members are independent of each other, in an MME, multiple MMs can have shared parameters. Sub-optimal identification of these shared parameters can negatively affect multiple MMs and thus the overall accuracy of the MME.

To fulfill these requirements, in this work we propose 3 *solutions*:

S1: Deep parameter identification with awareness of high-level data examples. To fulfill **R1**, we note that while the MMs themselves have rigid formats, there are no restrictions on how to identify their parameters. Thus, we follow the path of DL-based parameter identification. Unlike existing deep parameter identification methods [22, 49], we capture interconnections of low-level data examples from their corresponding high-level examples, which is done by stratifying both the input examples and the MM parameters, and mapping the two with a carefully-designed protocol.

S2: Adaptive error correction with error bounding. To fulfill **R2**, we design a sub-network for error estimation with no rigid assumptions on its distribution, and can bound the estimated error to an interval that is either user-defined or automatically learned.

S3: Attentional multi-MM ensemble with penalty to sub-optimal parameter identification. To fulfill **R3**, we use the attention mechanism [40, 48] to weight multiple MMs, thus building a deep MM ensemble (MME). We also introduce a simple-yet-effective penalty term in the loss function to guard against sub-optimal parameters, which can play a pivotal role to ensure model accuracy in the presence of shared parameters.

The main contributions of this work are summarized below.

- We present a generic, task-agnostic framework for mechanism-guided DL, which is comprised of 1) a Single-Mechanism-guided Deep Learning Model (S-MDLM) and 2) a Multi-Mechanism-guided Deep Learning Model (M-MDLM).

Table 1: Main Abbreviations and Notations

Symbol	Meaning
AA	Attentional Aggregator
CP	Constant Parameter
DL	Deep Learning
ECM	Error Correction Module
EE	Error Estimator
HE/LE	High/Low-level Example
HF/LF	High/Low-level Feature
HFL/LFL	High/Low-level Feature Learner
HPI/LPI	High/Low-level Parameter Identifier
HVP/LVP	High/Low-level Variable Parameter
MDLM	Mechanism-guided Deep Learning Model
MM	Mechanism Model
MME	Mechanism Model Ensemble
MOSP	Mechanical Operation State Prediction
MVP	Melt Viscosity Prediction
PIM	Parameter Identification Module
RMSE	Root Mean Square Error
S/M-MDLM	Single/Multi-Mechanism-guided Deep Learning Model
SU	Semantic Union
a, b	Parameters to identify in the Wiener-process MM
$\mathbf{w}$	Attentional weight vector in an AA
$\mathbf{x}$	Model input vector
$\mathbf{y}$	Groundtruth of the vector to predict
$\hat{\mathbf{y}}$	Predicted vector by an MDLM
$\hat{\mathbf{y}}^{\mathbf{E}}$	Uncorrected predicted vector by an MME
$\hat{\mathbf{y}}^{\mathbf{M}}$	Uncorrected predicted vector by an MM
$\hat{\mathbf{y}}^{\mathbf{M}}_{\mathbf{i}}$	Predicted vector by the i-th MM in an MME
$\boldsymbol{\beta}$	Error bound vector in an EE
$\boldsymbol{\Theta}$	Parameters to identify in an MM

- Addressing the aforementioned key requirements **R1**-**R3**, we implement the aforementioned solutions **S1**, **S2** in both the S-MDLM and M-MDLM, and **S3** in the M-MDLM.
- We instantiate our framework in 2 use cases: Mechanical Operation State Prediction (MOSP) [24] and Melt Viscosity Prediction (MVP) [22], where our models can yield significantly more accurate results than their best competitors.
- All our source code is available at https://github.com/sliang11/MDLM for reproducibility.

In the rest of the paper, Section 2 reviews related work. Sections 3 and 4 introduce S-MDLM and M-MDLM. Sections 5 and 6 present the use cases of MOSP and MVP. Section 7 concludes the paper. Table 1 shows the main abbreviations and notations in this paper.

2 RELATED WORK

Manufacturing analytics tasks (e.g. MOSP [24], MVP [22], power demand forecasting in smelting [49], state-of-charge estimation for batteries [47], etc.) traditionally rely on domain-specific MMs. whose parameters are identified by numerical optimization (e.g. least squares [49], maximum likelihood estimation [24], etc.). MMs are generally highly interpretable and robust against small data, yet they are often inaccurate on complex real-world data, with traditional parameter identification methods yielding sub-optimal results. Recently, DL has been increasingly adopted in tasks such as mechanical fault diagnosis [25] and distortion prediction in laser-based additive manufacturing [12]. Rather than using explicit domain knowledge, DL models directly learn from raw data. While DL models often perform better on large data, they widely suffer from limited interpretability and sensitivity to small data.

Bringing together MMs and DL, researchers have now proposed Mechanism-guided Deep Learning Models (MDLMs) which embed MMs in deep networks. This is mainly achieved in two ways: 1) DL for MM parameter identification. For instance, Yang et al. [49] used a neural network to identify the parameters of an MM that predicts power demand in smelting. Le Losq et al. [22] used deep multi-task learning to identify the parameters of MMs that predict melt viscosity. Physics-Informed Neural Networks (PINNs) [8, 11, 20, 34, 43] can be used to identify parameters in Partial Differential Equations (PDEs). 2) DL for approximation of unsolvable/undifferentiable MMs. For instance, PINNs are proposed to provide approximate solutions to PDEs. Existing MDLMs are often tailored to specific tasks or types of MMs. Also, they are often unable to fulfill the requirements **R1**-**R3** mentioned in Section 1, motivating us to propose a novel generic MDLM framework.

3 SINGLE-MECHANISM-GUIDED DEEP LEARNING MODEL (S-MDLM)

We now present our generic MDLM framework, beginning with the **Single-Mechanism-guided Deep Learning Model (S-MDLM)** which entails only one MM. Concretely, we will first discuss the architecture of an S-MDLM, and then discuss its loss function.

3.1 The S-MDLM Architecture

Shown in Fig. 1, an S-MDLM is an end-to-end network with 1) an MM as its core, 2) a **Parameter Identification Module (PIM)** which identifies the MM parameters, and 3) an **Error Correction Module (ECM)** which corrects errors in the MM's predictions. We now discuss these 3 parts one by one.

3.1.1 Mechanism Model (MM). As the core module for making predictions, many MMs are themselves differentiable [22, 24] and can be directly embedded into an S-MDLM. For an undifferentiable MM, we can simply approximate each of its undifferentiable terms $\mathbf{z} = \tau(\mathbf{q})$ with a mini-network, generate a set of $(\mathbf{q}, \mathbf{z})$ pairs to train it on, and embed it into the S-MDLM with all its parameters hard-coded. Besides this simple approach, we can also use more advanced methods such as PINNs [8, 11, 20, 34, 43].

3.1.2 Parameter Identification Module (PIM). The PIM uses subnetworks called **Feature Learners (FLs)** to learn features from the input, and **Parameter Identifiers (PIs)** to identify MM parameters from the features. As was mentioned in Section 1, in the PIM, we utilize the interconnections of multiple **Low-level Examples (LEs)** (i.e. the MM's raw input $\mathbf{x}$), which are often encoded in the **High-level Examples (HEs)** formed by the LEs. Thus, we make the HEs part of the PIM input to retain the interconnections. For instance, in the MOSP task mentioned in Section 1, the interconnections of individual OSVs are encoded in the OSV time series, and can be captured by having the latter as an input.

Note that it is not trivial to make the HEs part of the PI inputs, as we need to solve the following 2 problems:

- **HE acquisition:** How to obtain the HEs from the LEs?
- **Example-to-Parameter (E2P) mapping:** The parameters in an MM can often be heterogeneous, and it is thus not advisable to use all LEs and HEs to identify every parameter. Therefore, for each MM parameter, which LEs and/or HEs should be used to identify it?

We now discuss these 2 problems in detail.

