Neural Architecture and Hyperparameter Selection through Meta-Learning on Time Series

Erfan Moeini[1, 2], **Christopher Vox**[2], **Marie Anastacio**[1, 3], **Wadie Skaf**[1], **Mitra Barachi**[3], **Holger H. Hoos**[1, 3, 4]

[1] Chair for Artificial Intelligence Methodology (AIM), RWTH Aachen University, Aachen, Germany
[2] Volkswagen AG, Wolfsburg, Germany
[3] Leiden Institute of Advanced Computer Science (LIACS), Leiden University, Leiden, The Netherlands
[4] University of British Columbia, Vancouver, Canada

Abstract

Active research in time series classification and forecasting has led to the development of a wide range of machine learning models. For practitioners, the selection of a suitable model among these, along with their hyperparameters, remains a challenging task. While automated machine learning offers approaches for automatic selection of models for a given task, the practical efficacy of these methods is often limited, due to the computational complexity of searching over a large design space and the high dimensionality of time series datasets that poses additional challenges on generalisation quality. To fill this gap, we propose a meta-learning framework that transfers past knowledge from previous searches to recommend an architecture and its hyperparameters; specifically, this framework utilises a joint representation of deep neural architectures and time series datasets, and predicts the performance of neural architectures along with their hyperparameters on time series datasets. Our computational experiments reveal that the configurations proposed by our meta-learned surrogate achieve a performance gain of up to 34% on 4 out of the 8 forecasting datasets we considered and up to 60% on 36 out of 73 of our classification datasets, whilst reducing the computational cost to 10% of that required by the hyperparameter optimisation method HEBO to tune the architectures, showcasing the effectiveness of meta-learning in the time series domain.

1 Introduction

Time series analysis has been an active research area, with applications spanning a wide range of domains, such as weather forecasting (Mung and Phyu 2023) and healthcare (Skaf, Tosayeva, and Várkonyi 2022). To capture the temporal dependencies inherent to time series data, a variety of models have been proposed, with deep learning models using different architectural backbones (*e.g.*, transformers, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), or Multi-Layer Perceptrons (MLPs)) showing superior results (Ismail Fawaz et al. 2019; Middlehurst, Schäfer, and Bagnall 2024).

Even though past models adopted different architecture backbones and claimed superiority over one another, it is not realistic to assume that there exists a one-size-fits-all solution, *i.e.*, each architecture might perform best on certain types of datasets and has its own advantages and disadvantages. Manual selection often requires extensive domain-specific and expert knowledge, which renders the usability of these architectures for most users limited within constrained time and computational resources. Consequently, automating or guiding the selection of appropriate architectures and hyperparameter (HP) values based on the characteristics of given time series datasets can be seen as a promising research direction.

Extended Version

Next to recent advances in machine learning, there has also been considerable attention on automating many tasks involved in the selection and configuration of a machine learning model that once required human intervention. This has led to the development of the subfield known as *Automated Machine Learning (AutoML)* (see, *e.g.*, Hutter, Kotthoff, and Vanschoren 2019; Baratchi et al. 2024). In particular, Hyperparameter optimisation (HPO) and Neural Architecture Search (NAS)—two key areas within AutoML—aim at identifying high performing models from a search space of possible design choices (*i.e.*, HPs, architectures), using effective search strategies. However, the practical efficacy of these methods is often restricted due to the computational cost of exploring large, interdependent HP search spaces and the high dimensionality of time series data, which impacts model generalisation. Despite the potential performance improvements that AutoML can achieve, time and computational budgets are restrictive factors to their adoption when tackling large-scale problems. This can mainly be attributed to the fact that these methods are, in essence, negligent of the previous search history, *i.e.*, the search for promising architectures and HP candidates is initiated independently for any given task.

To allow for transferability of knowledge across tasks, we propose a meta-learning-based approach by building a surrogate model for predicting model performance. The main challenge to achieve this goal lies in modelling the performance of architecture and HP pairs by simultaneously capturing the similarities between datasets and architectures. To address this challenge, our contributions are as follows:

1. We extract and compile knowledge regarding the performance of a wide range of deep learning architectures proposed for time series classification and forecasting tasks. The performance of each architecture is recorded with different HP configurations evaluated during hyperparameter optimisation (HPO) for a fixed number of trials

The performance of each architecture is recorded with different HP configurations evaluated during hyperparameter optimisation (HPO) for a fixed number of trials with HEBO (Cowen-Rivers et al. 2022), winner of the 2020 Neurip black-box optimisation challenge.

2. Using meaningful representations for architectures and datasets, our surrogate model preserves their similarity structures in the embedding space, which enables to predict task-dependent performance for new datasets, and eliminates the need to train a new surrogate from scratch for new tasks.
3. We empirically evaluate the prediction of our surrogate on 45 univariate and 29 multivariate time series classification and 8 forecasting datasets, and demonstrate its superiority over our HPO baseline HEBO by using a k-shot configuration proposal scheme.

The remainder of this paper is structured as follows: Section 2 introduces the key definitions and notations. Section 3 discusses related work for time series analysis and the application of AutoML in this domain. Section 4 describes our meta-learning framework, the encoding schemes for architectures, HPs and datasets, and the meta-prediction strategy. Section 5 describes our experiments to evaluate and compare our approach to HPO and Section 6 presents the results of the evaluation. Section 7 presents some overall conclusions and potential directions for future work.

2 Problem Definition

Time series data is generally defined as a sequence of data points $\mathbf{x_i} \in \mathbb{R}^d$ that capture temporal changes for d variables, where $d \in \mathbb{N}$. Let $\mathbf{X} = [\mathbf{x_1}, \cdots, \mathbf{x_T}] \in \mathbb{R}^{T \times d}$ denote a multi-variate time series of temporal length $T \in \mathbb{N}$ with d variable dimensions. In the following, we present the main problem definitions with regard to the time series $\mathbf{X}$.

Time Series Classification. When classifying, the aim is to find a classifier $c : \mathbb{R}^{T \times d} \to \mathcal{L}_b$ that maps a given time series $\mathbf{X} \in \mathbb{R}^{T \times d}$ to a label $l \in \mathcal{L}_b$, where $\mathcal{L}_b = \{l_1, \cdots, l_k\}$ is the set of k possible labels for the given task: $l = c(\mathbf{X})$

Time Series Forecasting. When forecasting, the aim is to learn a function $f : \mathbb{R}^{L \times d} \to \mathbb{R}^{H \times d}$ that maps past observations within a look-back window of size L to future values within a horizon of length H at time-step t, where L, H and t are positive integers: $\hat{\mathbf{Y}}_{t:t+H} = f(\mathbf{X}_{t-L:t})$.

Meta-Learning for CASH. We apply meta-learning (Brazdil et al. 2022) to the combined algorithm selection and hyperparameter optimisation (CASH) problem. We aim to find an *arch-hyper* $A_{\boldsymbol{\lambda}}$ – defined by Wu et al. (2023b) as a neural network architecture A from a set of neural network architectures $\mathcal{A}$ with its training and model-specific HPs $\boldsymbol{\lambda}$ from their possible values Λ – that minimises a given loss function $\mathcal{L}$ based on the task (time series classification or forecasting):

$$A^*_{\boldsymbol{\lambda}^*} \in \underset{A \in \mathcal{A}, \boldsymbol{\lambda} \in \Lambda}{\arg\min}\, \mathcal{L}(A_{\boldsymbol{\lambda}}, D_{tr}, D_{val}). \tag{1}$$

We learn a meta-predictor model as surrogate for $\mathcal{L}$ to address the CASH problem for a new time series dataset D, split into training D_{tr} and validation D_{val} sets.

3 Related Work

Time series analysis and AutoML are key areas in machine learning, yet their intersection remains underexplored. This section reviews key contributions to deep learning for time series in addition to prior efforts to automate neural architecture search and hyperparameter optimisation.

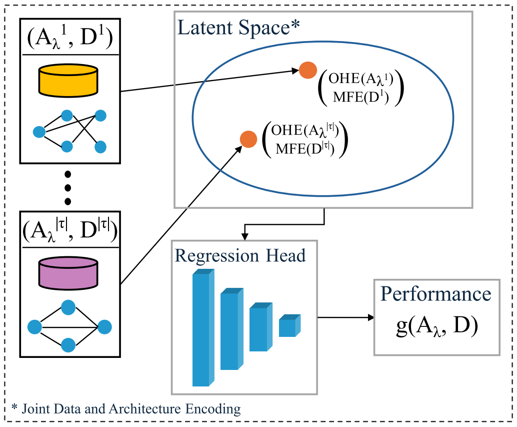
Deep Learning for Time Series. Many studies have focused on designing deep learning architectures to extract intricate temporal relations from time series data. For instance, *LightTS* (Zhang et al. 2022) uses an MLP-based structure on top of two down-sampling strategies preserving the majority of information. *ModernTCN* (Donghao and Xue 2024) employs a CNN-based architecture to capture temporal and cross-variable dependencies through convolutional layers at different granularities. To address the limitation of RNNs in handling sequential data with long look-back windows and forecast horizons *SegRNN* model, (Lin et al. 2023) decomposes the time series data into multiple segments, thereby replacing point-wise iterations with segment-wise iterations. More recently, transformers (Zhou et al. 2021; Wu et al. 2021) have gained popularity due to their ability to capture long-term temporal dependencies via self-attention. Despite all the efforts to address the deficiencies of time series models, it still remains unclear what models and HP configurations would be best suited for a given dataset.

Automatic Architecture and Hyperparameter Selection. To decide on the HP values to be used, many general-purpose HPO methods rely on some flavour of Bayesian optimisation (BO). *SMAC* (Hutter, Hoos, and Leyton-Brown 2011) uses a random forest (or Gaussian process) surrogate with an acquisition function to efficiently explore configuration spaces. *BOHB* (Falkner, Klein, and Hutter 2018) combines Bayesian optimisation with the early-stopping mechanism and adaptive resource allocation of HyperBand.*HEBO* (Cowen-Rivers et al. 2022) models complex non-uniform noise in the objective function through input warping and output transformations, and uses a multi-objective acquisition function with evolutionary optimisers to overcome potential conflicts between different acquisition functions.

Thornton et al. (2013) defined the CASH problem, unifying algorithm selection and HPO as a single hierarchical optimisation problem, embedding the algorithm choice as an additional HP. Addressing the CASH problem with HPO approaches led to many broad-spectrum AutoML frameworks based on classic machine learning algorithms (*e.g.*, AutoWeka (Thornton et al. 2013), AutoGluon (Erickson et al. 2020)). With the large variety of possible neural network architectures, Neural Architecture Search (NAS) has become a key direction in automated machine learning, aiming to optimise architectural HPs, such as the number of layers and their operations. Consequently, many NAS approaches have been proposed (see *e.g.*, BANANAS (White, Neiswanger, and Savani 2021), DARTS (Liu, Simonyan, and Yang 2019), AutoPyTorch (Zimmer, Lindauer, and Hutter 2021)). Recently, research has been conducted to extend AutoML to be applicable to time series data. *AutoCTS* (Wu et al. 2023b) introduces a joint NAS and HPO framework using graph isomorphism networks for time series forecasting. The prominent AutoML systems *AutoGluon* (Erickson et al. 2020) and *AutoPyTorch* (Zimmer, Lindauer, and Hutter 2021) introduced time series specific mechanisms in recent years – AutoGluon-TS (Shchur et al. 2023) and AutoPyTorch-TS (Deng et al. 2022), respectively. Despite the proven efficacy of these methods in the time series domain, they all suffer from one important limitation: the selection of appropriate architectures and HPs for a new dataset requires re-evaluation of the entire search space, disregarding the knowledge gained through observations on past datasets. To address this, Mu et al. (2023) proposed a meta-learning approach for time series classification, using decision-trees to recommend algorithms based on dataset meta-features. The method selects from 23 algorithms across 8 categories, but requires retraining to incorporate new algorithms. We address this limitation by proposing a general framework through encoding neural architectures in addition to datasets, thereby eliminating the need to retrain the surrogate for new architectures. We extend their work by also taking into account the forecasting task.

4 Proposed Framework

Our proposed framework relies on a meta-predictor that serves as a surrogate model of the performance of arch-hypers. This meta-predictor is later used in a meta-search step to recommend a few neural architectures along with their HP settings for a previously unseen dataset, based on performance predictions. This section explains the key components of this framework: (i) the encoding methods used to learn the joint representation of time series datasets and neural network architecture, (ii) the meta-predictor model and its structure, and (iii) the workflow for CASH using the meta-predictor.



(A_λ^1, D^1)
$(A_\lambda^{|\tau|}, D^{|\tau|})$
Latent Space*
$\begin{pmatrix}\text{OHE}(A_\lambda^1)\\ \text{MFE}(D^1)\end{pmatrix}$
$\begin{pmatrix}\text{OHE}(A_\lambda^{|\tau|})\\ \text{MFE}(D^{|\tau|})\end{pmatrix}$
Regression Head
Performance
$g(A_\lambda, D)$
* Joint Data and Architecture Encoding

Figure 1: Overview of our meta-predictor. The regression head is a MLP with ReLU activation.

4.1 Data and Architecture Embedding

Our approach relies on the ability of our meta-learning model to learn from a database of performance evaluation results of diverse architectures on different time series datasets. For this purpose, neural architectures encodings and their HP values are jointly represented with the encoding of the dataset to ensure that the performance prediction is conditioned on both dataset and architecture representations.

Time Series Dataset Encoding. Following similar works in the literature (Mu et al. 2023; Talkhi et al. 2024; Uddin and Lu 2024), we encode time series datasets using a set of meta-features, which describe their statistical and temporal properties. This choice is further motivated in Section 5.

Architecture Encoding. During our preliminary analysis (see Section 5), we identified that the one-hot structural encodings of neural architectures and HPs (Akhauri and Abdelfattah 2024; White et al. 2020; Wu et al. 2023b) performs well for our task. To one-hot encode a neural architecture A, we extract the associated computational directed acyclic graph (DAG) during a forward pass simulation. In this DAG representation, nodes represent operations and edges information flow between them. The adjacency matrix $M \in \mathbb{R}^{N\times N}$ of the corresponding DAG containing $N \in \mathbb{N}$ nodes is flattened and concatenated with the operations vector $O \in \mathbb{R}^N$ to derive a final representation for the architecture. The operations vector holds for each node the associated numerical value obtained from categorical encoding of the set of all available operations in the candidate architectures. We encode the information about HP values with the min-max normalised HP vector $\lambda \in \mathbb{R}^K$. The categorical HPs are initially encoded using a categorical encoding scheme; these include training HPs, such as learning rate, and model-specific HPs, such as the embedding dimension in transformers. To ensure similar dimensionality for the embedding vectors of different neural architectures, the corresponding adjacency matrix and operation vector are padded with zeros. We fixed the dimension of HP vector and the position of each HP across time series models, assigning reserved values to indicate irrelevance for the current architecture. In summary, using $\|$ to denote concatenation, the one-hot representation of a given architecture is obtained as:

$$\text{OHE}(A_{\boldsymbol{\lambda}}) = \text{FLATTEN}(M) \,\|\, O \,\|\, \lambda. \tag{2}$$

4.2 Meta-Predictor Model

The meta-predictor model is the key element in our framework. As illustrated in Figure 1, it is defined as a function $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ that takes the embedding vector representations of a time series dataset D and an arch-hyper A_λ as input , and learns to map their joint representation to a final performance metric v that is determined by the downstream time series task. Our approach for arch-hyper representation is inspired by *AutoCTS* (Wu et al. 2023b). We further extend their work by introducing model-specific HPs in addition to training HPs. Inspired by similar works in this domain (Lee, Hyung, and Hwang 2021; Shala et al. 2023), the concatenated data and arch-hyper encoding vectors are fed into a batch of four fully connected linear layers with ReLU activation to obtain a final performance value, as illustrated in Figure 1. HPs of the meta-predictor were tuned with 200 iterations of a simple BO approach. Let $\mathcal{T}$ denote the meta-database containing triplets of the form $\tau = (D, A_\lambda, v)$. Throughout the training phase, the learnable weights of the surrogate of the meta-predictor are updated by minimising a loss function $\hat{\mathcal{L}}$ over the predicted and true performance values v for each record τ sampled from the source database:

$$g^* \in \arg\min_{g} \sum_{(D,A_{\boldsymbol{\lambda}},v)\in\mathcal{T}} \hat{\mathcal{L}}(v, g(A_{\boldsymbol{\lambda}}, D)) \quad (3)$$

To decide which loss function to use, we consider that the objective of the meta-predictor is twofold. First, it should accurately estimate the performance of a given arch-hyper on time series datasets. Second, we need to preserve their relative ranking based on their performance. $\hat{\mathcal{L}}$ should penalise both the difference between the estimated and true performance value, and the incorrect ordering of arch-hypers based on those estimated performances. Therefore, we combined the Mean Squared Error (MSE) and Margin Ranking Loss (MRL) as the objective function for training the meta-predictor (Hwang et al. 2024). This objective function, referred to as Combined Ranking Loss (CRL), is defined as:

$$CRL(\mathbf{y}, \hat{\mathbf{y}}) = MSE(\mathbf{y}, \hat{\mathbf{y}}) + MRL(\mathbf{y}, \hat{\mathbf{y}}) \quad (4)$$

$$\text{where } MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{|\mathcal{T}|} \cdot \sum_{i=1}^{|\mathcal{T}|} (y_i - \hat{y}_i)^2,$$

$$\text{and } MRL(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{y_i > y_j} \max(0, \delta - (\hat{y}_i - \hat{y}_j)),$$

where $\mathbf{y} = [y_1, \ldots, y_{|\mathcal{T}|}]$ and $\hat{\mathbf{y}} = [\hat{y}_1, \ldots, \hat{y}_{|\mathcal{T}|}]$ with y_i and $\hat{y}_i$ being the ground-truth and predicted performance for the arch-hyper i, respectively, and δ denotes the margin parameter in MRL that indicates how strict the loss function should be when encountering a wrong ordering. Note that each y_i corresponds to an earlier introduced v.