1) HE acquisition. We obtain the HEs by a user-defined **Semantic Union (SU)** operation that details how LEs form an HE. We now give our recommended SU definitions for the most common types of

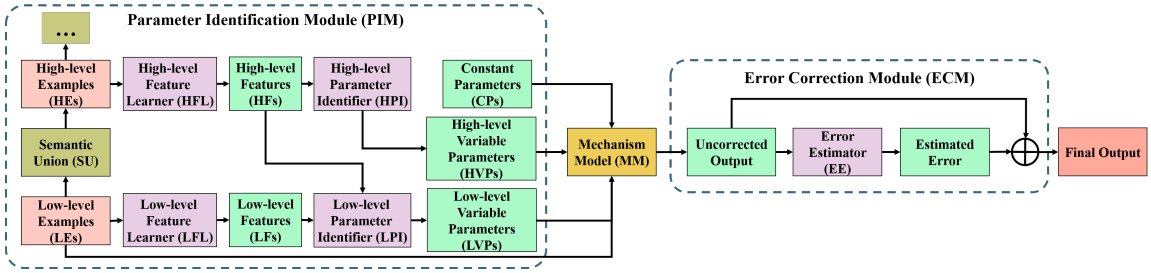


Figure 1: The architecture of a Single-Mechanism-guided Deep Learning Model (S-MDLM), an end-to-end network with a single MM as its core, and two modules for parameter identification and error correction. For simplicity, here we only show two levels of data examples. We can generalize to more than 2 levels of examples by multiple iterative Semantic Union (SU) operations.

LEs in manufacturing, which can be used *off the shelf*: if the LEs are data points at individual timestamps or time series subsequences, SU outputs the time series they form; if the LEs are pixels or image patches, SU outputs the image they form; if the LEs are video frames or snippets, SU outputs the video they form; if the LEs are 3D points or point clusters, SU outputs the point cloud they form. With the LEs and HEs obtained, we use 2 user-defined FLs, a **Low-level FL (LFL)** and a **High-level FL (HFL)**, to learn **Low-level Features (LFs)** and **High-level Features (HFs)** from them.

It is worth noting that the HEs are usually not static. In fact, an HE can be split when partitioning the training and testing sets, or grow as more LEs arrive in online prediction. For instance, as will be shown in Section 5, in the aforementioned MOSP task, the LEs of a machine are its OSVs from its first operation to total failure. Here we cannot use *all* these OSVs to form the OSV time series used as the HE, as we would need to wait till the machine is totally broken, which defeats the purpose of MOSP. In reality, the machine may have already been running for some time when we train the MOSP model, with some historical OSVs as training LEs. Once the model is trained, we use it to make online predictions for the machine, always forecasting the next OSV when a new OSV arrives.

In such cases, we feed into the HFL *part of* the entire HE formed by the *currently available* LEs. For instance, for MOSP, in training we feed into the HFL the OSV time series with only the historical OSVs; in online prediction, we extend this time series at each timestamp by appending the new OSV to its end. Note that this requires the HFL to support variable-sized input, for the HE will grow as more LEs arrive. Such HFL architectures exist for the most common types of HEs in manufacturing. For instance, RNNs [16] can handle variable-length time series, FCN [27] can handle variable-sized images, DGCNN [45] can handle variable-sized point clouds, etc.

2) E2P mapping. For E2P mapping, we draw on the parameters' *invariance* to the LEs and HEs to stratify them into 3 levels:

- **Low-level Variable Parameters (LVPs)**: parameters whose values vary for different LEs.
- **High-level Variable Parameters (HVPs)**: parameters whose values are fixed for LEs in a single HE, but vary for different HEs.
- **Constant Parameters (CPs)**: parameters whose values are fixed for both LEs and HEs.

We will show how to decide the level of each parameter later. For now, we describe how to map the LEs (LFs) and HEs (HFs) to the 3 levels of parameters as follows:

- For *LVPs*, we identify them with a user-defined **Low-level PI (LPI)**, which takes *both the LFs and HFs* as its input. We include the HFs to provide the context information on multi-LE interconnections missing in the LFs.
- For *HVPs*, we identify them with a user-defined **High-level PI (HPI)**, which takes the *HFs* as its input[1]. This is very different from HVP identification in a traditional MM, where the HVPs are derived based on a simple aggregation (e.g. summation [24]) of the LEs that treats them as isolated examples, ignoring their interconnections as encoded in the HEs (and HFs).
- For *CPs*, as they are independent of both the LEs and HEs, we use no explicit PI to identify them. Rather, we treat them as learnable parameters of the entire S-MDLM, directly feeding them to the MM and fine-tuning them with the rest of the network.

On how to decide the level of a parameter, one can draw on its physical meaning. For instance, as will be shown in Section 5, a Wiener-process MM [24] for the aforementioned MOSP task has 2 parameters: a which describes the variation of the degradation rates of different machines, and b which is a constant value describing the similarity of their degradation trends. Here the physical meaning of b explicitly specifies that it is a CP; as with a, note that the degradation rate of a machine can be derived from its OSV time series, which is its corresponding HE. From the physical meaning of a, we know that it varies for different machines but is fixed for a single machine. Thus, it is an HVP as it varies only for different HEs. In the rare case where we cannot derive the level of a parameter, we treat it as an LVP to maximize adaptiveness to varying LEs. Also note that if there are multiple parameters on a certain level, rather than using multiple PIs to identify them individually, we opt to use a unified PI to identify them all, as this simpler structure can make the network more robust, especially against small data.

Before moving on, we note that sometimes there are no obvious interconnections among LEs to justify forming HEs from them. In this case, there are no HEs, HVPs, HFL, HFs, or HPI. On the other hand, we can generalize to the (rare) cases where more than 2 levels of examples exist using multiple iterative SU operations, with an FL learning features from each level of examples, and a PI taking in the features from this level and all levels above it to identify the parameters on this level.

[1]Note that in the online prediction phase, as we extend the HFL input, the learned HFs, and thus the identified values of the HVPs, will change for different LEs. This seems to contradict the definition of the HVPs, which should remain fixed for all LEs from the same HE. However, note that the *identified* HVP values are *not* the same as the *groundtruth* HVP values (which are latent values that cannot be known). In fact, as the HFL input gets extended in online prediction, we are implicitly refining the HFL and HPI with the additional information, in the hope that the *identified* HVP values will converge to the *groundtruth* ones.

3.1.3 Error Correction Module (ECM). The ECM addresses the inability of MMs to effectively self-correct prediction errors (cf. Section 1), in which the MM's uncorrected prediction $\hat{\mathbf{y}}^M$ is passed on to a sub-network called the **Error Estimator (EE)** to obtain an error term vector $\mathbf{e}$ with same length as $\hat{\mathbf{y}}^M$, which is added to $\hat{\mathbf{y}}^M$ to yield the final output $\hat{\mathbf{y}}$. The EE architecture is

$$\mathbf{e} = \boldsymbol{\beta} \odot \tanh\big(g(\hat{\mathbf{y}}^M)\big) \tag{3}$$

where $\odot$ denotes element-wise multiplication, $g(\cdot)$ is a Multi-Layer Perceptron (MLP) that outputs a vector $\boldsymbol{\epsilon}$ with the same length as $\mathbf{e}$, the tanh function maps each term in $\boldsymbol{\epsilon}$ to $(-1, 1)$, and $\boldsymbol{\beta}$ is the error bound vector that ensures the absolute value of each term in $\mathbf{e}$ does not exceed that of its corresponding term in $\boldsymbol{\beta}$. $\boldsymbol{\beta}$ can either be pre-set by the user by domain knowledge, or be a learnable vector. For an MM with built-in error terms, we remove them when embedding the MM into an S-MDLM, substituting them with those estimated by the EE. This is to avoid the often unrealistic assumptions (e.g. the error being Gaussian [24]) underlying the built-in error terms.

3.2 The S-MDLM Loss Function

As with the loss function. for each example (LE, to be more specific) $\mathbf{x}$ with groundtruth label $\mathbf{y}$, the loss is written as

$$\mathcal{L}^{\mathcal{S}} = \ell(\mathbf{y}, \hat{\mathbf{y}}) + \lambda \ell(\hat{\mathbf{y}}, \hat{\mathbf{y}}^M) \tag{4}$$

where $\ell(\cdot)$ is a template loss function chosen by the user (e.g. L2, Cross Entropy, etc.), and λ is a hyperparameter. In the S-MDLM loss, the first term is the main loss, while the second is a penalty term preventing the absolute values of the estimated errors $\mathbf{e} = \hat{\mathbf{y}} - \hat{\mathbf{y}}^M$ from being too large. The S-MDLM is trained with standard back-propagation to minimize the total loss over all training examples.

4 MULTI-MECHANISM-GUIDED DEEP LEARNING MODEL (M-MDLM)

We now extend the S-MDLM to a Multi-Mechanism-guided Deep Learning Model (M-MDLM) with a multi-MM ensemble. As far as we know, this is the first attempt to ensemble multiple MMs. Next, we will discuss the M-MDLM architecture and its loss function.

4.1 The M-MDLM Architecture

The M-MDLM architecture is shown in Fig. 2, which is an end-to-end network extended from the S-MDLM architecture in Fig. 1. Like an S-MDLM, an M-MDLM has 3 parts: 1) an MM Ensemble (MME) as its core, setting it apart from the single MM in an S-MDLM; 2) a PIM to identify the MM parameters; 3) an ECM to correct prediction errors of the MME. We now present the 3 parts one by one.

4.1.1 Mechanism Model Ensemble (MME). The MME entails multiple MMs making individual predictions, which are aggregated by a sub-network called the **Attentional Aggregator (AA)**. Specifically, AA adopts the widely-used attention mechanism [40, 48] to weight the individual MM predictions and obtain their weighted average as the MME prediction. Formally, from a matrix $\hat{\mathbf{Y}}^M$ where each column is the prediction by one of m MME members, we obtain an m-dimensional weight vector $\mathbf{w}$ with

$$\mathbf{w} = \text{softmax}(h(\hat{\mathbf{Y}}^M)) \tag{5}$$

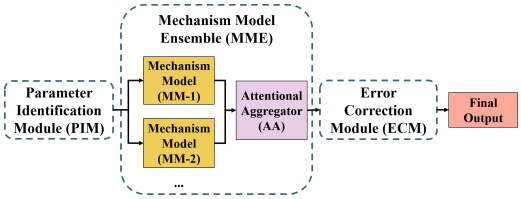


Figure 2: The architecture of a Multi-Mechanism-guided Deep Learning Model (M-MDLM), which is an end-to-end network extended from the S-MDLM architecture (Fig. 1), with an MM Ensemble (MME) as its core and two modules for parameter identification and error correction.

where $h(\cdot)$ is an MLP and the softmax function normalizes the weights to have a sum of 1. The weighted average $\hat{\mathbf{y}}^E$ of the individual predictions is the unified prediction by the MME.

4.1.2 Parameter Identification Module (PIM). The PIM of an M-MDLM is largely the same as that of an S-MDLM. The only caveat is that multiple MMs can share some parameters. As we will show later, this fact is critical to the design of the M-MDLM loss function.

4.1.3 Error Correction Module (ECM). We inherit the ECM of the S-MDLM, namely use the EE in Eq. 3 to estimate the error, and add it to the uncorrected MME prediction to obtain the final output.

4.2 The M-MDLM Loss Function

As with the loss function, for each training example $\mathbf{x}$, with m individual MMs in the MME, the loss function is written as

$$\mathcal{L}^{\mathcal{M}} = \ell(\mathbf{y}, \hat{\mathbf{y}}) + \lambda_1 \ell(\hat{\mathbf{y}}, \hat{\mathbf{y}}^E) + \lambda_2 \sum_{i=1}^{m} \ell(\mathbf{y}, \hat{\mathbf{y}}_i^M) \tag{6}$$

where $\mathbf{y}$ is the groundtruth, $\hat{\mathbf{y}}$ is the prediction by the M-MDLM, $\hat{\mathbf{y}}^E$ is the uncorrected prediction by the MME, $\hat{\mathbf{y}}_i^M$ is the individual prediction made by the i-th MM in the MME, and ℓ is a template loss function chosen by the user (e.g. L2, Cross Entropy, etc.).

In the M-MDLM loss, the first 2 terms serve the same purpose as the 2 terms in the S-MDLM loss (Eq. 4), namely being the main loss and limiting the error term. What is unique to the M-MDLM loss is its third term. Inspired by a similar design in a conventional DL ensemble [14], this term prevents *individual* MMs in the MME from being too weak. Specifically, a weak MM can be due to 1) *weak mechanism*, that is, the MM is inherently weak even with optimal parameter identification, or 2) *weak parameters*, that is, the MM performs poorly due to sub-optimal parameters. The latter is especially problematic for an M-MDLM where parameter identification for multiple MMs is done in an end-to-end fashion, and sub-optimal parameter settings for one MM can negatively affect those of other MMs. Critically, as was previously mentioned, some MMs can share parameters. Sub-optimal settings of such parameters can *directly* impact multiple MMs, significantly weakening the entire MME. While the AA can suppress the problem of *weak mechanism* by assigning a small weight to the weak MM, it cannot handle the problem of *weak parameters*, as it has no direct influence on the performance of *individual* MMs. By contrast, by simply summing up their prediction errors, the third term in the loss function forces the *individual* MMs to perform as well as possible, and the only way to do so is with better parameter identification. As will be shown in Section 6.2, this can be the *single most important* design feature ensuring an M-MDLM's accuracy.

5 CASE STUDY: MECHANICAL OPERATION STATE PREDICTION (MOSP)

Having presented our generic S-MDLM and M-MDLM, we now instantiate them in specific analytics tasks. We begin with an S-MDLM for MOSP. Specifically, from raw sensor readings (e.g. vibration signals) of a machine, we can derive a single-variate time series of Operation State Values (OSVs) quantifying its operation states over time. We defer discussion on how we specifically obtain the OSVs to Section 5.2. For now, it suffices to know that roughly speaking, the larger the OSV, the closer the machine is to total failure. Our task at a given timestamp is to predict the OSV at the next timestamp. Here we use the following Wiener-process MM [24]:

Definition 2. ***Wiener-process MM:*** *For a machine whose OSV at the current timestamp is x, its next OSV y is predicted as*

$$\hat{y}^M = x + \pi(x; a, b)(t_{k+1} - t_k) + \sigma_B \cdot B(t_{k+1} - t_k) \tag{7}$$

where $\pi(x; a, b) = ax^b$ [23] and $\sigma_B \cdot B(t_{k+1} - t_k)$ is a built-in error term with $B(\cdot)$ being a standard Brownian motion process and t_k, t_{k+1} being the current and next timestamps. The parameter a is the Unit-to-Unit-Variance (UtUV) parameter describing the variation of the degradation rates among different units (i.e. machines), while b and σ_B are constant parameters describing the similarity of the degradation trends among different units.

Next, we will present **DeepWiener**, our S-MDLM based on the Wiener-process MM, and experimentally validate its effectiveness.

5.1 DeepWiener: S-MDLM for MOSP

We design DeepWiener by strictly following the generic S-MDLM framework in Section 3 for both its architecture and loss function.

5.1.1 The DeepWiener Architecture. Following the generic S-MDLM architecture in Fig. 1, DeepWiener entails an MM, a PIM and an ECM.

For the MM, following what was mentioned in Section 3.1, we remove the built-in error term $\sigma_B \cdot B(t_{k+1} - t_k)$ from the original Wiener-process MM to avoid rigid assumptions about the error (in this case, the original Wiener-process MM assumes the error follows the Brownian movement which is Gaussian). We use the remainder of the MM as the network core, which is fully differentiable and requires no changes. Note that the removal of the built-in error term means the parameter σ_B is also removed, leaving only a and b as the parameters to identify.

For the PIM, each LE is an individual OSV. As with the HEs, as was mentioned in Section 3, for each machine, we use the time series of all its historical OSVs as the HE for training, and gradually extend the time series with newly-arrived OSVs for online prediction. As for the parameters, as was also mentioned in Section 3, a is an HVP and b is a CP. Thus, we use an HFL and an HPI to identify a from the HE, and let b be a learnable parameter for the entire network with no explicit sub-network to identify it. Note that the Wiener-process MM has no LVPs, thus there is no LFL or LPI in DeepWiener. Despite the absence of LVPs, we note that we are able to utilize the interconnections of the LEs (i.e. the OSVs) in our identification of the HVP a, especially when compared to the way a is identified in the original Wiener-process MM paper [24], which deduces a based on a simple summation of all training OSVs (with some postprocessing) without using their interconnections. By contrast, we learn such interconnections from the OSV time series they form. As for the sub-network architectures, the HFL of DeepWiener is a Recurrent Neural Network (RNN) with 2 hidden layers, each with 16 neurons; the HPI is an MLP with a single 16-dimensional hidden layer, followed by a PReLU activation.