Section 5 provides a detailed comparison of the impact of different objective functions on the performance of meta-predictors.

4.3 Meta-Search Workflow

The aim of the meta-predictor is to replace the time-consuming steps of selecting an architecture and optimising its HPs. Figure 2 illustrates the proposed framework. The steps of the search strategy are formalised in Algorithm 1. Given a new time series dataset, the meta-search algorithm starts by searching for the most similar dataset according to the L^1-norm distance between meta-feature vectors of two datasets (line 1). Then, it searches for the top-N architectures based on past performance evaluations on this dataset (line 2). For each of these N architectures, the meta-predictor predicts the expected performance of a grid of HP values and returns the top-K performing ones (line 6).

Algorithm 1 Meta-Search Algorithm

Input: Trained meta-predictor g; Set of historical datasets $\mathcal{D}$; Input dataset D_n; data similarity function s; Historical observations $\mathcal{T}$; HP search space Λ; Number of top architectures N; Number of HP configurations K per architecture; Evaluation function F

Output: List of top architecture and HP configurations $\mathcal{O}$.

```
    $D_s \leftarrow \arg\min_{D\in\mathcal{D}} s(D_n, D)$;   ▷ Find most similar dataset to D
 2: $\mathcal{A}^\star \leftarrow \arg\max_{\tau\in\mathcal{T}} F(\tau, D_n, N)$;   ▷ Find top-$N$ architectures on the similar dataset
    for all $A \in \mathcal{A}^\star$ do
 4:     $A_{\boldsymbol{\lambda}} = A \parallel \boldsymbol{\lambda}$;   ▷ Create candidate arch-hypers with each architecture and HP configuration in the search space
        $\mathbf{v} \leftarrow g(A_{\boldsymbol{\Lambda}}, D_n)$;   ▷ Predict performance values
 6:     $A_{\boldsymbol{\Lambda}^*} \leftarrow \arg\max_K(\mathbf{v})$;   ▷ Find top-$K$ HP configurations
        $\mathcal{O} \leftarrow \mathcal{O} \cup A_{\boldsymbol{\Lambda}^*}$;   ▷ Update the set of output
 8: end for

10: return $\mathcal{O}$
```

5 Experimental setup

The main aim of our experiments is to evaluate the effectiveness of our approach against the widely used approach of tuning the HPs of architectures on the task at hand. To do so, we compare the performance of arch-hypers suggested by our framework to those obtained by independently tuning the HPs of each architecture. In this section, we present the choice of architectures and datasets, the HPO baseline, the data collection approach and the design choices for the meta-predictor.

Architectures. We considered 20 architectures with different backbones that are frequently used in time series literature. Among CNN-based architectures, we chose ModernTCN (Donghao and Xue 2024), TS2Vec (Yue et al. 2022) and TimesNet (Inception, ResNeXt) (Wu et al. 2023a). Among RNN-based architectures, we use FiLM (Zhou et al. 2022), LMU (Voelker, Kajic, and Eliasmith 2019), Mamba (Gu and Dao 2024), SegRNN (Lin et al. 2023), and S4 (Hasani et al. 2023). Among MLP-based architectures, we included DLinear (Zeng et al. 2023), LightTS (Zhang et al. 2022), and TSMixer (Chen et al. 2023). Finally, among transformer-based architectures, we considered Autoformer (Wu et al. 2021), Crossformer (Zhang and Yan 2023), ETSformer (Woo et al. 2022), Informer (Zhou et al. 2021), PatchTST (Nie et al. 2023), Reformer (Kitaev, Kaiser, and Levskaya 2020), Transformer (Vaswani et al. 2017) and iTransformer (Liu et al. 2024).

Each of these architectures has two types of hyperparameters: (i) training HPs and (ii) model-specific HPs. While the training HPs are similar across all architectures, the model-specific HPs depend on the structure and the underlying functional units. Moreover, some HPs can only be considered depending on the downstream task (*e.g.* decoder component of transformers for regression tasks). A complete list of HPs can be found in the supplementary materials.

Datasets. For time series classification, we used the *UEA* multivariate classification datasets (Bagnall et al. 2018), excluding 2 out of 30 datasets: (i) *InsectWingBeats*, due to its extensive training cost, and (ii) *EigenWorms*, due to out-of-memory conditions with both HEBO and the meta-predictor. From the *UCR* univariate classification datasets (Dau et al. 2019), we sampled uniformly at random one third of each subject category specified by the authors (*e.g.*, *Motion*, *Au-*

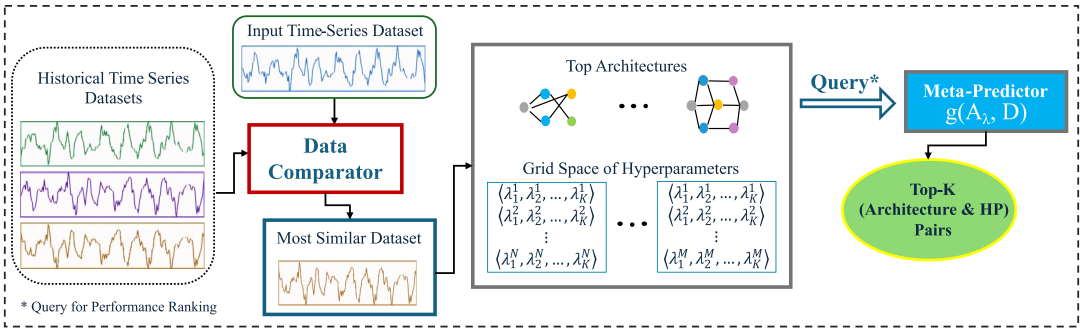

Input Time-Series Dataset
Historical Time Series Datasets
Data Comparator
Most Similar Dataset
Top Architectures
Grid Space of Hyperparameters
Query*
Meta-Predictor
g(A_λ, D)
Top-K (Architecture & HP) Pairs
* Query for Performance Ranking

Figure 2: Overview of the meta-search strategy. Once the most similar historical dataset is identified, the meta-predictor predicts the performance values of the top-performing architectures and their grid space of hyperparameters and ranks them accordingly.

dio, *Sensor*), resulting in a total of 45 datasets. For time series forecasting, we selected eight datasets frequently used in time series literature. An overview of the selected datasets can be found in the supplementary material.

Baseline and Data Collection. For our HPO baseline, we used the *RayTune* implementation of HEBO, winner of the NeurIPS 2020 black-box optimisation competition (Turner et al. 2021), based on preliminary experiments comparing the performance of top HP configurations suggested by several HPO algorithms, including BANANAS (White, Neiswanger, and Savani 2021), HyperOpt (Bergstra et al. 2015) and random search (Bergstra and Bengio 2012), with 128 iterations per algorithm. To train the meta-predictor, we need the performance of a subset of arch-hypers over time series datasets. A typical approach would have been to run a grid search over the HP space of each architecture and observe their performance on each dataset. However, due to the high dimensionality of the search spaces, this approach is inefficient. Instead, we kept track of the performance evaluations carried out during optimisation with HEBO over 128 trials per model-dataset pair. When running a BO algorithm – HEBO in our case – over the HP space of a neural architecture, the HP values are initially sampled uniformly at random, which introduces diversity in the collected samples. As the tuning continues, the Gaussian process surrogates are adapted based on past evaluations and thereby better performing candidates are sampled. This introduces a sampling bias, which is desirable in the context of our application since we are more interested in being accurate in high-performing areas of the HP space. Consequently, we obtained a tabular output indicating the performance of each sampled arch-hyper during the search process. We then replaced each arch-hyper with the corresponding embedding vector and fed it to our meta-predictor.

Design Choices

In the following, we present the result of a set of experiments conducted to select our data encodings and objective function. Therefore, we split the data such that 70% of the records corresponding to each (model, dataset) pair are used for training, and the remaining 30% are equally distributed between the validation and testing sets. We evaluate the meta-predictor based on MSE and Spearman's rank correlation coefficient (SRC) between the true and predicted values, to account for both of our objectives (have an accurate prediction and keep the ranking intact).

Table 1: Performance of meta-predictors trained with different encodings; the best performance is underlined.

Encoding		Spearman's Rank	MSE
Data	Architecture	Correlation	
MFE	arch2vec	0.848 $\pm$0.003	0.018 $\pm$0.001
MFE	AdjOp+GCN	0.841 $\pm$0.005	0.020 $\pm$0.001
MFE	AdjOp+GIN	0.856 $\pm$0.004	0.017 $\pm$0.001
MFE	OHE	0.868 $\pm$0.003	0.015 $\pm$0.001
MFE	OHE+LSTM	0.866 $\pm$0.006	0.015 $\pm$0.001
MFE	OHE+Transformer	0.868 $\pm$0.002	0.015 $\pm$0.001
VAE	arch2vec	0.838 $\pm$0.007	0.019 $\pm$0.001
VAE	AdjOp+GCN	0.837 $\pm$0.001	0.020 $\pm$0.000
VAE	AdjOp+GIN	0.854 $\pm$0.003	0.018 $\pm$0.001
VAE	OHE	0.862 $\pm$0.005	0.016 $\pm$0.001
VAE	OHE+LSTM	0.863 $\pm$0.002	0.016 $\pm$0.000
VAE	OHE+Transformer	0.854 $\pm$0.022	0.018 $\pm$0.004

Architecture and Dataset Encoding. We analysed the impact of different encoding methods on the performance of our meta-predictor. For the dataset encoding, we considered two options: (i) meta-feature encoding (MFE) and (ii) variational autoencoder (VAE). For the architecture encoding, we evaluated 6 methods: (i) one-hot encoding (OHE), (ii) AdjOp + Transformer, (iii) AdjOp + LSTM, (iv) AdjOp + Graph Isomorphism Network (GIN), (v) AdjOp + Graph Convolutional Network (GCN), and (vi) *arch2vec*. This resulted in a total of 12 possible combinations. We trained a meta-predictor for each (data, architecture) encoding pair with 3 different seeds, and compared the performance of the obtained meta-predictors to determine the effectiveness of these encoding schemes. Table 1 summarises the performance results of all encoding combinations. MFE/OHE and MFE/OHE+Transformer resulted in the best relative performance in terms of both regression and ranking criteria. However, due to the computational overhead of transformers, we

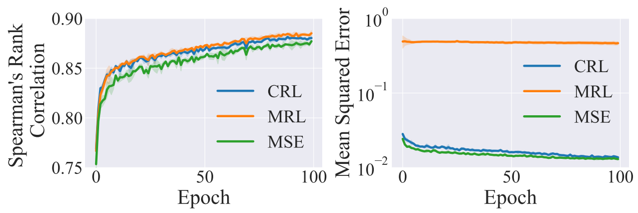


Figure 3: Impact of different loss functions on meta-predictor ranking and regression performance.

Table 2: HPO and meta-prediction performance on forecasting problems in terms of MSE; the best performance is underlined.

Dataset	HEBO	Meta	Change (%)
ETTh1	0.309 $\pm$0.005	0.345 $\pm$0.022	11.7 ↑
ETTh2	0.410 $\pm$0.007	0.268 $\pm$0.017	34.6 ↓
ETTm1	0.210 $\pm$0.005	0.293 $\pm$0.003	39.1 ↑
ETTm2	0.238 $\pm$0.013	0.162 $\pm$0.013	32.0 ↓
Electricity	0.134 $\pm$0.001	0.140 $\pm$0.004	4.6 ↑
Exchange	0.109 $\pm$0.010	0.076 $\pm$0.002	30.3 ↓
Traffic	0.558 $\pm$0.005	0.439 $\pm$0.059	21.3 ↓
Weather	0.105 $\pm$0.008	0.144 $\pm$0.002	37.3 ↑

selected MFE/OHE for the encodings used in all other experiments.

Choice of Objective Function. We also investigated the impact of the objective used in the meta-predictor on its performance. For this purpose, we trained 3 different meta-predictors, each with one of the objective functions introduced in Section 4. We set the margin parameter δ to 1.0 based on preliminary experiments with $\delta \in \{0.0, 0.5, 1.0, 2.0\}$. Figure 3 shows the performance of the 3 trained meta-predictors. MRL excelled in terms of the ranking metric, MSE in regression, and CRL found a strong compromise between the two objectives.

6 Experiments

We conducted independent experiments for time series classification and forecasting tasks. We run each task with three different seeds, applied to both training the meta predictor and conducting the search strategy.

Setup. For each task and each corresponding time series dataset D, we trained a meta-predictor over the performance data collected for all other datasets in the same task. During the training phase, we excluded the dataset on which we aimed to evaluate the meta-predictor and split the remaining data such that 70% were used for training and 15% for validation. The remaining 15% were used as a test set in Section 5, and are not used in this set of experiments. The performance data on the excluded dataset is used to evaluate the generalisation of our method. Our trained meta-predictor is integrated into our meta-search strategy to predict high-performing architecture(s) and their HP settings for each dataset D. Our method can suggest N architectures and K HP settings for each architecture, with N and K being configurable parameters of our method. To ensure the robustness of our results and reduce sensitivity to outliers, we keep the top-5 HP settings ($K = 5$) for the best found architecture ($N = 1$) (other values were considered in supplementary material). The performance comparison between the suggested architecture(s) and those obtained through HPO is shown in Tables 2-4 separately for each task.

Table 3: HPO and meta-prediction performance on UEA classification problems in terms of accuracy; the best performance is underlined.

Dataset	HEBO	Meta	Change (%)
ArticularyWordRecognition	98.4 $\pm$0.2	98.5 $\pm$0.6	0.1 ↑
AtrialFibrillation	73.3 $\pm$0.0	63.8 $\pm$10.7	12.9 ↓
BasicMotions	100.0 $\pm$0.0	97.5 $\pm$0.0	2.5 ↓
CharacterTrajectories	98.6 $\pm$0.1	99.2 $\pm$0.1	0.6 ↑
Cricket	96.9 $\pm$0.6	97.4 $\pm$1.2	0.5 ↑
DuckDuckGeese	64.0 $\pm$0.0	74.9 $\pm$0.0	17.0 ↑
ERing	95.8 $\pm$0.2	93.9 $\pm$0.9	2.0 ↓
Epilepsy	92.2 $\pm$0.4	97.1 $\pm$0.8	5.4 ↑
EthanolConcentration	31.1 $\pm$0.2	44.1 $\pm$3.8	41.8 ↑
FaceDetection	70.5 $\pm$0.1	74.8 $\pm$0.6	6.1 ↑
FingerMovements	62.6 $\pm$1.3	66.5 $\pm$5.6	6.3 ↑
HandMovementDirection	68.4 $\pm$0.7	66.0 $\pm$4.5	3.5 ↓
Handwriting	37.0 $\pm$0.4	30.7 $\pm$18.5	17.2 ↓
Heartbeat	79.8 $\pm$0.6	79.9 $\pm$1.5	0.1 ↑
JapaneseVowels	98.0 $\pm$0.2	98.5 $\pm$0.4	0.5 ↑
LSST	44.2 $\pm$1.5	66.2 $\pm$3.0	49.7 ↑
Libras	86.8 $\pm$0.2	87.4 $\pm$5.6	0.7 ↑
MotorImagery	68.4 $\pm$1.5	65.4 $\pm$0.8	4.4 ↓
NATOPS	96.1 $\pm$1.3	93.6 $\pm$1.0	2.6 ↓
PEMS-SF	88.9 $\pm$0.5	85.2 $\pm$4.2	4.1 ↓
PenDigits	98.7 $\pm$0.1	98.8 $\pm$0.1	0.1 ↑
PhonemeSpectra	16.0 $\pm$0.0	25.6 $\pm$3.1	60.2 ↑
RacketSports	90.1 $\pm$0.5	89.4 $\pm$2.7	0.8 ↓
SelfRegulationSCP1	91.5 $\pm$0.0	93.6 $\pm$0.6	2.3 ↑
SelfRegulationSCP2	59.3 $\pm$1.2	62.8 $\pm$5.4	5.9 ↑
SpokenArabicDigits	99.1 $\pm$0.1	98.5 $\pm$0.2	0.6 ↓
StandWalkJump	66.7 $\pm$0.0	62.8 $\pm$2.6	5.8 ↓
UWaveGestureLibrary	87.8 $\pm$0.2	87.3 $\pm$0.8	0.6 ↓

Results. We report the performance of the meta-predictor and HEBO. Overall, we would like to verify if the meta-predictor can recommend arch-hypers at least as competitive as those proposed by HEBO without performing the time-consuming HPO procedure. On our forecasting datasets (Table 2), the meta-predictor was able to find a significantly better arch-hyper on half of the datasets, whilst on the other half, the opposite was observed. On UEA (Table 3), the meta-predictor outperformed HEBO on 16 out of 28 datasets, with up to 60% improvement on `PhonemeSpectra` and an average improvement of 12.3%. On the others, HEBO performed on average 4.8% better. On UCR (Table 4), the meta-predictor outperformed HEBO on 20 out of 45 datasets with an average improve-

Table 4: HPO and meta-prediction performances on UCR classification problems in terms of accuracy; the best performance is underlined.