For the ECM, the EE is an MLP with the same architecture as the HPI. We treat the error bound as a learnable parameter.

5.1.2 The DeepWiener Loss Function. The DeepWiener loss instantiates the S-MDLM loss in Eq. 4 with $\ell(\cdot)$ being the L2 loss.

5.2 Experimental Study

We now experimentally evaluate DeepWiener, presenting the experimental settings and the results.

5.2.1 Experimental Settings. For experimental settings, we first present the 3 datasets we use:

- **NASA** [38]: a simulated dataset of the degradation of 100 turbofan engines, each with readings of multiple sensors that we convert into an OSV time series with the method proposed in [24, 35] after data cleaning and normalization (detailed in Appendix). We use the first half of each OSV time series for training and validation, and the rest for testing, acquiring 6,071 and 6,125 training and testing OSVs in total.
- **XJTU-SY** [44]: a real-world dataset of the run-to-failure process of 15 bearings. For each bearing, the original data entails its horizontal and vertical vibration signals which we treat separately. To obtain the OSV time series, we follow the advice of a domain expert and discard all data at the beginning of each signal which exhibits no obvious degradation (cf. Appendix). Next, a 1,280ms snippet is sampled every 1min from the remainder of each signal. The root mean square of the snippet is used as its corresponding OSV. We use the first half of each OSV time series for training and validation, and the rest for testing, acquiring 744 training and 736 testing OSVs from the 15 horizontal signals, as well as 1,488 training and 1,479 testing OSVs from the 15 vertical signals. We train different models for the horizontal and vertical signals.
- **HZ**: a private dataset from Hudong-Zhonghua, a world-class shipbuilder, entailing 4 machines used in its operations. For each machine, a domain expert manually scored its operation state once a day, thus obtaining 4 OSV time series. Using the first half of each time series for training and validation, and the rest for testing, we end up with 71 training and 70 testing OSVs.

For the rest of this section, we call each of the 100 engines in NASA, the 30 signals in XJTU-SY and the 4 machines in HZ a *unit*.

For rival methods, we consider 13 options, divided into 3 groups:

- **Wiener** [24]: the original Wiener-process MM, whose parameters are identified using the maximum likelihood estimation and optimal search method proposed by the original authors;
- Support Vector Regression (**SVR**) [10], Random Forest (**RF**) [4], **XGBoost** [5], Multi-Layer Perceptron (**MLP**) [36]: 4 general-purpose, data-driven (i.e. domain knowledge-free) regressors;
- **ARIMA** [3], **ROCKET** [9], **DeepAR** [37], **RNN** [16], **FCN** [17, 28], **InceptionTime** [19], **ResNet** [15, 18], **Informer** [50]: 8 data-driven models specialized for time series data.

Concerning input data, as was mentioned previously, DeepWiener uses the time series of all known OSVs as its HE. Specifically, in training, for each OSV from the ω-th unit used as an LE, we use

the time series of all training OSVs from the ω-th unit as the HE; in testing, for each OSV $\mathbf{x}$ from the ω-th unit used as an LE, we use the time series of all known OSVs up to $\mathbf{x}$ (including the training OSVs) from the ω-th unit as the HE. For Wiener, it can only take a single OSV as its input to predict the next OSV. For the data-driven methods, we formulate MOSP as a time series prediction problem. Specifically, at timestamp i, we let the methods use the OSV time series at timestamps $i-u+1$ to i to predict the OSVs at timestamps $i+v$ to $i+v+w-1$, where $u \in \{1,2,4,8\}$, $v \in \{1,2,4,8\}$ and $w \in \{1,2,4,8\}$ is chosen with 5-fold CV.

We implemented DeepWiener with PyTorch [32], with the initial values of b and β chosen from $\{0.1, 0.3, 0.5, 1\}$ and $\{0.01, 0.1\}$, and λ chosen from $\{0.1, 1, 5, 10\}$ by 5-fold CV, *except* for on HZ where we set λ to 0, following the advice of a domain expert to counter the high complexity of the data which calls for more adaptiveness introduced by the deep ECM. Using the AdamW optimizer, we set the initial learning rate to 0.02 with a reduce-on-plateau policy with a factor of 0.5, a patience of 10, and a minimum learning rate of 0.0001. We train the network for up to 100 epochs with an early-stop policy with a patience of 10 and delta chosen from $\{0.01, 0.1\}$. For the rival methods, we largely follow the implementations by either their original authors or third-party packages with these methods built-in, with minor changes to make them better suited to our data (see Appendix for details). We ran our experiments on an Ubuntu 18.04.4 server with an Intel Xeon Gold 6326 CPU @2.90GHz, 256GB RAM and an NVIDIA GeForce RTX 3090 GPU. We ran all neural network models and XGBoost on GPU, and ran the rest on CPU. All results were averaged over 10 runs.

5.2.2 Prediction Accuracy. We now assess the accuracy of DeepWiener and its rivals, starting by measuring their **Root Mean Square Errors (RMSEs)** on all testing examples. The smaller the RMSE, the more accurate the method. The results are shown in Fig. 3, where both Wiener and DeepWiener are far more accurate than their data-driven rivals, including RNN which is the most comparable data-driven method to DeepWiener as it is nearly identical to the DeepWiener HFL. This highlights the advantage of MMs in domain-specific use cases. Moreover, on average, DeepWiener reduces the RMSE of Wiener by a large margin of 10.9% on NASA, and 8.2% on XJTU-SY. On HZ, this margin is narrowed to 3.4%. However, note that the size of HZ is very small as compared to NASA and XJTU-SY, and MMs such as Wiener tend to perform better on very small data. The fact that DeepWiener still manages to beat Wiener on HZ highlights the advantage of MDLMs.

5.2.3 Ablation Studies. We now use ablation studies to examine key design features of DeepWiener (and S-MDLM in general), i.e. the PIM and ECM. Thus, we devise the following changes to it.

- **Na**: replacing the identified value of a by DeepWiener with the value identified by the Wiener;
- **Nb**: replacing the identified value of b by DeepWiener with the value identified by the Wiener;
- **NE**: removing the ECM.

Mixing and matching these changes, we have a total of $2 \times 2 \times 2 = 8$ ablation methods (including DeepWiener itself). Fig. 4 shows their RMSEs over all testing examples. As is indicated, the most important contributing factor to DeepWiener's high accuracy on NASA and XJTU-SY is deep parameter identification, especially for b, as is evidenced by the large gap between of DeepWiener and

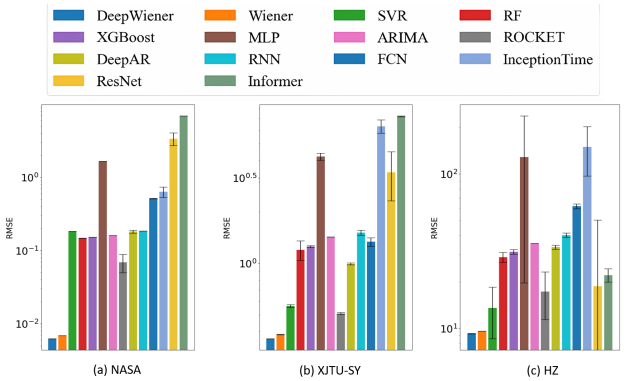


Figure 3: Root Mean Square Errors (RMSEs) of DeepWiener and its rivals over all testing examples (mean ± std over 10 runs). Note that the RMSE axes are in log scale.

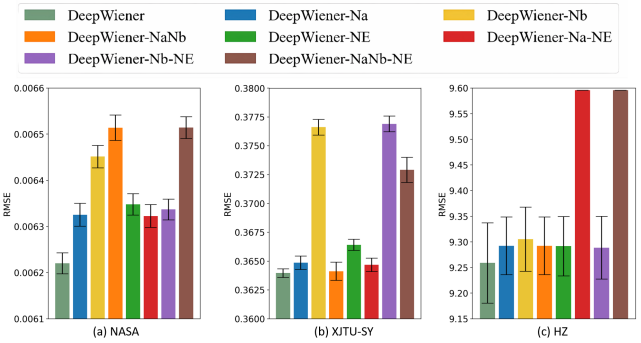


Figure 4: RMSEs of all ablation methods over all testing examples (mean ± std over 10 runs).

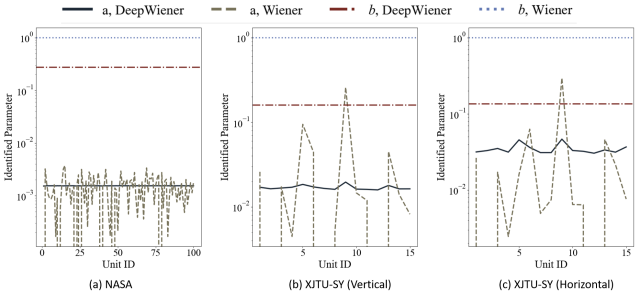


Figure 5: Identified parameter values by DeepWiener and Wiener. The parameter value axes are in log scale (not available for negative parameter values).

DeepWiener-Nb. Also, while DeepWiener-NaNb can have similar (or even better at times) accuracy than DeepWiener on XJTU-SY, its performance is less stable than DeepWiener over the 10 runs. Plus, it performs even worse than DeepWiener-Nb on NASA. Moreover, comparing DeepWiener with DeepWiener-NE, we find that the ECM can effectively mitigate prediction errors. Meanwhile, on HZ, it is the combined effect of of deep parameter identification and error correction that ensures high accuracy.

5.2.4 An Insight Into Parameter Identification. To explore the reason why DeepWiener can perform well, we note that in the ablation studies, parameter identification was key to DeepWiener's high accuracy. This is intuitive as the effect of error correction should be secondary to the MM's own (uncorrected) predictions, which

Table 2: Running Time of DeepWiener (s)

	Training Time	Avg. Prediction Time Per Example
NASA	$6.3 \times 10^{1} \pm 1.5 \times 10^{1}$	$8.5 \times 10^{-4} \pm 5.6 \times 10^{-5}$
XJTU-SY (Vertical)	$2.7 \times 10^{0} \pm 3.9 \times 10^{-1}$	$1.1 \times 10^{-3} \pm 7.9 \times 10^{-5}$
XJTU-SY (Horizontal)	$2.9 \times 10^{0} \pm 6.5 \times 10^{-1}$	$9.2 \times 10^{-4} \pm 5.8 \times 10^{-5}$
HZ	$8.3 \times 10^{-1} \pm 4.2 \times 10^{-1}$	$6.6 \times 10^{-4} \pm 1.4 \times 10^{-5}$

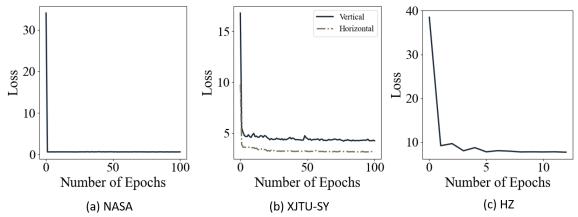


Figure 6: Learning curves of DeepWiener.

are *solely* affected by its parameters. Thus, we take a deeper look at the parameters a and b identified by DeepWiener and Wiener. As is shown in Fig. 5[1], the identified values of both a and b are vastly different for the 2 methods. In particular, we note that theoretically, a should have a positive value, which DeepWiener manages to suffice. By contrast, Wiener occasionally yields negative a values which are sub-optimal. Moreover, as a and b are jointly identified in an end-to-end network, the better identification of a in training can boost that of b, which played the most critical role in ensuring DeepWiener's accuracy in the ablation studies.

5.2.5 Efficiency. While we mainly focus on boosting prediction accuracy in the design of DeepWiener (and S-MDLM in general), we now show that it also has reasonable efficiency. Specifically, the training and average prediction time (per testing example) of DeepWiener is shown in Table. 2. Taking $<$ 100s to train on NASA, $\sim$ 3s on the 2 subsets of XJTU-SY and $\sim$ 1s on HZ, and with a per-example prediction time in the order of 10^{-4} —10^{-3}s, DeepWiener can easily fulfill the requirement of real-world applications. Also, Fig. 6 shows the learning curves (i.e. training losses over the training epochs) of DeepWiener. As is indicated, DeepWiener converges very rapidly, and can thus be trained with relatively few epochs.

6 CASE STUDY: MELT VISCOSITY PREDICTION (MVP)

Having shown the utility of the S-MDLM, we now turn to an M-MDLM in the task of Melt Viscosity Prediction (MVP), which is important to many manufacturing processes where melting of material is required [22]. The input $\mathbf{x}$ of MVP is the mole fractions of the melt components and the temperature. The output is $\hat{y}$ is the base-10 logarithm of the predicted viscosity. Next, we will present **DeepMelt**, our M-MDLM for MVP, and experimentally evaluate it.

6.1 DeepMelt: M-MDLM for MVP

We design DeepMelt by strictly following the generic M-MDLM framework in Section 4 for both its architecture and loss function.

6.1.1 The DeepMelt Architecture. DeepMelt follows the generic M-MDLM architecture in Fig. 2 with an MME, a PIM and an ECM.

[1]We omitted the identified parameters for HZ as these entail confidential information.

For the MME, we use the 5 MMs mentioned in [22]: AG [1], MYEGA [29], AM [2], FVT [6, 7], and VFT [42]. We omit their details for brevity, other than noting that they are all differentiable and require no modifications. Also note that there are several parameters shared by multiple MMs. The MLP in the AA, i.e. $h(\cdot)$ in Eq. 5, has 4 hidden layers, each with 400 neurons and a ReLU activation.

For the PIM, there are a total of 10 MM parameters, whose details we omit for brevity. However, we note that all of them are LVPs. Thus, there are no high-level elements in DeepMelt. The LFL and LPI are MLPs with the same architecture as that in the AA. Also note that 6 of the 10 parameters are shared by multiple MMs, which means we must properly handle the *weak parameter* problem mentioned in Section 4.2, as sub-optimal settings of shared parameters can directly impact multiple MMs.

For the ECM, the MLP in the EE also shares the architecture of the MLP in the AA. The error bound is considered to be learnable.

6.1.2 The DeepMelt Loss Function. The DeepMelt loss instantiates the M-MDLM loss in Eq. 6 with $m = 5$ and $\ell(\cdot)$ being the L2 loss.

6.2 Experimental Study

We now experimentally evaluate DeepMelt, presenting the experimental settings and the results.

6.2.1 Experimental Settings. For the experiments, we use the dataset in [22] which has already been split into training, validation and testing sets with 1198, 261 and 522 examples. Each example is 5-dimensional, comprised of the mole fractions of 4 components and the temperature. On the rival methods, we use 2 groups of methods:

- **iMelt** [22]: an MDLM tailored to the MVP task, using the same 5 MMs in our DeepMelt. Unlike our DeepMelt which uses an MME to make a unified prediction, iMelt only uses a unified network for parameter identification, after which the 5 MMs make their predictions individually. This means iMelt can be divided into 5 sub-methods depending on the MM: **iMelt-AG**, **iMelt-MYAGA**, **iMelt-AM**, **iMelt-FVT**, and **iMelt-VFT**.
- Support Vector Regression (**SVR**) [10], Random Forest (**RF**) [4], **XGBoost** [5], Multi-Layer Perceptron (**MLP**) [36]: 4 general-purpose, data-driven regressors.

We implemented DeepMelt with PyTorch [32], with $\lambda_1 = \lambda_2 = 1$. Using the Adam optimizer, we set the initial learning rate to 0.0006 with a weight decay of 0.1, and adopt the same policy as iMelt [22] to decide when to stop the training. For the rival methods, we largely follow their implementations by either their original authors or third-party packages with these methods built-in, with minor changes to make them better suited to our data (detailed in Appendix). We use the same hardware as we did in Section 5.2, running neural network methods on the GPU and the rest on the CPU, averaging all results over 10 runs.

6.2.2 Prediction Accuracy. On prediction accuracy, the RMSEs over all testing examples are shown in Fig. 7. As is indicated, both the MDLMs, i.e. DeepMelt and iMelt, yield far better results than the data-driven methods. Moreover, by building an ensemble of MMs, DeepMelt has achieved superior average prediction accuracy to iMelt. In particular, compared to its closes rival (in terms of average accuracy), iMelt-MYEGA, DeepMelt reduces the RMSE by a large margin of 14.8% on average.