Dataset	HEBO	Meta	Change (%)
Adiac	80.1 ±0.3	73.0 ±6.0	8.8 ↓
BME	100.0 ±0.0	99.4 ±0.5	0.6 ↓
Beef	83.3 ±0.0	85.1 ±2.7	2.2 ↑
CBF	99.9 ±0.1	98.6 ±1.0	1.4 ↓
Car	84.7 ±0.8	86.1 ±2.2	1.7 ↑
Coffee	100.0 ±0.0	100.0 ±0.0	0.0 ↕
CricketX	80.4 ±0.9	82.1 ±2.5	2.2 ↑
CricketY	79.0 ±0.6	79.7 ±2.4	0.8 ↑
Crop	76.6 ±0.1	80.8 ±1.6	5.5 ↑
DiatomSizeReduction	99.7 ±0.0	97.1 ±0.7	2.6 ↓
DodgerLoopWeekend	98.6 ±0.0	98.9 ±0.2	0.3 ↑
ECG200	92.4 ±0.5	93.2 ±1.6	0.9 ↑
ECG5000	94.3 ±0.0	97.1 ±0.1	2.9 ↑
EOGVerticalSignal	52.0 ±0.9	52.9 ±4.3	1.7 ↑
ElectricDevices	75.4 ±0.3	78.9 ±1.7	4.7 ↑
FaceAll	91.2 ±0.1	85.0 ±5.2	6.8 ↓
FaceFour	96.1 ±1.5	90.5 ±1.3	5.8 ↓
Fish	94.9 ±0.4	93.8 ±1.8	1.1 ↓
FreezerRegularTrain	99.8 ±0.1	99.2 ±0.2	0.6 ↓
Fungi	96.1 ±0.7	96.1 ±1.9	0.0 ↕
GestureMidAirD1	70.8 ±0.5	69.9 ±2.5	1.3 ↓
GesturePebbleZ2	92.3 ±1.4	87.8 ±2.4	4.9 ↓
GunPointAgeSpan	100.0 ±0.0	99.4 ±0.3	0.6 ↓
GunPointMaleVersusFemale	100.0 ±0.0	99.8 ±0.2	0.2 ↓
HouseTwenty	99.2 ±0.0	95.6 ±2.2	3.6 ↓
InlineSkate	47.1 ±0.1	39.0 ±4.7	17.3 ↓
InsectEPGSmallTrain	91.2 ±0.9	89.5 ±2.9	1.8 ↓
Lightning2	86.6 ±0.6	83.8 ±3.1	3.2 ↓
Lightning7	85.6 ±5.6	94.5 ±1.4	10.4 ↑
Mallat	81.5 ±0.2	83.2 ±1.2	2.2 ↑
MedicalImages	97.3 ±0.1	97.5 ±0.2	0.2 ↑
MelbournePedestrian	84.6 ±0.6	85.9 ±1.1	1.5 ↑
MiddlePhalanxOutlineCorrect	94.0 ±0.1	94.0 ±0.7	0.0 ↕
MixedShapesRegularTrain	88.0 ±0.0	80.2 ±4.0	8.8 ↓
PickupGestureWiimoteZ	100.0 ±0.0	100.0 ±0.0	0.0 ↕
Plane	98.3 ±0.0	100.0 ±0.0	1.7 ↑
PowerCons	97.0 ±0.3	94.1 ±1.4	3.0 ↓
SemgHandGenderCh2	89.9 ±0.3	92.2 ±1.1	2.6 ↑
ShapesAll	78.9 ±0.2	79.6 ±1.4	0.9 ↑
SmallKitchenAppliances	97.7 ±0.1	97.8 ±0.1	0.1 ↑
StarLightCurves	97.0 ±0.2	97.1 ±0.4	0.0 ↕
Strawberry	94.4 ±0.2	95.9 ±0.7	1.6 ↑
SwedishLeaf	77.9 ±0.3	82.6 ±0.3	6.1 ↑
UWaveGestureLibraryZ	99.9 ±0.0	99.7 ±0.1	0.2 ↓
Wafer	99.9 ±0.0	99.7 ±0.1	0.2 ↓

ment of 2.5%, on 5 datasets it reached the same performance, and on the remaining 20 it was outperformed by an average of 3.6%. To assess the overall statistical significance of these differences, we conducted a Wilcoxon signed-rank test with a standard significance threshold of 0.05, which detected no significant difference between the performance of the two methods for any of the two tasks. This confirms that our proposed framework is competitive against HEBO.

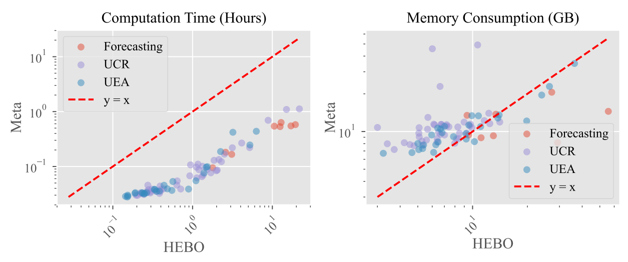


Figure 4: Efficiency comparison between HPO and meta-prediction approach in terms of computation time (hours) and memory consumption (gigabytes).

Figure 4 compares the time and memory consumption of HEBO against our meta-search strategy, demonstrating the clear advantages of our approach. Similar performing arch-hypers, if not better ones, were found using on average 5% for forecasting and 10% for classification of the time required for HEBO. On classification datasets, HEBO is more efficient in terms of memory usage – probably due to the memory required to store the encodings of the whole search space of arch-hypers in our method –, but values are still comparable as seen in the right plot in Figure 4. On forecasting, the meta-predictor uses in average 78% of the amount of memory required by HEBO.

7 Conclusion and Future Work

In this work, we examined the ability of a meta-learning approach to automatically determine effective deep learning models along with their hyperparameters for time series classification and forecasting tasks. Our empirical results clearly indicate that incorporating past knowledge from historical observations is viable in finding potentially high-performing architectures and HP configurations for new datasets, using on average 10% of the computational cost. To build our meta-predictor, we evaluated three possible objective functions and designed the CRL representing the best compromise between the ranking and regression metrics used to train the meta-predictor. Similarly, we evaluated the impact of different architectures and dataset encoding methods on the performance of the meta-predictor. Moreover, we analysed the role of different encoding schemes and showed that a simple one-hot encoding for architectures and meta-feature encoding for datasets contains enough information for the meta-predictor to predict performance. Future work should explore alternative encoding schemes that capture information flow during forward and backward passes (Akhauri and Abdelfattah 2024; Hwang et al. 2024), preserving similarities between structurally different but computationally similar architectures. Additionally, since our meta-predictor can, in theory, predict the performance of previously unseen architectures, it might provide value as a surrogate model within a NAS method.

Acknowledgements

We acknowledge support through an Alexander von Humboldt Professorship in Artificial Intelligence held by Holger Hoos. This work has been conducted during an internship at Volkswagen Aktiengesellschaft.

Disclaimer The results, opinions and conclusions expressed in this publication are not necessarily those of Volkswagen Aktiengesellschaft.

References

Akhauri, Y.; and Abdelfattah, M. S. 2024. Encodings for Prediction-based Neural Architecture Search. In *Proceedings of the International Conference on Machine Learning*.

Bagnall, A.; Dau, H. A.; Lines, J.; Flynn, M.; Large, J.; Bostrom, A. G.; Southam, P.; and Keogh, E. J. 2018. The UEA multivariate time series classification archive, 2018. arXiv:1811.00075v1.

Baratchi, M.; Wang, C.; Limmer, S.; Van Rijn, J. N.; Hoos, H.; Bäck, T.; and Olhofer, M. 2024. Automated machine learning: past, present and future. *Artificial Intelligence Review*.

Bergstra, J.; and Bengio, Y. 2012. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*.

Bergstra, J.; Komer, B.; Eliasmith, C.; Yamins, D.; and Cox, D. D. 2015. Hyperopt: a Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*.

Brazdil, P.; van Rijn, J.; Soares, C.; and Vanschoren, J. 2022. *Metalearning: Applications to Automated Machine Learning and Data Mining*. Springer.

Chen, S.-A.; Li, C.-L.; Arik, S. O.; Yoder, N. C.; and Pfister, T. 2023. TSMixer: An All-MLP Architecture for Time Series Forecast-ing. *Transactions on Machine Learning Research*.

Cowen-Rivers, A. I.; Lyu, W.; Tutunov, R.; Wang, Z.; Grosnit, A.; Griffiths, R. R.; Maraval, A. M.; Jianye, H.; Wang, J.; Peters, J.; and Bou-Ammar, H. 2022. HEBO: Pushing The Limits of Sample-Efficient Hyper-parameter Optimisation. *Journal of Artificial Intelligence Research*.

Dau, H. A.; Bagnall, A.; Kamgar, K.; Yeh, C.-C. M.; Zhu, Y.; Gharghabi, S.; Ratanamahatana, C. A.; and Keogh, E. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*.

Deng, D.; Karl, F.; Hutter, F.; Bischl, B.; and Lindauer, M. 2022. Efficient Automated Deep Learning for Time Series Forecasting. In *Proceedings of the Machine Learning and Knowledge Discovery in Databases Conference*.

Donghao, L.; and Xue, W. 2024. ModernTCN: A Modern Pure Convolution Structure for General Time Series Analysis. In *Proceedings of The International Conference on Learning Representations*.

Erickson, N.; Mueller, J. W.; Shirkov, A.; Zhang, H.; Larroy, P.; Li, M.; and Smola, A. 2020. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data. arXiv:2003.06505v1.

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the International Conference on Machine Learning*.

Gu, A.; and Dao, T. 2024. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *Proceedings of the First Conference on Language Modeling*.

Hasani, R. M.; Lechner, M.; Wang, T.; Chahine, M.; Amini, A.; and Rus, D. 2023. Liquid Structural State-Space Models. In *Proceedings of the International Conference on Learning Representations*.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the International Conference on Learning and Intelligent Optimization*.

Hutter, F.; Kotthoff, L.; and Vanschoren, J. 2019. *Automated Machine Learning: Methods, Systems, Challenges*. Springer.

Hwang, D.; Kim, H.; Kim, S.; and Shin, K. 2024. FlowerFormer: Empowering Neural Architecture Encoding Using a Flow-Aware Graph Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Ismail Fawaz, H.; Forestier, G.; Weber, J.; Idoumghar, L.; and Muller, P.-A. 2019. Deep learning for time series classification: a review. *Data Mining Knowledge Discovery*.

Kitaev, N.; Kaiser, L.; and Levskaya, A. 2020. Reformer: The Efficient Transformer. In *Proceedings of the International Conference on Learning Representations*.

Lee, H.; Hyung, E.; and Hwang, S. J. 2021. Rapid Neural Architecture Search by Learning to Generate Graphs from Datasets. In *Proceedings of The International Conference on Learning Representations*.

Lin, S.; Lin, W.; Wu, W.; Zhao, F.; Mo, R.; and Zhang, H. 2023. SegRNN: Segment Recurrent Neural Network for Long-Term Time Series Forecasting. arXiv:2308.11200v1.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *Proceedings of the International Conference on Learning Representations*.

Liu, Y.; Hu, T.; Zhang, H.; Wu, H.; Wang, S.; Ma, L.; and Long, M. 2024. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting. In *Proceedings of The International Conference on Learning Representations*.

Middlehurst, M.; Schäfer, P.; and Bagnall, A. 2024. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*.

Mu, T.; Wang, H.; Zheng, S.; Liang, Z.; Wang, C.; Shao, X.; and Liang, Z. 2023. TSC-AutoML: Meta-learning for Automatic Time Series Classification Algorithm Selection. In *Proceedings of the IEEE International Conference on Data Engineering*.

Mung, P. S.; and Phyu, S. 2023. Time Series Weather Data Forecasting Using Deep Learning. In *Proceedings of the IEEE Conference on Computer Applications*.

Nie, Y.; H. Nguyen, N.; Sinthong, P.; and Kalagnanam, J. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *Proceedings of the International Conference on Learning Representations*.

Shala, G.; Elsken, T.; Hutter, F.; and Grabocka, J. 2023. Transfer NAS with Meta-learned Bayesian Surrogates. In *Proceedings of the International Conference on Learning Representations*.

Shchur, O.; Turkmen, A. C.; Erickson, N.; Shen, H.; Shirkov, A.; Hu, T.; and Wang, B. 2023. AutoGluon–TimeSeries: AutoML for Probabilistic Time Series Forecasting. In *Proceedings of the International Conference on Automated Machine Learning*.

Skaf, W.; Tosayeva, A.; and Várkonyi, D. T. 2022. Towards Automatic Forecasting: Evaluation of Time-Series Forecasting Models for Chickenpox Cases Estimation in Hungary. In *Proceedings of the International Conference on Intelligent Systems Design and Applications*.

Talkhi, N.; Akhavan Fatemi, N.; Jabbari Nooghabi, M.; Soltani, E.; and Jabbari Nooghabi, A. 2024. Using meta-learning to recommend an appropriate time-series forecasting model. *BMC Public Health*, 24(1): 148.

Thornton, C.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining*.

Turner, R.; Eriksson, D.; McCourt, M.; Kiili, J.; Laaksonen, E.; Xu, Z.; and Guyon, I. 2021. Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*.

Uddin, S.; and Lu, H. 2024. Dataset meta-level and statistical features affect machine learning performance. *Scientific Reports*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In *Proceedings of Advances in Neural Information Processing Systems*.

Voelker, A.; Kajic, I.; and Eliasmith, C. 2019. Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks. In *Proceedings of the International Conference on Neural Information Processing Systems*.

White, C.; Neiswanger, W.; Nolen, S.; and Savani, Y. 2020. A Study on Encodings for Neural Architecture Search. In *Proceedings of Advances in Neural Information Processing Systems*.

White, C.; Neiswanger, W.; and Savani, Y. 2021. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Woo, G.; Liu, C.; Sahoo, D.; Kumar, A.; and Hoi, S. 2022. ETSformer: Exponential Smoothing Transformers for Time-series Forecasting. arXiv:2202.01381v2.

Wu, H.; Hu, T.; Liu, Y.; Zhou, H.; Wang, J.; and Long, M. 2023a. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. In *Proceedings of the International Conference on Learning Representations*.

Wu, H.; Xu, J.; Wang, J.; and Long, M. 2021. Autoformer: decomposition transformers with auto-correlation for long-term series forecasting. In *Proceedings of the International Conference on Neural Information Processing Systems*.

Wu, X.; Zhang, D.; Zhang, M.; Guo, C.; Yang, B.; and Jensen, C. S. 2023b. AutoCTS+: Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting. In *Proceedings of the ACM on Management of Data*.

Yue, Z.; Wang, Y.; Duan, J.; Yang, T.; Huang, C.; Tong, Y.; and Xu, B. 2022. TS2Vec: Towards Universal Representation of Time Series. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zeng, A.; Chen, M.; Zhang, L.; and Xu, Q. 2023. Are transformers effective for time series forecasting? In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, T.; Zhang, Y.; Cao, W.; Bian, J.; Yi, X.; Zheng, S.; and Li, J. 2022. Less Is More: Fast Multivariate Time Series Forecasting with Light Sampling-oriented MLP Structures. arXiv:2207.01186v1.

Zhang, Y.; and Yan, J. 2023. Crossformer: Transformer Utilizing Cross-Dimension Dependency for Multivariate Time Series Forecasting. In *Proceedings of the International Conference on Learning Representations*.

Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; and Zhang, W. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhou, T.; Ma, Z.; Wang, X.; Wen, Q.; Sun, L.; Yao, T.; Yin, W.; and Jin, R. 2022. FiLM: Frequency improved Legendre Memory Model for Long-term Time Series Forecasting. In *Proceedings of the International Conference on Neural Information Processing Systems*.

Zimmer, L.; Lindauer, M.; and Hutter, F. 2021. Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

In what follows, we provide supplementary materials for the paper titled **Neural Architecture and Hyperparameter Selection through Meta-Learning on Time Series**. We break this text into 6 sections, providing details over the following: (i) extra experiment results, (ii) dataset description, (iii) neural architectures and their hyperparameters (HPs) space, (iv) time-series data encoding via meta-features, (v) implementation details, and (vi) implementation of proposed algorithms

A Full Results

A.1 Meta-Search Experiments

In the main text, we only included the comparison results for the efficacy of hyperparameter optimisation (HPO) using HEBO and the meta-prediction approach using the setting with $N = 1$ architecture and $K = 5$ HP configurations, hereafter denoted as $1/5$ (N/K) setting. In this section, we provide the complete results using different settings to evaluate the robustness of the meta-prediction approach: (i) $N = All$, $K = 5$; (ii) $N = All$, $K = 1$; (iii) $N = 1$, $K = 5$; and (iv) $N = 1$, $K = 1$. As Table 1 represents, the meta-prediction approach surpasses HEBO in terms of the performance of suggested configurations under all proposed settings. The best performance improvement is achieved by the one-shot setting, indicating that the meta-predictor may suggest more promising HP configurations for certain architectures. For the classification tasks, we observe only marginal performance change when comparing the meta-prediction approach to HEBO. Table 2 summarizes the results for the multivariate classification tasks. Accordingly, we observe performance improvement only for the $1/5$ and $1/1$ settings, and slight performance degradation for the remaining settings. We observe similar pattern for the univariate UCR classification tasks with one slight difference, i.e., the performance improvement is observed for the $All/1$ and $1/1$ settings.

As already specified in the main text, our meta-learning approach can reach performance results comparable to those obtained using HEBO with only 10% computation time on average over all datasets. For the full efficiency results, we provide the *efficiency_table.csv* file along with this supplementary document which compares HEBO and the meta-predictor in terms of memory and computation time for each dataset with uncertainty values calculated based on the error propagation method. Accordingly, for the classification task, the average computation time ratio of meta-predictor to HEBO is 0.098 ± 0.181, and the average memory consumption ratio is 1.578 ± 6.310. Similarly, for the forecasting task, the average computation time and memory consumption ratio of meta-predictor to HEBO are 0.048 ± 0.067 and 0.783 ± 1.557, respectively. The results strongly indicate the superiority of the meta-predictor in terms of computation time. However, the improvement in terms of memory consumption is less significant, which can be attributed to the large arch-hyper search space and data encoding vectors that are traversed during the meta-search phase.

A.2 Hyperparameter Optimisation Experiments

For the sake of completeness, we also provide the full results of HPO experiments, which summarizes the performance of each of the benchmarked models on the time-series datasets that we considered throughout this paper. Table 4 specifies the results summary for classification datasets, including per-model results along with the final rankings based on the average performance across all datasets. Similarly, Table 5 provides similar information for the forecasting datasets. Accordingly, we observe that *TS2Vec* obtains best performance on average over all the classification datasets (including both UEA and UCR benchmarks). Finally, the *S4* state-space model achieves state-of-the-art performance on the forecasting datasets. The bootstrapping cost of building the meta-dataset involves running HEBO for 128 trials per model and dataset combination. Given that we evaluated multiple models across a wide range of datasets, this process can be computationally intensive (roughly 200 GPU-days of compute using NVIDIA H100 GPU). However, this initial investment is offset by the efficiency gains achieved through the meta-predictor, which can provide effective HP recommendations with significantly reduced computational cost in subsequent HPO tasks.

Table 1: Comparison of performance results between HPO and the meta-prediction approach based on different settings for forecasting task. Each setting represents the number of top architectures (N) and HP configurations K that are used to obtain the results.

Dataset / Config	$N = All, K = 5$			$N = All, K = 1$			$N = 1, K = 5$			$N = 1, K = 1$		
	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)
ETTh1	0.557 ± 0.253	$\underline{0.531 \pm 0.230}$	$4.8 \downarrow$	0.557 ± 0.277	$\underline{0.518 \pm 0.234}$	$7.1 \downarrow$	$\underline{0.309 \pm 0.005}$	0.345 ± 0.022	$11.7 \uparrow$	$\underline{0.313}$	0.339 ± 0.013	$8.3 \uparrow$
ETTh2	$\underline{0.933 \pm 1.101}$	0.949 ± 1.100	$1.8 \uparrow$	$\underline{0.855 \pm 0.896}$	0.896 ± 0.941	$4.8 \uparrow$	0.410 ± 0.007	$\underline{0.268 \pm 0.017}$	$34.6 \downarrow$	0.410	$\underline{0.273 \pm 0.006}$	$33.5 \downarrow$
ETTm1	$\underline{0.435 \pm 0.171}$	0.449 ± 0.174	$3.2 \uparrow$	0.441 ± 0.192	$\underline{0.430 \pm 0.171}$	$2.7 \downarrow$	$\underline{0.210 \pm 0.005}$	0.293 ± 0.003	$39.1 \uparrow$	$\underline{0.206}$	0.291 ± 0.003	$41.3 \uparrow$
ETTm2	0.374 ± 0.306	$\underline{0.314 \pm 0.200}$	$16.1 \downarrow$	0.411 ± 0.339	$\underline{0.305 \pm 0.213}$	$25.9 \downarrow$	0.238 ± 0.013	$\underline{0.162 \pm 0.013}$	$32.0 \downarrow$	0.231	$\underline{0.160 \pm 0.004}$	$30.7 \downarrow$
Electricity	$\underline{0.197 \pm 0.071}$	0.217 ± 0.091	$9.9 \uparrow$	$\underline{0.191 \pm 0.066}$	0.206 ± 0.084	$7.6 \uparrow$	$\underline{0.134 \pm 0.001}$	0.140 ± 0.004	$4.6 \uparrow$	$\underline{0.133}$	0.139 ± 0.001	$4.5 \uparrow$
Exchange	0.515 ± 0.469	$\underline{0.327 \pm 0.396}$	$36.5 \downarrow$	0.476 ± 0.383	$\underline{0.278 \pm 0.259}$	$41.6 \downarrow$	0.109 ± 0.010	$\underline{0.076 \pm 0.002}$	$30.3 \downarrow$	0.105	$\underline{0.075 \pm 0.002}$	$28.3 \downarrow$
Traffic	$\underline{0.550 \pm 0.121}$	0.640 ± 0.169	$16.4 \uparrow$	$\underline{0.547 \pm 0.126}$	0.588 ± 0.133	$7.4 \uparrow$	0.558 ± 0.005	$\underline{0.439 \pm 0.059}$	$21.3 \downarrow$	0.559	$\underline{0.403 \pm 0.015}$	$27.9 \downarrow$
Weather	0.297 ± 0.222	$\underline{0.223 \pm 0.152}$	$24.9 \downarrow$	0.297 ± 0.216	$\underline{0.218 \pm 0.153}$	$26.6 \downarrow$	$\underline{0.105 \pm 0.008}$	0.144 ± 0.002	$37.3 \uparrow$	$\underline{0.103}$	0.141 ± 0.002	$36.9 \uparrow$
Avg	0.482 ± 0.222	$\underline{0.456 \pm 0.248}$	$5.4 \downarrow$	0.472 ± 0.198	$\underline{0.430 \pm 0.234}$	$9.0 \downarrow$	0.259 ± 0.160	$\underline{0.233 \pm 0.123}$	$10.0 \downarrow$	0.258 ± 0.162	$\underline{0.228 \pm 0.115}$	$11.6 \downarrow$

Table 2: Comparison of performance results between HPO and the meta-prediction approach based on different settings for multivariate classification datasets (UEA). Each setting represents the number of top architectures (N) and HP configurations K that are used to obtain the results.

Dataset / Config	$N = All$, $K = 5$			$N = All$, $K = 1$			$N = 1$, $K = 5$			$N = 1$, $K = 1$		
	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)
ArticularyWordRecognition	$\underline{98.6 \pm 0.5}$	97.4 ± 1.1	$1.2 \downarrow$	$\underline{98.7 \pm 0.6}$	97.9 ± 0.6	$0.9 \downarrow$	98.4 ± 0.2	$\underline{98.5 \pm 0.6}$	$0.1 \uparrow$	98.5 ± 0.3	$\underline{98.8 \pm 0.2}$	$0.3 \uparrow$
AtrialFibrillation	$\underline{69.1 \pm 17.2}$	59.8 ± 7.3	$13.4 \downarrow$	$\underline{69.3 \pm 19.2}$	61.6 ± 6.7	$11.1 \downarrow$	$\underline{73.3 \pm 0.0}$	63.8 ± 10.7	$12.9 \downarrow$	$\underline{83.3 \pm 14.1}$	64.2 ± 12.4	$22.9 \downarrow$
BasicMotions	$\underline{100.0 \pm 0.0}$	94.2 ± 4.0	$5.8 \downarrow$	$\underline{100.0 \pm 0.0}$	96.2 ± 2.1	$3.8 \downarrow$	$\underline{100.0 \pm 0.0}$	97.5 ± 0.0	$2.5 \downarrow$	$\underline{100.0 \pm 0.0}$	97.5 ± 0.0	$2.5 \downarrow$
CharacterTrajectories	$\underline{98.7 \pm 0.4}$	98.3 ± 0.7	$0.4 \downarrow$	$\underline{98.8 \pm 0.5}$	98.6 ± 0.5	$0.2 \downarrow$	98.6 ± 0.1	$\underline{99.2 \pm 0.1}$	$0.6 \uparrow$	98.6 ± 0.1	$\underline{99.3 \pm 0.2}$	$0.7 \uparrow$
Cricket	$\underline{97.6 \pm 1.3}$	93.6 ± 2.5	$4.1 \downarrow$	$\underline{97.8 \pm 1.3}$	95.1 ± 1.8	$2.7 \downarrow$	96.9 ± 0.6	$\underline{97.4 \pm 1.2}$	$0.5 \uparrow$	97.2 ± 0.0	$\underline{97.7 \pm 0.8}$	$0.5 \uparrow$
DuckDuckGeese	$\underline{62.8 \pm 1.9}$	62.3 ± 10.1	$0.9 \downarrow$	63.6 ± 1.7	$\underline{63.7 \pm 7.1}$	$0.1 \uparrow$	64.0 ± 0.0	$\underline{74.9 \pm 0.0}$	$17.0 \uparrow$	63.0 ± 1.4	$\underline{74.9 \pm 0.0}$	$18.9 \uparrow$
ERing	$\underline{95.2 \pm 0.8}$	90.6 ± 4.5	$4.8 \downarrow$	$\underline{95.3 \pm 1.0}$	93.7 ± 2.4	$1.7 \downarrow$	$\underline{95.8 \pm 0.2}$	93.9 ± 0.9	$2.0 \downarrow$	$\underline{95.9 \pm 0.0}$	94.6 ± 0.5	$1.3 \downarrow$
EigenWorms	$\underline{73.3 \pm 11.6}$	63.6 ± 6.3	$13.2 \downarrow$	$\underline{74.0 \pm 12.8}$	65.4 ± 6.2	$11.7 \downarrow$	-	-	-	$\underline{68.7 \pm 0.0}$	67.2 ± 1.3	$2.1 \downarrow$
Epilepsy	$\underline{93.2 \pm 2.6}$	91.2 ± 4.5	$2.2 \downarrow$	$\underline{93.9 \pm 2.7}$	92.9 ± 3.6	$1.1 \downarrow$	92.2 ± 0.4	$\underline{97.1 \pm 0.8}$	$5.4 \uparrow$	93.2 ± 0.5	$\underline{97.6 \pm 0.4}$	$4.7 \uparrow$
EthanolConcentration	31.8 ± 1.2	$\underline{42.1 \pm 2.7}$	$32.6 \uparrow$	32.3 ± 1.2	$\underline{43.9 \pm 2.4}$	$35.9 \uparrow$	31.1 ± 0.2	$\underline{44.1 \pm 3.8}$	$41.8 \uparrow$	31.4 ± 0.3	$\underline{46.4 \pm 3.8}$	$47.8 \uparrow$
FaceDetection	67.3 ± 6.5	$\underline{73.3 \pm 3.1}$	$8.9 \uparrow$	67.6 ± 7.0	$\underline{74.2 \pm 1.9}$	$9.9 \uparrow$	70.5 ± 0.1	$\underline{74.8 \pm 0.6}$	$6.1 \uparrow$	70.6 ± 0.2	$\underline{75.6 \pm 0.1}$	$7.2 \uparrow$
FingerMovements	62.8 ± 2.4	$\underline{64.7 \pm 5.2}$	$3.0 \uparrow$	64.0 ± 2.3	$\underline{68.4 \pm 5.1}$	$6.8 \uparrow$	62.6 ± 1.3	$\underline{66.5 \pm 5.6}$	$6.3 \uparrow$	65.0 ± 0.0	$\underline{73.5 \pm 0.0}$	$13.1 \uparrow$
HandMovementDirection	$\underline{61.7 \pm 11.3}$	61.1 ± 7.2	$1.0 \downarrow$	62.7 ± 12.4	$\underline{65.6 \pm 3.9}$	$4.6 \uparrow$	$\underline{68.4 \pm 0.7}$	66.0 ± 4.5	$3.5 \downarrow$	$\underline{68.9 \pm 0.0}$	67.7 ± 1.1	$1.8 \downarrow$
Handwriting	$\underline{40.2 \pm 9.7}$	29.1 ± 11.0	$27.6 \downarrow$	$\underline{40.7 \pm 10.9}$	35.7 ± 8.6	$12.3 \downarrow$	$\underline{37.0 \pm 0.4}$	30.7 ± 18.5	$17.2 \downarrow$	37.8 ± 0.4	$\underline{43.7 \pm 1.9}$	$15.5 \uparrow$
Heartbeat	78.4 ± 1.2	$\underline{79.1 \pm 2.1}$	$0.9 \uparrow$	79.0 ± 1.2	$\underline{79.6 \pm 1.9}$	$0.7 \uparrow$	79.8 ± 0.6	$\underline{79.9 \pm 1.5}$	$0.1 \uparrow$	79.2 ± 1.8	$\underline{80.4 \pm 1.8}$	$1.4 \uparrow$
JapaneseVowels	$\underline{98.1 \pm 0.7}$	97.0 ± 1.0	$1.1 \downarrow$	$\underline{98.3 \pm 0.7}$	97.2 ± 0.8	$1.1 \downarrow$	98.0 ± 0.2	$\underline{98.5 \pm 0.4}$	$0.5 \uparrow$	97.9 ± 0.2	$\underline{98.6 \pm 0.6}$	$0.7 \uparrow$
LSST	45.8 ± 8.2	$\underline{59.8 \pm 6.9}$	$30.7 \uparrow$	46.4 ± 8.7	$\underline{61.6 \pm 7.9}$	$32.6 \uparrow$	44.2 ± 1.5	$\underline{66.2 \pm 3.0}$	$49.7 \uparrow$	44.2 ± 2.5	$\underline{67.7 \pm 4.1}$	$53.2 \uparrow$
Libras	85.1 ± 2.9	$\underline{85.7 \pm 4.7}$	$0.7 \uparrow$	86.0 ± 2.7	$\underline{87.8 \pm 4.0}$	$2.0 \uparrow$	86.8 ± 0.2	$\underline{87.4 \pm 5.6}$	$0.7 \uparrow$	85.6 ± 2.3	$\underline{89.3 \pm 1.0}$	$4.3 \uparrow$
MotorImagery	$\underline{65.2 \pm 3.2}$	64.8 ± 3.1	$0.7 \downarrow$	66.6 ± 3.8	$\underline{67.5 \pm 3.1}$	$1.4 \uparrow$	$\underline{68.4 \pm 1.5}$	65.4 ± 0.8	$4.4 \downarrow$	69.0 ± 1.4	$\underline{70.0 \pm 4.1}$	$1.4 \uparrow$
NATOPS	$\underline{95.0 \pm 1.7}$	91.9 ± 2.6	$3.2 \downarrow$	$\underline{95.9 \pm 2.1}$	93.4 ± 1.7	$2.6 \downarrow$	$\underline{96.1 \pm 1.3}$	93.6 ± 1.0	$2.6 \downarrow$	$\underline{95.6 \pm 3.9}$	94.6 ± 0.3	$1.0 \downarrow$
PEMS-SF	$\underline{87.9 \pm 1.3}$	82.4 ± 5.0	$6.2 \downarrow$	$\underline{89.1 \pm 1.3}$	85.4 ± 3.9	$4.2 \downarrow$	$\underline{88.9 \pm 0.5}$	85.2 ± 4.2	$4.1 \downarrow$	$\underline{89.3 \pm 0.4}$	87.9 ± 4.7	$1.5 \downarrow$
PenDigits	$\underline{98.6 \pm 0.3}$	98.0 ± 0.6	$0.6 \downarrow$	$\underline{98.7 \pm 0.4}$	98.3 ± 0.4	$0.4 \downarrow$	98.7 ± 0.1	$\underline{98.8 \pm 0.1}$	$0.1 \uparrow$	98.4 ± 0.5	$\underline{98.9 \pm 0.1}$	$0.5 \uparrow$
PhonemeSpectra	$\underline{17.0 \pm 5.1}$	15.4 ± 5.4	$9.8 \downarrow$	$\underline{17.2 \pm 5.6}$	16.3 ± 5.8	$5.2 \downarrow$	16.0 ± 0.0	$\underline{25.6 \pm 3.1}$	$60.2 \uparrow$	14.8 ± 1.7	$\underline{27.3 \pm 0.9}$	$84.2 \uparrow$
RacketSports	$\underline{89.6 \pm 1.6}$	85.8 ± 2.5	$4.3 \downarrow$	$\underline{90.4 \pm 1.9}$	87.1 ± 2.8	$3.7 \downarrow$	$\underline{90.1 \pm 0.5}$	89.4 ± 2.7	$0.8 \downarrow$	89.2 ± 2.3	$\underline{91.4 \pm 1.7}$	$2.6 \uparrow$
SelfRegulationSCP1	91.0 ± 2.4	$\underline{93.4 \pm 0.8}$	$2.6 \uparrow$	91.3 ± 2.7	$\underline{93.9 \pm 0.4}$	$2.8 \uparrow$	91.5 ± 0.0	$\underline{93.6 \pm 0.6}$	$2.3 \uparrow$	92.5 ± 1.4	$\underline{94.1 \pm 0.7}$	$1.7 \uparrow$
SelfRegulationSCP2	59.0 ± 0.9	$\underline{62.2 \pm 3.4}$	$5.4 \uparrow$	60.0 ± 1.1	$\underline{64.1 \pm 3.2}$	$6.8 \uparrow$	59.3 ± 1.2	$\underline{62.8 \pm 5.4}$	$5.9 \uparrow$	60.6 ± 0.8	$\underline{65.9 \pm 7.2}$	$8.9 \uparrow$
SpokenArabicDigits	$\underline{99.0 \pm 0.2}$	98.3 ± 0.3	$0.7 \downarrow$	$\underline{99.2 \pm 0.2}$	98.5 ± 0.4	$0.7 \downarrow$	$\underline{99.1 \pm 0.1}$	98.5 ± 0.2	$0.6 \downarrow$	$\underline{99.3 \pm 0.1}$	98.6 ± 0.2	$0.7 \downarrow$
StandWalkJump	$\underline{70.9 \pm 5.0}$	59.8 ± 8.7	$15.7 \downarrow$	$\underline{72.0 \pm 5.6}$	63.7 ± 9.4	$11.5 \downarrow$	$\underline{66.7 \pm 0.0}$	62.8 ± 2.6	$5.8 \downarrow$	$\underline{73.4 \pm 9.4}$	66.8 ± 5.6	$9.0 \downarrow$
UWaveGestureLibrary	$\underline{88.3 \pm 2.0}$	86.3 ± 3.9	$2.2 \downarrow$	$\underline{88.9 \pm 2.0}$	87.8 ± 2.9	$1.3 \downarrow$	$\underline{87.8 \pm 0.2}$	87.3 ± 0.8	$0.6 \downarrow$	87.9 ± 0.2	$\underline{89.8 \pm 1.2}$	$2.1 \uparrow$
Avg	$\underline{76.6 \pm 22.5}$	75.2 ± 21.7	$1.8 \downarrow$	$\underline{77.2 \pm 22.4}$	77.1 ± 21.0	$0.1 \downarrow$	77.2 ± 23.1	$\underline{78.6 \pm 20.9}$	$1.8 \uparrow$	77.6 ± 22.7	$\underline{80.0 \pm 19.1}$	$3.2 \uparrow$

B Dataset Description

In this section, we provide a detailed description of the datasets that we used for evaluating HPO (using HEBO) and the meta-prediction experiments for both the classification and forecasting tasks. We use the same datasets to evaluate the efficacy of our proposed meta-search approach in comparison to HPO.