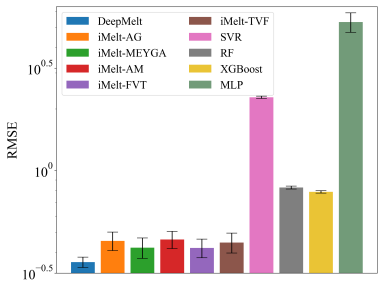


Figure 7: Root Mean Square Errors (RMSEs) of DeepMelt and its rivals over all testing examples (mean ± std over 10 runs). Note that the RMSE axis is in log scale.

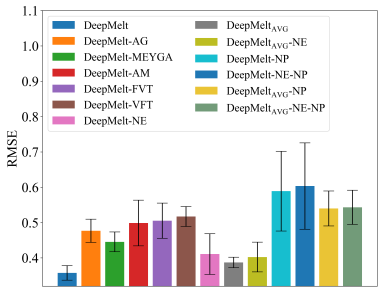


Figure 8: RMSEs of all ablation methods over all testing examples (mean ± std over 10 runs).

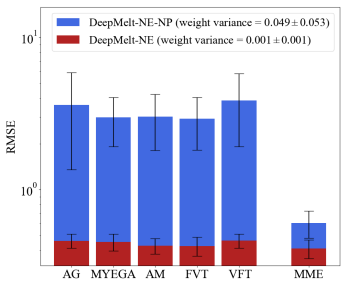


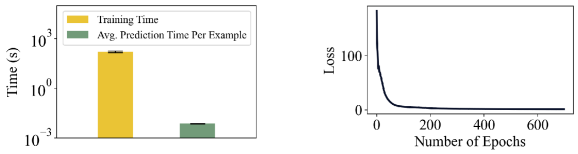
Figure 9: RMSEs of individual MMs, and the variance of the attentional weight vector, with and without the third term of the DeepMelt loss.

6.2.3 Ablation Studies. We now carry out ablation studies to validate the effectiveness of the key design features of DeepMelt (and M-MDLM in general), which are the MME, the use of the attention mechanism in the AA, the third term in the loss function (Eq. 6) that penalizes weak MME members, and the ECM. Thus, we devise the following changes to DeepMelt.

- **S-MDLM**: removing the MME by substituting DeepMelt with an S-MDLM for each of the 5 MMs, which has nearly identical architectures for the LFL, LPI and ECM to DeepMelt except for different input/output dimensions. We refer to these S-MDLMs as **DeepMelt-AG**, **DeepMelt-MYAGA**, **DeepMelt-AM**, **DeepMelt-FVT**, and **DeepMelt-VFT**.
- **AVG**: replacing the attention mechanism with simple averaging.
- **NP**: removing the third term in the loss function (Eq. 6).
- **NE**: removing the ECM.

The RMSEs of the ablation methods are shown in Fig. 8. As is indicated, DeepMelt beats all S-MDLMs, highlighting the power of ensembling multiple MMs. Also, the attention mechanism has enhanced DeepMelt's accuracy over DeepMelt$_{AVG}$, and the ECM has boosted the accuracy. However, the *single most important* contributing factor to DeepMelt's high accuracy is its third loss term, without which DeepMelt would perform poorer than the S-MDLMs.

6.2.4 An Insight Into the Third Loss Term. In view of its huge contribution to DeepMelt's accuracy, we now take a deeper look at how the third loss term works. As was mentioned in Section 4.2, the problem of sub-optimal parameters is especially damaging to M-MDLMs such as DeepMelt where there are shared parameters. While the AA cannot handle this problem, the third loss term can do so by forcing the *individual* MMs to perform as well as possible. We validate this claim in Fig 9, which shows the RMSEs of the individual MMs and the MME in DeepMelt-NE-NP and DeepMelt-NE (we use the methods with NE to avoid interference from the ECM). Also, it shows the average variance of the 5 values in the AA weight vector $\mathbf{w}$ in Eq. 5 across all testing examples over the 10 runs. The higher the variance, the more heavily a model relies on the AA to suppress weak predictions by individual MMs. As can be seen, compared to DeepMelt-NE-NP where the third loss term is missing, in DeepMelt-NE, the individual predictions are far better for all 5 MMs, which is only possible with better parameter identification (as there are no other factors influencing the accuracy of a fixed MM). By contrast, the weight variance of DeepMelt-NE-NP is much larger than DeepMelt-NE, which indicates that DeepMelt-NE-NP tries to make up for the weak individual predictions (due to the absence of the third loss term) by relying more heavily on the AA. However, this effect of this is limited, as is evidenced by its inferior MME predictions to DeepMelt-NE. This highlights the irreplaceable role the third loss term plays in ensuring the MME's accuracy.



(a) Running Time (b) Learning Curve

Figure 10: Efficiency of DeepMelt. (a) Running time of DeepMelt (mean ± std over 10 runs). The time axis is in log scale. (b) Learning curve of DeepMelt.

6.2.5 Efficiency. While we mainly focus on prediction accuracy, we now show that DeepMelt has reasonable efficiency. As is shown in Fig. 10 (a), DeepMelt can be trained in < 200s and can make a prediction in < 0.01s, which is sufficient for real-world deployment. Also, as is shown in Fig. 10 (b), DeepMelt converges very rapidly and can thus be trained with relatively few epochs.

7 CONCLUSIONS

In this paper, we proposed a generic, task-agnostic framework for mechanism-guided DL in manufacturing applications, which supports the embedding of one or more MMs in deep networks. To fulfill key requirements **R1**-**R3** listed in Section 1, we presented a deep PIM with awareness of high-level data examples, a data-driven ECM with error bounding, and an attentional MME with a simple-yet-effective loss term penalizing weak MMs. We instantiated our generic MDLMs in the 2 use cases of MOSP and MVP, with extensive experiments showcasing their effectiveness and efficiency.

ACKNOWLEDGMENTS

The work is supported by the Ministry of Science and Technology of China, National Key Research and Development Program (No. 2020YFB1710001). Shen Liang is funded by the Data Intelligence Institute of Paris (diiP), and IdEx Université Paris Cité (ANR-18-IDEX-0001).

REFERENCES

[1] Gerold Adam and Julian H Gibbs. 1965. On the temperature dependence of cooperative relaxation properties in glass-forming liquids. *The journal of chemical physics* 43, 1 (1965), 139–146.

[2] I Avramov and A Milchev. 1988. Effect of disorder on diffusion and viscosity in condensed systems. *Journal of non-crystalline solids* 104, 2-3 (1988), 253–260.

[3] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[4] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[5] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[6] Morrel H Cohen and GS Grest. 1979. Liquid-glass transition, a free-volume approach. *Physical Review B* 20, 3 (1979), 1077.

[7] Morrel H Cohen and Gary S Grest. 1984. The nature of the glass transition. *Journal of Non-Crystalline Solids* 61 (1984), 749–759.

[8] Arka Daw, M Maruf, and Anuj Karpatne. 2021. PID-GAN: A GAN Framework based on a Physics-informed Discriminator for Uncertainty Quantification with Physics. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 237–247.

[9] Angus Dempster, François Petitjean, and Geoffrey I Webb. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* 34, 5 (2020), 1454–1495.

[10] Harris Drucker, Christopher J Burges, Linda Kaufman, Alex Smola, and Vladimir Vapnik. 1996. Support vector regression machines. *Advances in neural information processing systems* 9 (1996).

[11] Mengge Du, Yuntian Chen, and Dongxiao Zhang. 2022. AutoKE: An automatic knowledge embedding framework for scientific machine learning. *IEEE Transactions on Artificial Intelligence* (2022).

[12] Jack Francis and Linkan Bian. 2019. Deep learning for distortion prediction in laser-based additive manufacturing using big data. *Manufacturing Letters* 20 (2019), 10–14.

[13] Simon Hagmeyer and Peter Zeiler. 2023. A Comparative Study on Methods for Fusing Data-Driven and Physics-Based Models for Hybrid Remaining Useful Life Prediction of Air Filters. *IEEE Access* (2023).

[14] Shizhong Han, Zibo Meng, Ahmed-Shehab Khan, and Yan Tong. 2016. Incremental boosting convolutional neural network for facial action unit recognition. *Advances in neural information processing systems* 29 (2016).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[16] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. 2021. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting* 37, 1 (2021), 388–427.

[17] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.

[18] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data mining and knowledge discovery* 33, 4 (2019), 917–963.

[19] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F Schmidt, Jonathan Weber, Geoffrey I Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. 2020. Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery* 34, 6 (2020), 1936–1962.

[20] Jungeun Kim, Kookjin Lee, Dongeun Lee, Sheo Yon Jhin, and Noseong Park. 2021. DPM: a novel training method for physics-informed neural networks in extrapolation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 8146–8154.

[21] Vladimir M Krasnopolsky and Ying Lin. 2012. A neural network nonlinear multimodel ensemble to improve precipitation forecasts over continental US. *Advances in Meteorology* 2012 (2012).

[22] Charles Le Losq, Andrew P Valentine, Bjorn O Mysen, and Daniel R Neuville. 2021. Structure and properties of alkali aluminosilicate glasses and melts: insights from deep learning. *Geochimica et Cosmochimica Acta* 314 (2021), 27–54.

[23] Naipeng Li, Yaguo Lei, Liang Guo, Tao Yan, and Jing Lin. 2017. Remaining useful life prediction based on a general expression of stochastic process models. *IEEE Transactions on Industrial Electronics* 64, 7 (2017), 5709–5718.

[24] Naipeng Li, Yaguo Lei, Tao Yan, Ningbo Li, and Tianyu Han. 2018. A Wiener-process-model-based method for remaining useful life prediction considering unit-to-unit variability. *IEEE Transactions on Industrial Electronics* 66, 3 (2018), 2092–2101.

[25] Xiang Li, Wei Zhang, Qian Ding, and Xu Li. 2019. Diagnosing rotating machines with weakly supervised data using deep transfer learning. *IEEE transactions on industrial informatics* 16, 3 (2019), 1688–1697.

[26] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. 2020. Explainable ai: A review of machine learning interpretability methods. *Entropy* 23, 1 (2020), 18.

[27] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3431–3440.

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3431–3440.

[29] John C Mauro, Yuanzheng Yue, Adam J Ellison, Prabhat K Gupta, and Douglas C Allan. 2009. Viscosity of glass-forming liquids. *Proceedings of the National Academy of Sciences* 106, 47 (2009), 19780–19784.

[30] Scott McQuade and Claire Monteleoni. 2012. Global climate model tracking using geospatial neighborhoods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 26. 335–341.

[31] Claire Monteleoni, Gavin A Schmidt, Shailesh Saroha, and Eva Asplund. 2011. Tracking climate models. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 4, 4 (2011), 372–392.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019). https://pytorch.org/

[33] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830. https://scikit-learn.org/stable/index.html

[34] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics* 378 (2019), 686–707.

[35] Emmanuel Ramasso. 2014. Investigating computational geometry for failure prognostics. *International Journal of prognostics and health management* 5, 1 (2014), 005.

[36] Frank Rosenblatt. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review* 65, 6 (1958), 386.

[37] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191.

[38] Abhinav Saxena and Kai Goebel. 2008. Turbofan engine degradation simulation data set. *NASA Ames Prognostics Data Repository* (2008), 1551–3203.

[39] Skipper Seabold and Josef Perktold. 2010. Statsmodels: Econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*, Vol. 57. Austin, TX, 10–25080. https://www.statsmodels.org/stable/index.html

[40] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. 2018. DiSAN: Directional Self-Attention Network for RNN/CNN-Free Language Understanding. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. AAAI Press, 5446–5455.

[41] Chang Wei Tan, Christoph Bergmeir, François Petitjean, and Geoffrey I Webb. 2021. Time series extrinsic regression. *Data Mining and Knowledge Discovery* 35, 3 (2021), 1032–1060.

[42] Hans Vogel. [n.d.]. Das Temperaturabhangigkeitsgesetz der Viskositat von Flussigkeiten. *Physikalische Zeitschrift* 22 ([n. d.]), 645–646.

[43] Nils Wandel, Michael Weinmann, Michael Neidlin, and Reinhard Klein. 2022. Spline-pinn: Approaching pdes without data using fast, physics-informed hermite-spline cnns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 8529–8538.

[44] Biao Wang, Yaguo Lei, Naipeng Li, and Ningbo Li. 2018. A hybrid prognostics approach for estimating remaining useful life of rolling element bearings. *IEEE Transactions on Reliability* 69, 1 (2018), 401–412.

[45] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12.

[46] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[47] Zhongbao Wei, Guangzhong Dong, Xinan Zhang, Josep Pou, Zhongyi Quan, and Hongwen He. 2020. Noise-immune model identification and state-of-charge estimation for lithium-ion battery using bilinear parameterization. *IEEE Transactions on Industrial Electronics* 68, 1 (2020), 312–323.

[48] Hongzuo Xu, Yijie Wang, Songlei Jian, Zhenyu Huang, Yongjun Wang, Ning Liu, and Fei Li. 2021. Beyond Outlier Detection: Outlier Interpretation by Attention-Guided Triplet Deviation Network. In *WWW*. ACM / IW3C2, 1328–1339.

[49] Jie Yang, Tianyou Chai, Chaomin Luo, and Wen Yu. 2018. Intelligent demand forecasting of smelting process using data-driven and mechanism model. *IEEE Transactions on Industrial Electronics* 66, 12 (2018), 9745–9755.

[50] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 11106–11115.

A SUPPLEMENTAL MATERIAL

We now provide supplemental material concerning the reproducibility of the results of our experiments. Note that this does not involve the data preprocessing protocols for the HZ dataset in Section 5.2, as it is a proprietary dataset that entails confidential business information.

A.1 Experimental Details of Mechanical Operation State Prediction (MOSP)

We now provide additional details about the experiments on our DeepWiener S-MDLM for the MOSP task in Section 5.2, including details on data preprocessing, as well as rival method implementation and hyperparameter settings.

A.1.1 Data Preprocessing. On data preprocessing, recall that in our experiments, we used 2 datasets: NASA [38] and XJTU-SY [44]. We now separately discuss how we preprocessed these datasets.

1) *NASA.* Recall that NASA entails simulated sensor readings of turbofan engines which reflect their degradation process. Therefore, the main objective of data preprocessing for NASA is to convert the readings of multiple sensors into a single-variate OSV time series. To complete this objective, we first remove the sensors where all the readings are the same. Then, for each of the remaining sensors, we use min-max normalization to normalize all its readings to the range $[0, 1]$. Then, we draw on the method proposed in [24, 35] to obtain the OSVs. Specifically, suppose there are D sensors for a given engine. At timestamp t_k, we denote its sensor reading vector as $\mathbf{r}_k = r_k^1, r_k^2, \ldots, r_k^D$, and use a linear model

$$o_k = \mathbf{l} \cdot \mathbf{r} + c \tag{8}$$

to obtain the OSV o_k at t_k, where $\mathbf{l}$ is the linear coefficient vector and c is the bias. To fit this linear model, we note that apart from the data we use in our experiments, NASA comes with a separate dataset (which is NOT used in our experiments in Section 5.2) dedicated to fitting the linear model. We call this dataset the *fitting set*. For each engine in the *fitting set*, apart from all its sensor readings, the length of its total lifespan (from first being operational to total failure) T is also known, and its groundtruth OSV o'_k at a given timestamp t_k is calculated as [24, 35]

$$o'_k = \begin{cases} 0 & 0 \le t_k \le 0.05T \\ \exp\left(\frac{\log(0.05)}{0.95T}(T - t_k)\right) & 0.05T < t_k < 0.95T \\ 1 & 0.95T \le t_k \le T \end{cases} \tag{9}$$

With both the raw sensor readings and groundtruth OSVs known on the fitting set, we can use them to fit the linear model in Eq. 8 to obtain the OSVs for the data used in our experiments in Section 5.2. For the OSV time series of each engine, we use its first half for training and validation, and its second half for testing.

2) *XJTU-SY.* For XJTU-SY, recall that the raw data we used is the raw vibration signals of the bearings. According to our domain expert, for each raw signal, a large number of data points at its beginning reflect no significant degradation, and should be discarded. Therefore, there are 2 main objectives of data preprocessing for XJTU-SY: *First*, we need to decide at which point the degradation process begins, so that we can discard all data points prior to it. *Second*, we need to convert the remaining data points in the raw vibration signal into the OSV time series. We now elaborate on how we complete these 2 objectives one by one.