For time-series forecasting, we fix the prediction horizon and set it to 192 within our experiments, which is a mid-range value frequently used in this domain. However, we allow the look-back window to be tuned during HPO experiments. As indicated in the work by Zhou et al. (2021), this is mainly because of the fact that different models might need different amount of historical information to predict the future values more accurately.

Moreover, for multivariate classification task, we select all datasets available in the UEA benchmark (Bagnall et al. 2018), excluding 2 out of 30 datasets (see Table 7): (i) *InsectWingbeat*, due to considerable training times required by time-series models, and (ii) *EigenWorms*, due to out-of-memory condition encountered when using both HEBO and the meta-predictor.

For the univariate classification datasets (UCR) (Dau et al. 2019), we randomly sample one third of datasets from each subject category specified by its authors, such as *Motion*, *Audio*, *Sensor*, etc. If the number of datasets for a category falls below 3 – thereby sampling one-third results in no datasets – we randomly choose exactly one dataset as a representative for that subject. This results in a total of 45 out of 128 datasets (see Table 8).

C Meta-Feature Encoding

In this section, we summarize the meta-features that we use to represent the characteristics of time-series datasets. These meta-features are listed in Table 9. Furthermore, we delve deeper into how we encode time-series datasets using these meta-features while ensuring consistency in the dimensionality of the encoding vectors across all datasets.

Assuming that the original dataset is a multivariate time series of length L with N variables, denoted as $D \in \mathbb{R}^{L \times N}$, the encoded dataset using K meta-features can be denoted as a vector $d \in \mathbb{R}^{NK}$. This means that the characteristics of each single variable can be represented by K meta-features, resulting in a vector of size NK. Given that the number of variables N might vary across different datasets, the final dimensionality of encoded datasets based on the meta-features (which are calculated per variable) will also vary. In order to ensure consistency in the dimension of encoded datasets, we apply different aggregations on relevant meta-features across all attributes. We consider the following 7 aggregation operations: (i) mean (μ); (ii) standard deviation (σ); (iii) minimum ($\min$); (iv) maximum ($\max$); (v) first quartile ($Q1$); (vi) second quartile ($Q2$); and (vii) third quartile ($Q3$). Thus, given a time-series with N variables, and considering the usage of K meta-features to describe the task properties, the final encoded representation of data can be denoted as

Table 3: Comparison of performance results between HPO and the meta-prediction approach based on different settings for univariate classification datasets (UCR). Each setting represents the number of top architectures (N) and HP configurations K that are used to obtain the results.

Dataset / Configs	$N = All$, $K = 5$			$N = All$, $K = 1$			$N = 1$, $K = 5$			$N = 1$, $K = 1$		
	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)	HEBO	Meta	Change (%)
Adiac	75.2 ± 3.5	70.6 ± 4.4	6.2 ↓	74.6 ± 6.5	75.5 ± 5.6	1.3 ↑	80.1 ± 0.3	73.0 ± 6.0	8.8 ↓	80.3	82.5 ± 2.2	2.7 ↑
BME	99.9 ± 0.3	98.8 ± 1.1	1.1 ↓	100.0 ± 0.0	99.2 ± 0.7	0.8 ↓	100.0 ± 0.0	99.4 ± 0.5	0.6 ↓	100.0	99.5 ± 0.4	0.5 ↓
Beef	85.6 ± 2.1	78.9 ± 7.2	7.8 ↓	87.2 ± 2.5	82.3 ± 5.1	5.6 ↓	83.3 ± 0.0	85.1 ± 2.7	2.2 ↑	83.3	86.3 ± 2.0	3.6 ↑
CBF	98.5 ± 0.9	96.7 ± 1.6	1.9 ↓	98.9 ± 0.7	97.4 ± 0.9	1.6 ↓	99.9 ± 0.1	98.6 ± 1.0	1.4 ↓	100.0	98.6 ± 1.2	1.4 ↓
Car	88.5 ± 2.9	84.0 ± 7.2	5.1 ↓	88.3 ± 3.5	86.6 ± 5.8	2.0 ↓	84.7 ± 0.8	86.1 ± 2.2	1.7 ↑	85.0	88.0 ± 8.3	3.6 ↑
Coffee	100.0 ± 0.0	99.3 ± 1.7	0.7 ↓	100.0 ± 0.0	100.0 ± 0.0	0.0 ↕	100.0 ± 0.0	100.0 ± 0.0	0.0 ↕	100.0	100.0 ± 0.0	0.0 ↕
CricketX	67.1 ± 7.0	69.6 ± 6.9	3.7 ↑	67.4 ± 7.1	71.1 ± 6.9	5.4 ↑	80.4 ± 0.9	82.1 ± 2.5	2.2 ↑	81.5	83.8 ± 1.4	2.8 ↑
CricketY	68.2 ± 5.7	69.4 ± 5.7	1.8 ↑	68.6 ± 5.7	70.9 ± 6.4	3.2 ↑	79.0 ± 0.6	79.7 ± 2.4	0.8 ↑	80.0	82.4 ± 1.8	3.0 ↑
Crop	76.8 ± 0.5	80.4 ± 2.1	4.6 ↑	77.1 ± 0.6	81.2 ± 1.8	5.4 ↑	76.6 ± 0.1	80.8 ± 1.6	5.5 ↑	76.6	82.1 ± 0.3	7.2 ↑
DiatomSizeReduction	98.2 ± 0.9	94.7 ± 2.9	3.6 ↓	97.9 ± 1.6	96.8 ± 1.9	1.2 ↓	99.7 ± 0.0	97.1 ± 0.7	2.6 ↓	99.7	97.8 ± 0.1	1.9 ↓
DodgerLoopWeekend	98.6 ± 0.1	98.9 ± 0.3	0.2 ↑	98.7 ± 0.3	99.0 ± 0.0	0.3 ↑	98.6 ± 0.0	98.9 ± 0.2	0.3 ↑	98.6	99.0 ± 0.0	0.4 ↑
ECG200	92.4 ± 0.9	92.1 ± 2.1	0.3 ↓	92.5 ± 1.4	93.8 ± 1.1	1.4 ↑	92.4 ± 0.5	93.2 ± 1.6	0.9 ↑	93.0	95.1 ± 1.6	2.3 ↑
ECG5000	94.4 ± 0.1	96.8 ± 0.4	2.5 ↑	94.5 ± 0.2	97.0 ± 0.2	2.6 ↑	94.3 ± 0.0	97.1 ± 0.1	2.9 ↑	94.4	97.1 ± 0.1	2.9 ↑
EOGVerticalSignal	49.6 ± 3.5	51.4 ± 3.9	3.7 ↑	51.3 ± 4.3	52.9 ± 4.7	3.2 ↑	52.0 ± 0.9	52.9 ± 4.3	1.7 ↑	53.0	54.9 ± 4.5	3.6 ↑
ElectricDevices	71.7 ± 2.3	76.7 ± 2.2	7.0 ↑	72.7 ± 2.6	77.4 ± 1.9	6.5 ↑	75.4 ± 0.3	78.9 ± 1.7	4.7 ↑	75.7	79.5 ± 1.8	5.0 ↑
FaceAll	87.6 ± 3.8	83.4 ± 5.3	4.7 ↓	86.2 ± 5.9	84.8 ± 4.0	1.7 ↓	91.2 ± 0.1	85.0 ± 5.2	6.8 ↓	91.4	86.7 ± 6.6	5.1 ↓
FaceFour	92.5 ± 2.2	88.3 ± 6.5	4.5 ↓	93.2 ± 2.8	92.2 ± 1.7	1.0 ↓	96.1 ± 1.5	90.5 ± 1.3	5.8 ↓	97.7	93.7 ± 2.1	4.1 ↓
Fish	89.3 ± 2.9	89.2 ± 3.3	0.1 ↓	89.1 ± 3.6	90.8 ± 3.3	1.9 ↑	94.9 ± 0.4	93.8 ± 1.8	1.1 ↓	95.4	94.2 ± 1.5	1.3 ↓
FreezerRegularTrain	99.2 ± 0.7	98.8 ± 0.9	0.4 ↓	99.5 ± 0.5	99.4 ± 0.3	0.1 ↓	99.8 ± 0.1	99.2 ± 0.2	0.6 ↓	100.0	99.4 ± 0.1	0.6 ↓
Fungi	96.7 ± 2.2	93.9 ± 2.3	2.9 ↓	97.8 ± 2.5	95.6 ± 2.1	2.2 ↓	96.1 ± 0.7	96.1 ± 1.9	0.0 ↕	97.3	96.8 ± 0.7	0.5 ↓
GestureMidAirD1	67.3 ± 2.5	67.3 ± 2.7	0.0 ↕	68.1 ± 2.4	69.4 ± 2.7	1.9 ↑	70.8 ± 0.5	69.9 ± 2.5	1.3 ↓	71.5	69.9 ± 2.5	2.3 ↓
GesturePebbleZ2	89.9 ± 2.7	87.2 ± 3.4	3.0 ↓	91.6 ± 2.3	89.3 ± 2.8	2.5 ↓	92.3 ± 1.4	87.8 ± 2.4	4.9 ↓	94.3	89.9 ± 1.0	4.6 ↓
GunPointAgeSpan	98.3 ± 1.0	96.9 ± 1.9	1.4 ↓	98.5 ± 0.8	97.7 ± 1.5	0.8 ↓	100.0 ± 0.0	99.4 ± 0.3	0.6 ↓	100.0	99.6 ± 0.2	0.4 ↓
GunPointMaleVersusFemale	100.0 ± 0.0	99.7 ± 0.4	0.3 ↓	100.0 ± 0.0	99.9 ± 0.2	0.1 ↓	100.0 ± 0.0	99.8 ± 0.2	0.2 ↓	100.0	100.0 ± 0.0	0.0 ↕
HouseTwenty	91.6 ± 4.0	91.7 ± 2.7	0.1 ↑	92.3 ± 3.5	92.8 ± 2.6	0.5 ↑	99.2 ± 0.0	95.6 ± 2.2	3.6 ↓	99.2	96.4 ± 2.2	2.8 ↓
InlineSkate	37.8 ± 4.9	34.4 ± 4.8	9.1 ↓	37.9 ± 4.8	36.0 ± 4.9	5.0 ↓	47.1 ± 0.1	39.0 ± 4.7	17.3 ↓	47.3	41.7 ± 4.7	11.8 ↓
InsectEPGSmallTrain	100.0 ± 0.0	100.0 ± 0.0	0.0 ↕	100.0 ± 0.0	100.0 ± 0.0	0.0 ↕	91.2 ± 0.9	89.5 ± 2.9	1.8 ↓	91.8	90.7 ± 2.6	1.2 ↓
Lightning2	82.6 ± 4.9	81.8 ± 4.9	1.0 ↓	83.3 ± 4.9	84.4 ± 3.7	1.3 ↑	86.6 ± 0.6	83.8 ± 3.1	3.2 ↓	87.7	85.4 ± 2.1	2.7 ↓
Lightning7	80.3 ± 4.6	80.3 ± 5.5	0.0 ↕	82.0 ± 5.0	82.6 ± 4.1	0.8 ↑	85.6 ± 5.6	94.5 ± 1.4	10.4 ↑	93.4	97.6 ± 0.4	4.5 ↑
Mallat	94.3 ± 5.1	92.3 ± 5.7	2.2 ↓	96.3 ± 1.6	96.3 ± 1.4	0.1 ↑	81.5 ± 0.2	83.2 ± 1.2	2.2 ↑	81.7	83.6 ± 1.5	2.4 ↑
MedicalImages	75.5 ± 3.4	77.2 ± 4.1	2.3 ↑	75.8 ± 3.2	79.4 ± 3.1	4.8 ↑	97.3 ± 0.1	97.5 ± 0.2	0.2 ↑	97.4	97.7 ± 0.2	0.3 ↑
MelbournePedestrian	96.2 ± 0.9	96.3 ± 1.7	0.1 ↑	96.3 ± 0.9	97.1 ± 0.9	0.9 ↑	84.6 ± 0.6	85.9 ± 1.1	1.5 ↑	85.6	86.4 ± 1.3	0.9 ↑
MiddlePhalanxOutlineCorrect	83.8 ± 1.2	83.5 ± 3.0	0.3 ↓	84.4 ± 1.6	85.7 ± 1.3	1.6 ↑	94.0 ± 0.1	94.0 ± 0.7	0.0 ↕	94.1	94.8 ± 0.5	0.8 ↑
MixedShapesRegularTrain	91.5 ± 2.0	93.0 ± 1.1	1.6 ↑	91.4 ± 2.2	94.0 ± 1.0	2.8 ↑	88.0 ± 0.0	80.2 ± 4.0	8.8 ↓	88.0	82.2 ± 3.5	6.6 ↓
PickupGestureWiimoteZ	77.0 ± 6.2	74.1 ± 4.8	3.8 ↓	77.7 ± 5.9	75.6 ± 4.4	2.7 ↓	100.0 ± 0.0	100.0 ± 0.0	0.0 ↕	100.0	100.0 ± 0.0	0.0 ↕
Plane	99.1 ± 0.5	98.8 ± 0.7	0.3 ↓	99.2 ± 0.4	98.9 ± 0.6	0.2 ↓	98.3 ± 0.0	100.0 ± 0.0	1.7 ↑	98.3	100.0 ± 0.0	1.7 ↑
PowerCons	99.7 ± 0.7	100.0 ± 0.0	0.3 ↑	99.7 ± 0.7	100.0 ± 0.0	0.3 ↑	97.0 ± 0.3	94.1 ± 1.4	3.0 ↓	97.5	96.1 ± 1.2	1.4 ↓
SemgHandGenderCh2	95.6 ± 1.2	92.5 ± 3.3	3.3 ↓	95.9 ± 1.1	94.9 ± 1.7	1.0 ↓	89.9 ± 0.3	92.2 ± 1.1	2.6 ↑	90.3	92.5 ± 1.6	2.5 ↑
ShapesAll	79.3 ± 5.4	80.7 ± 5.6	1.8 ↑	79.1 ± 5.5	82.1 ± 5.9	3.8 ↑	78.9 ± 0.2	79.6 ± 1.4	0.9 ↑	79.2	80.8 ± 1.2	2.1 ↑
SmallKitchenAppliances	74.2 ± 3.3	74.3 ± 4.4	0.1 ↑	74.8 ± 2.7	76.5 ± 4.2	2.3 ↑	97.7 ± 0.1	97.8 ± 0.1	0.1 ↑	97.8	97.8 ± 0.2	0.0 ↕
StarLightCurves	96.7 ± 1.3	96.6 ± 1.3	0.2 ↓	96.5 ± 1.6	97.2 ± 0.6	0.7 ↑	97.0 ± 0.2	97.1 ± 0.4	0.0 ↕	97.3	97.6 ± 0.2	0.3 ↑
Strawberry	96.9 ± 0.2	96.8 ± 0.7	0.1 ↓	97.2 ± 0.3	97.4 ± 0.4	0.2 ↑	94.4 ± 0.2	95.9 ± 0.7	1.6 ↑	94.7	96.6 ± 1.0	2.0 ↑
SwedishLeaf	92.3 ± 1.5	92.7 ± 3.2	0.4 ↑	93.1 ± 1.4	94.2 ± 1.7	1.2 ↑	77.9 ± 0.3	82.6 ± 0.3	6.1 ↑	78.3	82.9 ± 0.1	5.8 ↑
UWaveGestureLibraryZ	72.2 ± 3.7	76.6 ± 3.3	6.1 ↑	72.1 ± 3.9	77.8 ± 2.7	7.9 ↑	99.9 ± 0.0	99.7 ± 0.1	0.2 ↓	99.9	99.8 ± 0.1	0.1 ↓
Wafer	99.8 ± 0.1	99.6 ± 0.1	0.2 ↓	99.8 ± 0.1	99.7 ± 0.1	0.1 ↓	99.9 ± 0.0	99.7 ± 0.1	0.2 ↓	99.9	100.0 ± 0.0	0.1 ↑
Avg	86.7 ± 14.1	86.1 ± 13.8	0.7 ↓	87.1 ± 13.9	87.6 ± 13.4	0.6 ↑	89.2 ± 12.1	88.8 ± 12.5	0.4 ↓	89.7 ± 12.0	89.9 ± 11.9	0.2 ↑

Table 4: Full results for the classification task. The results are based on top-5 records obtained during hyperparameter optimisation. "/" indicates the out-of-memory situation. "-" indicates Cholesky decomposition error encountered during HPO experiments using Ray framework.