For the *first* objective of finding the start of significant degradation, we draw on the 3σ rule as guided by our domain expert. Specifically, for all data points in each raw vibration signal, we calculate the mean value μ and standard deviation σ of their absolute amplitudes. We find the first point in the signal whose absolute amplitude exceeds $\mu + 3\sigma$. This is statistically the first outlier in the signal, and likely indicates the start of significant degradation as the bearing is now exhibiting excessively violent vibration. We discard all data before this point, only keeping the rest.

For the *second* objective of transforming each raw vibration signals to an OSV time series, we note that in the original XJTU-SY dataset [44], a 1,280ms snippet is already sampled every 1min. We follow the advice of our domain expert and use the root mean square of the snippet (which has 32,768 raw data points with an original sampling rate of 25.6 kHz) as its corresponding OSV. Specifically, let the data points in the k-th snippet be $r_k^1, r_k^2, \ldots, r_k^{32768}$, the corresponding OSV is calculated as

$$o_k = \sqrt{\frac{1}{32768}\sum_{i=1}^{32768}(r_k^i)^2} \tag{10}$$

The OSV time series is thus formed by the OSVs of all snippets. Similar to what was done for NASA, for the OSV time series of each signal in XJTU-SY, we use its first half for training and validation, and its second half for testing.

A.1.2 Rival Method Implementation and Hyperparameter Settings. On the implementation and hyperparameter settings, we note that for our DeepWiener, these have been presented in Sections 5.1 and 5.2. Here, we focus on the rival methods used in the experiments in Section 5.2 (more specifically, in Fig. 3).

To be concrete, we implemented Wiener with Python, and implemented MLP and RNN with PyTorch [32]. For SVR and RF, we adopted their implementations in the Scikit-learn [33] package. For ARIMA, we adopted its implementation in the Statmodels [39] package. For FCN, ResNet and InceptionTime, we adopted their implementations in [41]. For XGBoost, DeepAR and Informer, we used their implementations by their original authors [5, 37, 50]. As with hyperparameter settings, except for Wiener, RNN and MLP which we implemented from scratch, we mostly inherited the default hyperparameter settings of the other rival methods in either the implementations by the original authors or the third-party packages we used in our experiments. Table 3 shows the hyperparameter settings in detail[1]. Specifically, for Wiener, the parameter b must be searched from a pre-defined range. We set this range to be both wide enough to cover a large number of candidate values, and also be suited to the datasets we used.

As with the hyperparameters for SVR, RF, XGBoost, ARIMA, ROCKET, FCN, InceptionTime and ResNet, we consider multiple candidate settings *(including the default settings)* from which we choose the best one by 5-fold CV. For MLP, we adopt a commonly-used MLP architecture than also performs reasonably well on our data. For RNN, the network architecture is identical to that of the DeepWiener HFL except for the fully-connected layer to make the final prediction (which the HFL does not need). Also, the optimizer, learning rate and the number of epochs are all the same as those

[1]Note that in Table 3, we only list the settings that differ from the default hyperparameters (except for Wiener, MLP and RNN where we list the full settings).

Table 3: Hyperparameter Settings for the Rival Methods of DeepWiener for Mechanical Operation State Prediction (MOSP)

Method	Model Hyperparameters
Wiener	Search range of b: {0.01,0.02,0.03,...,1}
SVR	Kernel function: {RBF, Sigmoid} Kernel coefficient: {0.001,0.01,0.1,1, "scale"} Regularization parameter: {0.01,1,10,100}
RF	Number of trees: {100, 500, 1000}
XGBoost	Number of trees: {100, 500, 1000, 2000} Regularization parameter: {0, 0.01,1,10,100}
MLP	Number of hidden layers: 3 Dimensions of each hidden layer (in order of forward propagation): 32, 16, 32 Batch normalization after each hidden layer Activation after each hidden layer: PReLU Loss function: L2 Optimizer: Adam Learning rate: reduce on plateau with factor = 0.5, patience = 50, minimum learning rate = 0.0001 Number of training epochs: 100
ARIMA	Autoregression parameter: {0, 1,2,4} Differentiation parameter: {0,1,2} Moving average parameter: {0,1,2}
ROCKET	Number of kernels: {100, 1000, 5000, 10000}
RNN	Number of RNN hidden layers: 2 Dimension of each RNN hidden layer: 16 Number of fully-connected hidden layers: 1 Dimension of fully-connected hidden layer: 1 Activation after fully-connected hidden layer: PReLU Loss function: L2 Optimizer: AdamW Learning rate: reduce on plateau with factor = 0.5, patience = 10, minimum learning rate = 0.0001 Number of training epochs: 100 (with an early-stop policy with a patience of 10 and delta $\in \{0.01, 0.1\}$)
DeepAR	Number of LSTM units: {32,64,128,256}
FCN	Number of filters: {8,16,32} Kernel size: {1, 2, 4}
InceptionTime	Number of filters: {8,16,32} Kernel size: {1, 2, 4}
ResNet	Number of filters: {8,16,32} Kernel size: {1, 2, 4}
Informer	–

Table 4: Hyperparameter Settings for the Rival Methods of DeepMelt for Melt Viscosity Prediction (MVP)

Method	Model Hyperparameters
iMelt	–
SVR	Kernel function: {RBF, Sigmoid} Kernel coefficient: {0.001,0.01,0.1,1, "scale"} Regularization parameter: {0.01,1,10,100}
RF	Number of trees: {100, 500, 1000}
XGBoost	Number of trees: {100, 500, 1000, 2000} Regularization parameter: {0, 0.01,1,10,100}
MLP	Number of hidden layers: 4 Dimensions of each hidden layer: 400 Activation after each hidden layer: ReLU Loss function: L2 Optimizer: Adam Learning rate: 0.0006 Weight decay: 0.01 Number of training epochs: 10,000

of DeepWiener in order to maximize the comparability between DeepWiener and its rival. For DeepAR, we use its settings in the extended version of the Informer paper [50] (where it is used as a baseline method), available at https://arxiv.org/abs/2012.07436. These settings entail multiple candidates from which we choose the best by 5-fold CV. The reason why we use these settings, rather than the DeepAR settings by its original authors [37], is that the latter did not provide concrete advice on the search range for the hyperparameters that we can follow. For Informer, we remain faithful to its hyperparameter settings by the original authors.

A.2 Experimental Details of Melt Viscosity Prediction (MVP)

We now turn to additional details about the experiments on our DeepMelt M-MDLM for the MVP task in Section 6.2, which concerns rival method implementation and hyperparameter settings. Note that unlike in the experiments for the MOSP task where explicit data preprocessing was required, in the MVP experiments, we use the same dataset as that in [22], where the authors have already preprocessed the data and split it into training, validation and testing sets.

On the implementation and hyperparameter settings, similar to the case with DeepWiener, for our DeepMelt, these settings have been presented in Sections 6.1 and 6.2. Here, we focus on the rival methods used in the experiments in Section 6.2 (more specifically, in Fig. 7).

To be concrete, for iMelt and XGBoost, we used the implementations by the original authors of these methods [5, 22]. For the data-driven methods, we implemented MLP with PyTorch [32] while adopting the implementations in Scikit-learn [33] for SVR and RF. Similar to what we did in the MOSP experiments, except for MLP which we implemented from scratch, we mostly inherited the default hyperparameter settings of the other rival methods in either the implementations by their original authors or the third-party packages we used in our experiments. Table 4 shows the hyperparameter settings in detail[1]. Specifically, for iMelt, we remain faithful to the settings by the original authors, as they have already fine-tuned them to the dataset used in both their paper [22] and ours. For SVR, RF and XGBoost, we consider multiple candidate settings *(including the default settings)* from which we choose the one with the lowest RMSE on the validation set. Again, we note that the validation set was already present in the original dataset [22]. For MLP, we use the same architecture (i.e. number of hidden layers, dimensions of each hidden layer, activation function) as the MLP used as the LFL of our DeepMelt model to maximize comparability between DeepMelt and its rival. The optimizer, learning rate and weight decay are also set to be the same as those of both DeepMelt and iMelt, again to maximize comparability. As with the number of training epochs, we found that MLP could not obtain near-optimal accuracy with a small number of epochs, thus we set it to a large number of 10,000.

[1]Note that in Table 4, we only list the settings that differ from the default hyperparameters (except for MLP where we list the full settings).