Benchmark	Datasets / Models	Transformer-based								RNN-based				MLP-based			CNN-based			
		Autoformer (2021)	Crossformer (2023)	ETSformer (2022)	Informer (2021)	PatchTST (2023)	Reformer (2020)	Transformer (2017)	iTransformer (2024)	FiLM (2022)	LMU (2019)	SegRNN (2023)	S4 (2023)	DLinear (2023)	LightTS (2022)	TSMixer (2023)	ModernTCN (2024)	TS2Vec (2022)	TimesNet (Inception) (2023)	TimesNet (ResNeXt) (2023)
UEA	ArticularyWordRecognition	79.7 ± 1.0	98.3 ± 0.3	98.5 ± 0.2	98.3 ± 0.0	98.5 ± 0.2	98.4 ± 0.2	98.4 ± 0.2	98.8 ± 0.1	96.2 ± 0.3	98.1 ± 0.1	95.3 ± 0.4	97.9 ± 0.2	98.0 ± 0.0	97.9 ± 0.1	98.3 ± 0.0	98.2 ± 0.2	99.4 ± 0.2	98.7 ± 0.0	98.8 ± 0.1
	AtrialFibrillation	60.0 ± 0.0	46.7 ± 0.0	65.4 ± 3.0	66.7 ± 0.0	46.7 ± 0.0	73.3 ± 0.0	73.3 ± 0.0	68.0 ± 3.0	60.0 ± 0.0	92.0 ± 3.0	66.7 ± 0.0	66.7 ± 0.0	46.7 ± 0.0	60.0 ± 0.0	66.7 ± 0.0	56.0 ± 3.7	40.0 ± 0.0	73.3 ± 0.0	46.7 ± 0.0
	BasicMotions	84.0 ± 2.2	98.0 ± 1.1	100.0 ± 0.0	100.0 ± 0.0	85.5 ± 1.1	100.0 ± 0.0	100.0 ± 0.0	96.0 ± 1.4	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	98.0 ± 1.1	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	CharacterTrajectories	93.3 ± 0.4	98.6 ± 0.1	98.2 ± 0.1	98.5 ± 0.0	97.9 ± 0.1	-	98.6 ± 0.1	98.7 ± 0.1	98.2 ± 0.2	98.4 ± 0.1	97.8 ± 0.2	98.7 ± 0.2	98.0 ± 0.0	98.7 ± 0.0	98.7 ± 0.0	98.7 ± 0.1	99.5 ± 0.1	98.5 ± 0.2	98.5 ± 0.1
	Cricket	43.6 ± 0.7	95.0 ± 0.8	97.5 ± 0.6	96.9 ± 0.6	96.6 ± 0.8	97.2 ± 0.0	96.9 ± 0.6	93.3 ± 1.8	95.8 ± 0.0	97.2 ± 0.0	95.5 ± 1.2	95.8 ± 0.0	98.3 ± 0.6	92.0 ± 0.6	95.2 ± 0.8	96.1 ± 0.6	100.0 ± 0.0	97.2 ± 0.0	96.4 ± 0.8
	DuckDuckGeese	36.0 ± 0.0	56.8 ± 3.9	40.8 ± 1.1	61.2 ± 1.1	26.0 ± 0.0	61.2 ± 1.1	64.0 ± 0.0	60.0 ± 1.4	58.0 ± 0.0	60.4 ± 1.7	60.0 ± 0.0	64.0 ± 0.0	68.0 ± 0.0	62.0 ± 0.0	58.0 ± 0.0	58.0 ± 2.0	64.0 ± 0.0	64.4 ± 0.9	58.4 ± 0.9
	ERing	90.8 ± 0.2	95.6 ± 0.4	96.2 ± 0.4	95.2 ± 0.0	96.1 ± 0.2	95.5 ± 0.3	95.8 ± 0.2	94.9 ± 0.6	88.4 ± 0.4	95.5 ± 0.3	89.8 ± 0.2	95.7 ± 0.1	95.9 ± 0.0	95.9 ± 0.0	94.6 ± 0.2	96.2 ± 0.5	93.7 ± 0.0	95.6 ± 0.3	96.1 ± 0.3
	EigenWorms	60.1 ± 1.5	52.1 ± 0.4	62.6 ± 0.6	64.7 ± 0.6	53.6 ± 0.6	66.2 ± 1.5	/	55.7 ± 1.4	46.7 ± 0.3	67.2 ± 1.6	44.1 ± 0.4	63.1 ± 0.7	50.2 ± 0.4	55.7 ± 0.0	57.9 ± 0.8	65.8 ± 1.0	92.7 ± 0.7	68.7 ± 0.0	67.8 ± 0.3
	Epilepsy	88.0 ± 3.8	90.7 ± 2.3	93.5 ± 0.5	91.5 ± 0.9	97.2 ± 0.3	90.6 ± 0.9	92.2 ± 0.4	81.8 ± 0.6	82.5 ± 0.3	92.9 ± 0.6	97.2 ± 0.6	93.9 ± 0.4	67.4 ± 0.0	97.1 ± 0.0	86.8 ± 1.0	97.0 ± 0.6	98.1 ± 0.4	91.6 ± 0.4	95.7 ± 0.9
	EthanolConcentration	33.2 ± 0.9	33.8 ± 0.4	30.4 ± 0.6	33.5 ± 0.5	30.7 ± 0.4	32.4 ± 0.5	31.1 ± 0.2	30.6 ± 0.5	30.1 ± 0.3	30.7 ± 0.8	30.5 ± 0.2	30.9 ± 0.4	31.4 ± 0.2	30.2 ± 0.4	31.4 ± 0.2	30.3 ± 0.4	31.0 ± 0.5	32.7 ± 0.7	31.8 ± 0.4
	FaceDetection	67.7 ± 0.3	70.1 ± 0.1	70.1 ± 0.1	70.6 ± 0.1	69.4 ± 0.3	70.6 ± 0.4	70.5 ± 0.1	69.7 ± 0.1	70.1 ± 0.1	70.0 ± 0.3	68.8 ± 0.2	70.7 ± 0.2	70.2 ± 0.3	70.2 ± 0.2	68.8 ± 0.2	69.4 ± 0.3	54.6 ± 0.3	70.7 ± 0.1	70.1 ± 0.4
	FingerMovements	64.8 ± 0.4	59.8 ± 0.8	63.6 ± 1.1	64.8 ± 0.8	62.6 ± 0.5	64.4 ± 0.5	62.6 ± 1.3	62.8 ± 0.4	61.6 ± 0.5	64.0 ± 0.7	64.8 ± 0.8	61.0 ± 0.7	64.6 ± 0.9	62.0 ± 0.0	64.4 ± 1.1	63.8 ± 1.3	58.6 ± 0.9	64.0 ± 0.0	64.0 ± 0.0
	HandMovementDirection	42.4 ± 2.0	61.9 ± 1.1	65.4 ± 0.7	66.5 ± 0.6	62.2 ± 0.0	67.0 ± 0.8	68.4 ± 0.7	58.1 ± 0.0	71.1 ± 0.7	67.9 ± 0.6	47.8 ± 0.7	66.0 ± 1.1	68.6 ± 0.6	65.9 ± 0.6	68.6 ± 0.6	60.8 ± 0.0	39.7 ± 1.2	66.2 ± 1.4	65.9 ± 0.6
	Handwriting	36.4 ± 0.4	28.3 ± 1.5	34.3 ± 0.6	34.8 ± 0.6	28.6 ± 0.8	33.0 ± 0.5	37.0 ± 0.4	27.9 ± 0.2	32.9 ± 0.2	37.8 ± 0.4	21.8 ± 0.2	31.2 ± 0.5	24.3 ± 0.3	24.0 ± 0.4	24.5 ± 0.5	29.2 ± 0.3	58.8 ± 0.6	32.4 ± 0.3	34.4 ± 0.3
	Heartbeat	74.4 ± 0.3	77.4 ± 0.7	77.5 ± 0.4	79.4 ± 0.4	73.3 ± 0.2	78.7 ± 0.4	79.8 ± 0.6	77.9 ± 0.4	75.6 ± 0.0	77.7 ± 0.4	73.8 ± 0.2	77.7 ± 0.2	76.8 ± 0.3	77.8 ± 0.2	77.8 ± 0.2	76.7 ± 0.4	77.1 ± 1.3	78.0 ± 0.0	78.5 ± 0.4
	JapaneseVowels	97.8 ± 0.0	97.5 ± 0.2	97.9 ± 0.2	97.9 ± 0.3	95.8 ± 0.1	97.7 ± 0.2	98.0 ± 0.2	97.9 ± 0.2	95.2 ± 0.1	97.5 ± 0.2	96.4 ± 0.3	97.5 ± 0.2	97.9 ± 0.2	97.9 ± 0.2	98.4 ± 0.0	98.0 ± 0.1	99.5 ± 0.0	97.8 ± 0.0	98.1 ± 0.0
	LSST	51.7 ± 0.5	41.5 ± 1.2	39.9 ± 0.4	40.4 ± 0.1	56.8 ± 0.5	40.4 ± 0.3	44.2 ± 1.5	39.2 ± 0.6	39.6 ± 0.1	41.9 ± 0.5	51.9 ± 0.6	40.1 ± 0.7	34.8 ± 0.1	41.1 ± 0.0	39.0 ± 0.4	36.5 ± 0.4	61.5 ± 0.1	40.8 ± 0.7	39.3 ± 0.3
	Libras	85.7 ± 1.0	87.7 ± 0.8	89.4 ± 0.4	86.2 ± 0.5	82.2 ± 0.7	86.8 ± 0.6	86.8 ± 0.2	89.5 ± 0.3	87.2 ± 0.0	83.1 ± 0.5	91.9 ± 0.3	85.1 ± 0.3	78.9 ± 0.0	87.4 ± 1.2	86.7 ± 0.8	88.0 ± 0.5	88.4 ± 0.6	80.8 ± 1.9	83.2 ± 0.5
	MotorImagery	62.2 ± 0.4	64.8 ± 1.8	62.4 ± 0.5	67.4 ± 0.9	63.8 ± 1.3	65.4 ± 1.1	68.4 ± 1.5	67.0 ± 1.0	65.4 ± 1.5	65.0 ± 1.7	68.8 ± 0.4	65.0 ± 1.0	70.6 ± 0.5	65.0 ± 0.0	66.6 ± 0.5	65.6 ± 1.1	60.0 ± 0.0	65.4 ± 1.1	64.6 ± 1.3
	NATOPS	90.1 ± 1.7	88.3 ± 0.8	92.6 ± 0.6	94.0 ± 0.6	81.3 ± 0.9	95.0 ± 0.4	96.1 ± 1.3	87.4 ± 1.0	93.9 ± 0.7	92.8 ± 0.0	95.1 ± 0.6	93.3 ± 0.0	96.7 ± 0.0	96.2 ± 0.3	96.1 ± 1.0	93.7 ± 0.6	97.2 ± 0.0	94.9 ± 1.1	96.7 ± 0.4
	PEMS-SF	85.5 ± 0.7	87.1 ± 1.2	86.7 ± 0.9	87.2 ± 1.3	86.9 ± 0.3	86.6 ± 0.5	88.9 ± 0.5	91.3 ± 0.0	85.5 ± 1.4	87.9 ± 0.7	85.1 ± 0.9	89.6 ± 0.0	85.1 ± 0.2	87.6 ± 0.5	89.0 ± 0.7	87.4 ± 0.6	88.8 ± 1.6	86.7 ± 0.6	87.9 ± 0.9
	PenDigits	98.4 ± 0.2	97.8 ± 0.2	97.9 ± 0.1	98.6 ± 0.1	97.4 ± 0.1	98.5 ± 0.1	98.7 ± 0.1	98.2 ± 0.1	95.6 ± 0.0	98.1 ± 0.1	97.1 ± 0.2	98.5 ± 0.0	93.0 ± 0.0	96.7 ± 0.1	97.2 ± 0.1	97.8 ± 0.2	99.0 ± 0.0	98.6 ± 0.1	98.6 ± 0.1
	PhonemeSpectra	12.8 ± 0.1	13.5 ± 0.3	14.7 ± 0.4	13.8 ± 0.1	14.6 ± 0.4	13.7 ± 0.2	16.0 ± 0.0	11.4 ± 0.1	9.3 ± 0.1	13.4 ± 0.1	20.8 ± 0.5	14.6 ± 0.2	8.2 ± 0.0	17.0 ± 0.2	10.9 ± 0.1	14.2 ± 0.2	26.9 ± 0.2	15.0 ± 0.2	19.4 ± 0.4
	RacketSports	85.9 ± 0.7	84.7 ± 0.6	87.0 ± 1.4	89.5 ± 0.5	83.4 ± 1.3	90.8 ± 0.8	90.1 ± 0.5	79.4 ± 0.9	78.8 ± 0.5	86.8 ± 0.5	79.3 ± 0.8	85.5 ± 0.7	82.2 ± 0.0	81.3 ± 0.6	82.9 ± 0.0	83.2 ± 1.1	91.3 ± 0.9	90.4 ± 0.4	91.3 ± 0.7
	SelfRegulationSCP1	66.1 ± 0.8	92.2 ± 0.4	90.4 ± 0.4	92.0 ± 0.2	89.4 ± 0.6	91.8 ± 0.4	91.5 ± 0.0	93.4 ± 0.3	93.2 ± 0.0	92.6 ± 0.5	85.4 ± 0.6	93.3 ± 0.3	93.4 ± 0.1	93.0 ± 0.3	93.0 ± 0.2	92.6 ± 0.1	86.5 ± 0.2	92.5 ± 0.3	91.9 ± 0.8
	SelfRegulationSCP2	61.7 ± 0.4	57.1 ± 0.2	58.7 ± 0.3	58.8 ± 0.3	56.9 ± 0.3	58.9 ± 0.4	59.3 ± 1.2	58.7 ± 0.3	60.6 ± 0.7	59.0 ± 0.7	59.1 ± 0.5	60.3 ± 0.5	58.1 ± 0.5	59.0 ± 0.7	59.2 ± 1.5	58.1 ± 0.3	59.7 ± 0.9	58.2 ± 0.5	58.1 ± 0.3
	SpokenArabicDigits	98.9 ± 0.1	98.5 ± 0.1	99.1 ± 0.0	98.9 ± 0.2	98.5 ± 0.1	98.9 ± 0.1	99.1 ± 0.1	99.1 ± 0.1	98.2 ± 0.1	98.9 ± 0.3	98.1 ± 0.2	98.9 ± 0.2	97.6 ± 0.1	98.1 ± 0.3	98.7 ± 0.1	98.6 ± 0.1	99.3 ± 0.1	99.0 ± 0.0	99.3 ± 0.1
	StandWalkJump	64.0 ± 3.7	46.7 ± 0.0	66.7 ± 0.0	69.3 ± 3.6	60.0 ± 0.0	69.3 ± 3.6	66.7 ± 0.0	66.7 ± 0.0	60.0 ± 0.0	78.7 ± 3.0	53.3 ± 0.0	60.0 ± 0.0	66.7 ± 0.0	66.7 ± 0.0	66.7 ± 0.0	68.0 ± 3.0	66.7 ± 0.0	73.3 ± 0.0	53.3 ± 0.0
	UWaveGestureLibrary	59.4 ± 1.2	85.7 ± 0.3	87.7 ± 0.2	87.5 ± 0.4	86.9 ± 0.5	87.4 ± 0.3	87.8 ± 0.2	88.8 ± 0.3	83.5 ± 1.1	86.4 ± 0.9	82.2 ± 1.4	87.4 ± 0.5	84.4 ± 0.3	87.9 ± 0.6	87.6 ± 0.3	87.6 ± 0.2	92.0 ± 0.4	87.8 ± 0.3	87.9 ± 0.3
	Avg	68.1 ± 1.3	72.6 ± 1.1	74.7 ± 0.8	76.0 ± 0.9	71.7 ± 0.6	75.3 ± 0.9	77.1 ± 0.6	73.9 ± 0.9	72.9 ± 0.5	77.0 ± 1.0	73.0 ± 0.6	75.3 ± 0.4	72.6 ± 1.1	74.6 ± 0.4	74.5 ± 0.6	74.7 ± 1.1	76.7 ± 0.6	76.3 ± 0.6	74.9 ± 0.5
	Rank	19	16	9	5	18	6	1	13	15	2	14	6	16	11	12	9	3	4	8
UCR	Adiac	74.7 ± 0.6	51.2 ± 1.1	74.6 ± 1.9	63.1 ± 1.1	75.8 ± 0.9	43.0 ± 3.0	56.3 ± 3.5	52.9 ± 0.7	73.5 ± 1.0	70.5 ± 1.2	48.3 ± 3.4	77.3 ± 0.7	71.8 ± 0.6	67.1 ± 3.8	66.4 ± 1.2	78.5 ± 0.3	80.1 ± 0.3	47.3 ± 1.0	73.7 ± 1.7
	BME	96.4 ± 2.4	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	99.5 ± 0.6	100.0 ± 0.0	99.9 ± 0.3	99.4 ± 0.3	98.8 ± 0.3	99.4 ± 0.3	99.9 ± 0.3	100.0 ± 0.0	100.0 ± 0.0	98.8 ± 0.3	99.4 ± 0.3	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	Beef	100.0 ± 0.0	76.7 ± 2.4	87.4 ± 1.5	85.3 ± 1.9	84.0 ± 1.5	80.7 ± 1.5	86.0 ± 1.5	76.0 ± 1.5	86.7 ± 0.0	86.0 ± 1.5	40.0 ± 0.0	87.4 ± 1.5	81.3 ± 3.0	80.0 ± 0.0	82.0 ± 1.8	84.7 ± 1.9	83.3 ± 0.0	86.0 ± 1.5	84.0 ± 1.5
	CBF	83.1 ± 1.8	97.6 ± 0.6	97.9 ± 0.3	98.7 ± 0.5	98.1 ± 0.9	99.3 ± 0.3	99.0 ± 0.2	96.2 ± 0.3	98.1 ± 0.4	98.8 ± 0.3	93.8 ± 1.0	98.2 ± 0.1	93.1 ± 1.0	94.9 ± 0.6	97.3 ± 0.6	96.0 ± 0.6	99.9 ± 0.1	97.3 ± 0.2	97.6 ± 0.5
	Car	42.7 ± 0.9	77.7 ± 0.9	91.7 ± 0.0	81.7 ± 2.0	80.7 ± 0.9	79.7 ± 0.8	81.0 ± 1.5	81.3 ± 1.4	83.0 ± 0.7	87.7 ± 2.2	39.3 ± 0.9	91.0 ± 0.9	85.0 ± 0.0	83.3 ± 0.0	87.0 ± 1.8	89.3 ± 0.9	84.7 ± 0.8	85.3 ± 1.9	87.3 ± 0.9
	Coffee	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	98.6 ± 2.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	53.6 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	CricketX	45.3 ± 1.9	63.2 ± 1.0	62.5 ± 0.4	63.0 ± 0.2	69.0 ± 0.4	59.8 ± 0.2	63.4 ± 0.2	50.9 ± 0.5	60.1 ± 0.4	64.4 ± 1.7	60.2 ± 1.6	66.3 ± 0.1	34.5 ± 0.1	61.9 ± 0.5	57.9 ± 0.5	57.6 ± 1.0	80.4 ± 0.9	59.4 ± 0.5	62.0 ± 0.8
	CricketY	31.0 ± 0.3	65.0 ± 0.2	63.4 ± 1.0	64.7 ± 0.4	70.4 ± 0.8	61.9 ± 1.0	64.8 ± 0.4	56.7 ± 0.3	61.4 ± 1.1	65.5 ± 1.1	65.4 ± 1.4	66.2 ± 0.0	43.7 ± 0.2	63.9 ± 0.7	58.7 ± 0.5	61.0 ± 0.2	79.0 ± 0.6	62.7 ± 0.9	66.8 ± 0.6
	Crop	74.8 ± 0.1	76.3 ± 0.1	76.0 ± 0.2	77.5 ± 0.1	74.7 ± 0.2	76.8 ± 0.1	77.4 ± 0.0	77.2 ± 0.0	73.3 ± 0.3	77.0 ± 0.2	75.0 ± 0.2	77.5 ± 0.2	68.4 ± 0.1	75.2 ± 0.3	75.5 ± 0.2	76.4 ± 0.2	76.6 ± 0.1	77.2 ± 0.1	77.1 ± 0.2
	DiatomSizeReduction	91.5 ± 0.8	87.5 ± 1.8	98.5 ± 0.4	96.7 ± 0.4	96.8 ± 0.3	92.2 ± 2.0	93.8 ± 0.9	93.0 ± 3.1	96.7 ± 0.4	97.2 ± 0.2	79.5 ± 3.3	97.8 ± 0.3	92.3 ± 1.7	96.7 ± 0.4	96.9 ± 0.5	98.3 ± 0.6	99.7 ± 0.0	97.6 ± 0.2	97.8 ± 0.1
	DodgerLoopWeekend	89.9 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.9 ± 0.4	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.7 ± 0.3	97.8 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0	98.6 ± 0.0
	ECG200	91.4 ± 0.5	89.4 ± 0.5	93.2 ± 0.8	91.0 ± 0.7	91.0 ± 1.0	89.2 ± 0.4	89.4 ± 0.5	90.8 ± 1.1	92.0 ± 0.0	92.2 ± 0.4	90.4 ± 0.9	92.8 ± 0.4	86.0 ± 0.7	90.4 ± 0.9	92.2 ± 0.4	92.2 ± 0.4	92.4 ± 0.5	91.0 ± 0.0	91.2 ± 0.4
	ECG5000	94.5 ± 0.1	94.7 ± 0.1	94.4 ± 0.1	94.6 ± 0.1	94.7 ± 0.1	-	94.6 ± 0.0	94.3 ± 0.1	94.3 ± 0.1	94.4 ± 0.0	94.2 ± 0.1	94.5 ± 0.2	94.2 ± 0.0	94.2 ± 0.1	94.3 ± 0.1	94.6 ± 0.1	94.3 ± 0.0	94.5 ± 0.0	94.5 ± 0.0
	EOGVerticalSignal	23.3 ± 1.2	47.5 ± 2.2	49.9 ± 0.3	49.7 ± 1.1	53.8 ± 1.1	45.4 ± 1.0	55.6 ± 0.8	43.9 ± 1.5	51.5 ± 0.9	51.5 ± 0.7	36.7 ± 0.8	51.6 ± 0.3	37.8 ± 0.4	49.3 ± 0.8	47.0 ± 1.0	50.2 ± 0.3	52.0 ± 0.9	43.1 ± 0.3	42.9 ± 0.6
	ElectricDevices	70.4 ± 0.4	66.8 ± 0.7	69.7 ± 0.6	74.5 ± 0.2	72.3 ± 0.4	74.9 ± 0.5	75.4 ± 0.2	62.2 ± 0.2	62.6 ± 0.3	71.6 ± 0.1	74.4 ± 0.1	69.3 ± 0.3	48.7 ± 0.3	69.1 ± 0.6	65.2 ± 0.4	66.6 ± 0.3	75.4 ± 0.3	71.2 ± 0.5	72.5 ± 0.4
	FaceAll	85.8 ± 0.6	86.3 ± 0.8	85.4 ± 0.8	76.6 ± 0.3	89.3 ± 0.1	-	76.1 ± 0.3	87.6 ± 0.2	79.0 ± 1.3	89.0 ± 0.6	79.6 ± 0.4	90.8 ± 0.4	83.5 ± 0.3	87.6 ± 1.4	86.6 ± 0.7	90.0 ± 0.6	91.2 ± 0.1	77.3 ± 0.4	81.3 ± 0.5
	FaceFour	87.1 ± 0.6	91.8 ± 1.5	91.6 ± 1.3	90.2 ± 0.6	90.5 ± 0.6	90.2 ± 1.0	89.8 ± 0.0	88.4 ± 1.0	95.7 ± 0.5	91.6 ± 1.0	67.5 ± 0.7	90.9 ± 0.0	87.3 ± 0.5	86.8 ± 1.0	89.5 ± 1.0	89.6 ± 0.5	96.1 ± 1.5	90.9 ± 0.0	92.3 ± 1.5
	Fish	82.8 ± 0.3	83.9 ± 1.7	89.0 ± 0.5	85.6 ± 0.9	85.8 ± 2.6	85.7 ± 1.3	84.1 ± 0.5	86.2 ± 0.7	87.2 ± 0.5	87.3 ± 0.2	35.5 ± 1.9	88.0 ± 0.7	87.0 ± 0.5	85.5 ± 1.2	86.0 ± 1.3	90.2 ± 0.5	94.9 ± 0.4	86.7 ± 0.3	87.5 ± 0.6
	FreezerRegularTrain	98.1 ± 0.7	99.7 ± 0.2	98.1 ± 0.2	99.5 ± 0.1	99.8 ± 0.1	99.6 ± 0.1	99.8 ± 0.0	94.4 ± 0.2	98.7 ± 0.3	99.6 ± 0.0	78.0 ± 0.2	99.7 ± 0.1	94.7 ± 0.0	99.1 ± 0.1	99.2 ± 0.2	99.5 ± 0.3	99.8 ± 0.1	98.7 ± 0.2	98.9 ± 0.2
	Fungi	88.3 ± 0.9	97.3 ± 0.9	97.9 ± 0.5	98.5 ± 0.5	97.7 ± 0.9	98.7 ± 0.3	98.7 ± 0.3	87.0 ± 0.4	87.3 ± 0.3	97.8 ± 1.2	88.7 ± 0.5	93.0 ± 0.0	93.0 ± 0.0	91.0 ± 0.2	93.0 ± 0.4	92.1 ± 0.6	96.1 ± 0.7	98.8 ± 0.5	98.7 ± 0.6
	GestureMidAirD1	57.9 ± 0.4	65.9 ± 0.9	64.1 ± 0.7	67.7 ± 1.3	61.7 ± 1.2	67.7 ± 0.9	68.0 ± 0.9	64.4 ± 0.4	64.6 ± 0.0	67.9 ± 0.9	63.2 ± 0.3	68.5 ± 0.0	63.8 ± 0.0	66.9 ± 0.0	66.9 ± 1.3	70.0 ± 0.0	70.8 ± 0.5	68.0 ± 0.4	65.1 ± 0.4
	GesturePebbleZ2	37.5 ± 1.1	91.0 ± 0.7	91.9 ± 0.3	91.6 ± 0.7	85.4 ± 1.0	92.5 ± 1.2	92.7 ± 0.6	87.0 ± 1.0	91.8 ± 0.0	89.5 ± 0.4	86.2 ± 1.5	90.1 ± 0.3	83.5 ± 0.0	83.8 ± 2.0	92.7 ± 1.0	86.3 ± 0.6	92.3 ± 1.4	88.7 ± 0.7	85.5 ± 2.3
	GunPointAgeSpan	90.5 ± 1.0	91.6 ± 0.9	97.7 ± 0.2	97.7 ± 0.4	97.0 ± 0.7	96.7 ± 0.3	97.7 ± 0.5	90.5 ± 0.9	91.3 ± 0.2	98.5 ± 0.3	97.3 ± 0.1	97.4 ± 0.3	89.5 ± 0.3	96.8 ± 0.7	90.9 ± 0.5	92.1 ± 0.6	100.0 ± 0.0	97.5 ± 0.5	97.7 ± 0.3
	GunPointMaleVersusFemale	96.7 ± 0.2	99.8 ± 0.3	100.0 ± 0.0	100.0 ± 0.0	99.5 ± 0.2	100.0 ± 0.0	100.0 ± 0.0	99.7 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	99.1 ± 0.4	99.8 ± 0.2	99.9 ± 0.2	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	HouseTwenty	68.2 ± 1.1	88.6 ± 0.8	88.7 ± 1.2	90.8 ± 0.6	89.6 ± 0.4	90.1 ± 0.4	91.9 ± 0.4	79.2 ± 0.4	84.4 ± 0.5	89.3 ± 0.4	63.7 ± 0.4	90.3 ± 0.5	79.2 ± 0.4	80.5 ± 0.9	81.0 ± 0.8	81.5 ± 0.0	99.2 ± 0.0	89.9 ± 0.0	90.4 ± 0.5
	InlineSkate	23.5 ± 0.5	32.4 ± 0.5	35.6 ± 1.4	32.9 ± 0.3	32.5 ± 0.6	32.2 ± 0.8	33.5 ± 0.5	36.2 ± 0.2	32.3 ± 0.2	35.8 ± 0.5	32.1 ± 0.7	36.9 ± 0.4	27.7 ± 0.5	33.7 ± 0.5	36.3 ± 0.5	34.7 ± 0.4	47.1 ± 0.1	33.4 ± 0.4	33.7 ± 0.4
	InsectEPGSmallTrain	64.4 ± 0.3	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	72.5 ± 0.7	100.0 ± 0.0	100.0 ± 0.0	83.1 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	83.1 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	-	100.0 ± 0.0	100.0 ± 0.0
	Lightning2	67.6 ± 1.4	80.3 ± 2.0	81.0 ± 0.9	79.7 ± 0.9	78.7 ± 0.0	79.3 ± 0.9	82.6 ± 0.9	80.3 ± 0.0	80.3 ± 1.7	84.2 ± 0.9	80.0 ± 1.4	78.4 ± 0.8	73.8 ± 0.0	75.4 ± 0.0	78.7 ± 0.0	76.4 ± 0.9	91.2 ± 0.9	80.3 ± 0.0	78.4 ± 0.8
	Lightning7	46.0 ± 0.8	76.7 ± 1.0	77.8 ± 0.6	83.9 ± 0.6	78.4 ± 0.6	82.5 ± 0.6	85.5 ± 0.8	68.8 ± 1.2	74.0 ± 0.0	78.9 ± 0.8	76.1 ± 0.8	74.3 ± 0.6	71.2 ± 0.0	74.0 ± 0.0	72.9 ± 0.6	74.5 ± 1.5	86.6 ± 0.6	82.8 ± 0.8	84.1 ± 0.7
	Mallat	90.7 ± 0.4	95.7 ± 0.5	96.0 ± 0.2	96.4 ± 0.3	96.3 ± 0.3	95.5 ± 0.3	96.2 ± 0.0	97.3 ± 0.4	96.9 ± 0.2	97.0 ± 0.6	80.0 ± 2.8	97.1 ± 0.1	97.8 ± 0.0	97.6 ± 0.2	97.4 ± 0.3	97.2 ± 0.2	85.6 ± 5.6	96.7 ± 0.2	96.1 ± 0.3
	MedicalImages	68.5 ± 1.4	73.5 ± 0.3	72.2 ± 0.2	74.5 ± 0.6	74.8 ± 0.4	71.3 ± 0.0	75.2 ± 0.2	73.7 ± 0.3	70.3 ± 0.6	74.0 ± 0.5	77.3 ± 0.2	76.5 ± 0.4	59.5 ± 0.3	73.1 ± 0.7	75.2 ± 0.4	74.8 ± 0.5	81.5 ± 0.2	73.6 ± 0.3	73.5 ± 0.4
	MelbournePedestrian	94.8 ± 0.3	90.7 ± 0.4	94.9 ± 0.0	96.3 ± 0.2	89.0 ± 0.3	96.3 ± 0.2	96.3 ± 0.1	94.9 ± 0.2	93.9 ± 0.1	95.7 ± 0.1	94.2 ± 0.3	95.7 ± 0.1	90.1 ± 0.8	93.0 ± 0.1	95.0 ± 0.0	93.9 ± 0.1	97.3 ± 0.1	96.3 ± 0.1	97.1 ± 0.0
	MiddlePhalanxOutlineCorrect	84.9 ± 0.7	57.5 ± 0.2	84.7 ± 0.8	81.2 ± 0.1	79.2 ± 1.3	80.2 ± 0.8	80.6 ± 1.1	59.3 ± 0.2	84.0 ± 0.3	84.0 ± 1.0	80.3 ± 1.7	83.1 ± 0.9	68.4 ± 1.0	61.0 ± 2.3	59.8 ± 0.5	84.7 ± 0.9	84.6 ± 0.6	80.5 ± 0.3	82.4 ± 0.4
	MixedShapesRegularTrain	85.7 ± 0.8	88.7 ± 0.6	93.7 ± 0.4	89.5 ± 0.4	89.1 ± 0.3	88.1 ± 0.2	89.2 ± 0.1	89.5 ± 0.5	89.1 ± 0.1	89.8 ± 0.2	90.8 ± 0.5	90.7 ± 0.2	83.5 ± 0.2	90.9 ± 0.0	91.0 ± 0.2	94.0 ± 0.1	94.0 ± 0.1	89.0 ± 0.1	89.3 ± 0.2
	PickupGestureWiimoteZ	42.4 ± 1.7	67.2 ± 1.8	69.6 ± 0.9	72.0 ± 0.0	67.6 ± 0.9	68.8 ± 1.8	74.0 ± 2.0	74.0 ± 1.4	76.0 ± 0.0	75.6 ± 2.2	53.6 ± 3.8	75.6 ± 0.9	70.0 ± 0.0	73.6 ± 0.9	76.0 ± 0.0	70.8 ± 1.1	88.0 ± 0.0	68.4 ± 0.9	76.4 ± 1.7
	PigAirwayPressure	6.3 ± 0.2	11.5 ± 0.5	13.6 ± 0.5	11.3 ± 0.2	7.5 ± 0.3	11.0 ± 0.4	13.4 ± 0.6	11.7 ± 0.5	12.8 ± 0.8	10.9 ± 0.3	7.6 ± 0.2	11.9 ± 0.2	12.0 ± 0.4	11.8 ± 0.7	12.6 ± 0.2	10.4 ± 0.3	86.2 ± 1.2	11.8 ± 0.3	12.7 ± 0.3
	Plane	99.0 ± 0.0	98.5 ± 0.5	99.0 ± 0.0	99.0 ± 0.0	98.8 ± 0.4	99.0 ± 0.0	99.0 ± 0.0	98.5 ± 0.5	99.0 ± 0.0	98.6 ± 0.5	99.2 ± 0.4	99.0 ± 0.0	100.0 ± 0.0	98.3 ± 0.4	98.5 ± 0.5	99.2 ± 0.4	100.0 ± 0.0	99.0 ± 0.0	99.0 ± 0.0
	PowerCons	94.2 ± 1.7	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	99.4 ± 0.0	-	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	100.0 ± 0.0	98.3 ± 0.0	100.0 ± 0.0	100.0 ± 0.0
	SemgHandGenderCh2	84.3 ± 0.3	95.5 ± 0.2	94.2 ± 0.5	94.6 ± 0.2	91.0 ± 0.3	91.5 ± 0.6	94.9 ± 0.1	93.1 ± 0.5	92.8 ± 0.1	96.4 ± 0.3	85.0 ± 0.2	95.8 ± 0.2	92.2 ± 0.0	95.5 ± 0.0	94.1 ± 0.1	94.3 ± 0.3	97.0 ± 0.3	93.9 ± 0.9	94.4 ± 0.6
	ShapesAll	53.4 ± 1.3	78.4 ± 0.1	75.9 ± 0.2	77.1 ± 0.3	73.9 ± 0.7	72.2 ± 0.2	75.3 ± 0.4	77.0 ± 0.3	73.4 ± 0.5	76.5 ± 0.7	72.2 ± 0.6	76.9 ± 0.1	63.2 ± 0.3	74.9 ± 0.3	75.6 ± 0.5	74.1 ± 0.5	89.9 ± 0.3	76.0 ± 0.4	77.4 ± 0.1
	SmallKitchenAppliances	76.8 ± 0.6	69.1 ± 1.1	73.8 ± 0.7	71.5 ± 0.5	69.5 ± 0.8	75.8 ± 0.2	71.5 ± 0.8	47.9 ± 0.7	70.2 ± 0.4	73.2 ± 0.8	75.8 ± 0.4	75.5 ± 1.9	51.8 ± 0.6	53.0 ± 0.8	47.5 ± 1.0	69.0 ± 0.9	78.9 ± 0.2	69.2 ± 0.1	69.6 ± 1.2
	StarLightCurves	93.4 ± 0.1	93.8 ± 0.2	97.9 ± 0.1	95.2 ± 0.2	95.6 ± 0.4	96.2 ± 0.2	94.5 ± 0.1	85.7 ± 0.0	95.0 ± 0.1	96.7 ± 0.1	96.7 ± 0.2	96.9 ± 0.0	92.3 ± 0.2	95.9 ± 0.4	94.8 ± 0.1	97.6 ± 0.1	97.7 ± 0.1	94.3 ± 0.1	94.3 ± 0.2
	Strawberry	96.6 ± 0.2	95.6 ± 0.6	97.0 ± 0.2	97.5 ± 0.2	96.7 ± 0.2	96.8 ± 0.1	97.4 ± 0.2	95.1 ± 0.2	96.2 ± 0.3	97.0 ± 0.2	95.5 ± 0.1	96.8 ± 0.1	94.6 ± 0.0	96.4 ± 0.3	96.1 ± 0.2	96.2 ± 0.4	97.0 ± 0.2	97.2 ± 0.4	96.7 ± 0.2
	SwedishLeaf	90.8 ± 0.4	90.5 ± 1.0	90.8 ± 0.5	93.7 ± 0.4	92.7 ± 0.4	92.9 ± 0.6	94.1 ± 0.3	91.2 ± 0.4	85.7 ± 0.3	90.8 ± 0.6	89.9 ± 0.5	92.2 ± 0.2	83.6 ± 0.1	85.8 ± 0.5	86.0 ± 0.3	92.2 ± 0.4	94.4 ± 0.2	90.0 ± 0.3	93.4 ± 0.3
	UWaveGestureLibraryZ	56.6 ± 0.3	72.3 ± 0.7	72.6 ± 0.2	69.4 ± 0.6	72.8 ± 0.1	68.3 ± 0.6	68.7 ± 0.4	70.3 ± 0.3	68.7 ± 0.2	72.2 ± 0.6	72.4 ± 0.3	71.7 ± 0.2	58.7 ± 0.1	71.6 ± 0.3	71.4 ± 0.2	71.4 ± 0.4	77.9 ± 0.3	67.1 ± 0.1	66.5 ± 0.2
	Wafer	99.7 ± 0.1	99.7 ± 0.0	99.6 ± 0.1	99.8 ± 0.0	99.7 ± 0.0	99.8 ± 0.0	99.8 ± 0.0	99.6 ± 0.1	99.7 ± 0.0	99.7 ± 0.1	99.6 ± 0.0	99.7 ± 0.0	95.5 ± 0.0	99.7 ± 0.1	99.6 ± 0.0	99.7 ± 0.0	99.9 ± 0.0	99.8 ± 0.0	99.9 ± 0.0
	Avg	74.4 ± 0.9	81.6 ± 0.9	84.4 ± 0.7	83.8 ± 0.6	82.9 ± 0.7	81.8 ± 0.9	83.8 ± 0.8	79.7 ± 0.8	82.6 ± 0.5	84.6 ± 0.8	75.4 ± 1.2	84.8 ± 0.5	77.5 ± 0.6	81.7 ± 0.9	81.8 ± 0.7	83.5 ± 0.6	89.1 ± 1.0	82.8 ± 0.5	83.9 ± 0.7
	Rank	19	15	4	6	9	12	6	16	11	3	18	2	17	14	12	8	1	10	5
Aggregated Results	Avg	71.9 ± 1.1	78.2 ± 1.0	80.6 ± 0.7	80.8 ± 0.7	78.5 ± 0.7	79.3 ± 0.9	81.3 ± 0.7	77.4 ± 0.8	78.9 ± 0.5	81.7 ± 0.9	74.5 ± 1.0	81.1 ± 0.5	75.6 ± 0.5	78.9 ± 0.7	79.0 ± 0.6	80.1 ± 0.8	84.2 ± 0.8	80.3 ± 0.6	80.4 ± 0.6
	Rank	19	15	6	5	14	10	3	16	12	2	18	4	17	12	11	9	1	8	7

Table 5: Full results for the long-term forecasting task. The results are based on top-5 records obtained during hyperparameter optimisation. The prediction length is fixed and equal to 192, while the input sequence length is tuned during the experiments.

Models	Transformer-Based												RNN-Based						MLP-Based						CNN-Based							
	Autoformer (2021)		Crossformer (2023)		ETSformer (2022)		Informer (2021)		PatchTST (2023)		Reformer (2020)		Mamba (2024)		S4 (2023)		SegRNN (2023)		DLinear (2023)		LightTS (2022)		TSMixer (2023)		ModernTCN (2024)		TS2Vec (2022)		TimesNet (Inception) (2023)		TimesNet (ResNeXt) (2023)	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
ETTh1	0.495 ± 0.02	0.487 ± 0.009	0.463 ± 0.033	0.468 ± 0.025	0.531 ± 0.023	0.508 ± 0.022	1.138 ± 0.034	0.802 ± 0.024	0.442 ± 0.047	0.43 ± 0.016	1.138 ± 0.125	0.801 ± 0.047	0.441 ± 0.02	0.449 ± 0.016	0.309 ± 0.005	0.364 ± 0.004	0.405 ± 0.008	0.425 ± 0.005	0.411 ± 0.009	0.42 ± 0.009	0.407 ± 0.004	0.416 ± 0.005	0.443 ± 0.003	0.453 ± 0.003	0.405 ± 0.001	0.414 ± 0.001	0.783 ± 0.016	0.656 ± 0.008	0.484 ± 0.014	0.458 ± 0.011	0.504 ± 0.05	0.469 ± 0.026
ETTh2	0.426 ± 0.005	0.421 ± 0.007	0.826 ± 0.309	0.649 ± 0.125	0.443 ± 0.079	0.442 ± 0.056	4.171 ± 1.072	1.504 ± 0.183	0.385 ± 0.029	0.404 ± 0.016	2.605 ± 0.795	1.27 ± 0.128	0.441 ± 0.025	0.427 ± 0.011	0.41 ± 0.007	0.414 ± 0.004	0.368 ± 0.012	0.4 ± 0.008	0.433 ± 0.048	0.446 ± 0.029	0.4 ± 0.038	0.429 ± 0.025	0.626 ± 0.07	0.563 ± 0.039	0.392 ± 0.04	0.4 ± 0.018	1.565 ± 0.035	0.978 ± 0.019	0.485 ± 0.049	0.449 ± 0.026	0.454 ± 0.013	0.44 ± 0.016
ETTm1	0.472 ± 0.026	0.475 ± 0.016	0.34 ± 0.006	0.378 ± 0.004	0.446 ± 0.027	0.457 ± 0.033	0.793 ± 0.038	0.655 ± 0.026	0.354 ± 0.022	0.382 ± 0.013	0.823 ± 0.067	0.651 ± 0.03	0.398 ± 0.016	0.409 ± 0.01	0.21 ± 0.005	0.292 ± 0.003	0.319 ± 0.003	0.368 ± 0.001	0.326 ± 0.001	0.359 ± 0.002	0.354 ± 0.009	0.388 ± 0.006	0.372 ± 0.016	0.409 ± 0.015	0.354 ± 0.014	0.387 ± 0.01	0.582 ± 0.005	0.541 ± 0.002	0.359 ± 0.012	0.385 ± 0.006	0.434 ± 0.055	0.422 ± 0.027
ETTm2	0.285 ± 0.048	0.34 ± 0.032	0.56 ± 0.384	0.483 ± 0.126	0.245 ± 0.004	0.323 ± 0.007	0.879 ± 0.611	0.68 ± 0.198	0.216 ± 0.001	0.288 ± 0.002	1.019 ± 0.186	0.802 ± 0.082	0.305 ± 0.026	0.342 ± 0.013	0.238 ± 0.013	0.308 ± 0.009	0.237 ± 0.02	0.3 ± 0.015	0.222 ± 0.004	0.3 ± 0.006	0.229 ± 0.005	0.311 ± 0.006	0.273 ± 0.026	0.358 ± 0.026	0.216 ± 0.004	0.291 ± 0.003	0.535 ± 0.029	0.553 ± 0.014	0.227 ± 0.001	0.297 ± 0.001	0.23 ± 0.002	0.304 ± 0.002
Electricity	0.202 ± 0.003	0.315 ± 0.003	0.158 ± 0.01	0.252 ± 0.008	0.195 ± 0.003	0.311 ± 0.002	0.298 ± 0.011	0.382 ± 0.009	0.147 ± 0.001	0.242 ± 0.002	0.31 ± 0.022	0.382 ± 0.012	0.192 ± 0.01	0.296 ± 0.012	0.134 ± 0.001	0.243 ± 0.002	0.149 ± 0.0	0.246 ± 0.001	0.147 ± 0.001	0.244 ± 0.001	0.147 ± 0.0	0.244 ± 0.0	0.148 ± 0.002	0.252 ± 0.003	0.165 ± 0.018	0.265 ± 0.022	0.375 ± 0.008	0.428 ± 0.007	0.195 ± 0.017	0.293 ± 0.014	0.185 ± 0.011	0.286 ± 0.01
Exchange	0.268 ± 0.023	0.377 ± 0.015	0.848 ± 0.146	0.693 ± 0.05	0.185 ± 0.01	0.305 ± 0.01	1.457 ± 0.262	0.992 ± 0.084	0.184 ± 0.012	0.306 ± 0.01	1.439 ± 0.252	0.983 ± 0.087	0.228 ± 0.045	0.337 ± 0.023	0.109 ± 0.01	0.233 ± 0.012	0.211 ± 0.038	0.332 ± 0.031	0.265 ± 0.106	0.37 ± 0.07	0.468 ± 0.345	0.486 ± 0.185	0.742 ± 0.352	0.67 ± 0.19	0.204 ± 0.018	0.318 ± 0.014	0.901 ± 0.084	0.709 ± 0.031	0.236 ± 0.038	0.35 ± 0.022	0.212 ± 0.013	0.331 ± 0.009
Traffic	0.598 ± 0.021	0.363 ± 0.015	0.499 ± 0.01	0.264 ± 0.005	0.583 ± 0.003	0.37 ± 0.006	0.675 ± 0.027	0.363 ± 0.01	0.391 ± 0.003	0.272 ± 0.006	0.675 ± 0.027	0.363 ± 0.019	0.648 ± 0.011	0.351 ± 0.007	0.558 ± 0.005	0.313 ± 0.008	0.536 ± 0.008	0.272 ± 0.003	0.396 ± 0.0	0.275 ± 0.001	0.396 ± 0.001	0.275 ± 0.001	0.432 ± 0.006	0.307 ± 0.008	0.423 ± 0.016	0.308 ± 0.017	0.762 ± 0.0	0.427 ± 0.0	0.638 ± 0.028	0.341 ± 0.024	0.689 ± 0.036	0.381 ± 0.04
Weather	0.294 ± 0.008	0.356 ± 0.007	0.212 ± 0.033	0.27 ± 0.014	0.255 ± 0.029	0.339 ± 0.037	0.389 ± 0.137	0.425 ± 0.101	0.191 ± 0.005	0.242 ± 0.005	0.741 ± 0.307	0.612 ± 0.159	0.216 ± 0.008	0.262 ± 0.008	0.105 ± 0.008	0.142 ± 0.009	0.209 ± 0.027	0.267 ± 0.019	0.189 ± 0.0	0.256 ± 0.0	0.196 ± 0.003	0.263 ± 0.003	0.196 ± 0.006	0.267 ± 0.008	0.199 ± 0.018	0.25 ± 0.015	0.851 ± 0.012	0.645 ± 0.006	0.212 ± 0.008	0.259 ± 0.006	0.21 ± 0.006	0.258 ± 0.008
Avg	0.38 ± 0.024	0.392 ± 0.015	0.488 ± 0.183	0.432 ± 0.066	0.36 ± 0.033	0.382 ± 0.028	1.225 ± 0.449	0.725 ± 0.107	0.289 ± 0.022	0.321 ± 0.01	1.094 ± 0.325	0.733 ± 0.086	0.359 ± 0.023	0.359 ± 0.013	0.259 ± 0.008	0.289 ± 0.007	0.304 ± 0.019	0.326 ± 0.014	0.299 ± 0.041	0.334 ± 0.027	0.325 ± 0.123	0.352 ± 0.066	0.404 ± 0.127	0.41 ± 0.07	0.295 ± 0.019	0.329 ± 0.014	0.794 ± 0.035	0.617 ± 0.014	0.354 ± 0.026	0.354 ± 0.016	0.365 ± 0.03	0.361 ± 0.021
Rank	11	11	13	13	9	10	16	15	2	2	15	16	8	8	1	1	5	3	4	5	6	6	12	12	3	4	14	14	7	7	10	9

Table 6: Dataset descriptions for the forecasting task. The dataset size represents the (Train, Validation, Test) split which adheres to the configuration proposed by Wu et al. (2023). The forecasting horizon is set to 192 for all datasets as a mid-range value.

Category	Dataset	# Dimensions	Series Length	Dataset Size	Frequency
Electricity	ETTh1	7	{24, 48, 96, 168, 336, 720}	(8545, 2881, 2881)	Hourly
	ETTh2	7	{24, 48, 96, 168, 336, 720}	(8545, 2881, 2881)	Hourly
	ETTm1	7	{24, 48, 96, 192, 288, 672}	(34465, 11521, 11521)	15 Min.
	ETTm2	7	{24, 48, 96, 192, 288, 672}	(34465, 11521, 11521)	15 Min.
	Electricity	321	{24, 48, 96, 168, 336, 720}	(18317, 2633, 5261)	Hourly
Currency	Exchange	8	{96, 192, 336, 720}	(5120, 665, 1422)	Daily
Traffic	Traffic	862	{24, 48, 96, 168, 336, 720}	(12185, 1757, 3509)	Hourly
Weather	Weather	21	{24, 48, 96, 144, 168, 192, 288, 336, 720}	(36792, 5271, 10540)	10 Min.

Table 7: Dataset descriptions for UEA benchmark. The dataset size represents the (Train, Test) split specified by the original authors. The dataset marked red is excluded from our experiments.

Category	Dataset	# Dimensions	# Classes	Series Length	Dataset Size
Audio	DuckDuckGeese	1345	5	270	(60, 40)
	Heartbeat	61	2	405	(204, 205)
	JapaneseVowels	12	9	29	(270, 370)
	InsectWingbeat	200	10	78	(30000, 20000)
	Phoneme	11	39	217	(3315, 3353)
	SpokenArabicDigits	13	10	93	(6599, 2199)
ECG	AtrialFibrillation	2	3	640	(15, 15)
	StandWalkJump	4	3	2500	(12, 15)
EEG/MEG	FaceDetection	144	2	62	(5890, 3524)
	FingerMovements	28	2	50	(316, 100)
	HandMovementDirection	10	4	400	(320, 147)
	MotorImagery	64	2	3000	(278, 100)
	SelfRegulationSCP1	6	2	896	(268, 293)
	SelfRegulationSCP2	7	2	1152	(200, 180)
HAR	BasicMotions	6	4	100	(40, 40)
	Cricket	6	12	1197	(108, 72)
	Epilepsy	3	4	206	(137, 138)
	ERing	4	6	65	(30, 30)
	Handwriting	3	26	152	(150, 850)
	Libras	2	15	45	(180, 180)
	NATOPS	24	6	51	(180, 180)
	RacketSports	6	4	30	(151, 152)
	UWaveGestureLibrary	3	8	315	(120, 320)
Motion	ArticularyWordRecognition	9	25	144	(275, 300)
	CharacterTrajectories	3	20	182	(1422, 1436)
	EigenWorms	6	5	17984	(128, 131)
	PenDigits	2	10	8	(7494, 3498)
Other	LSST	6	14	36	(2459, 2466)
	PEMS-SF	963	7	144	(267, 173)
Spectro	EthanolConcentration	3	4	1751	(261, 263)

Table 8: Dataset descriptions for 45 sampled UCR datasets. The dataset size represents the (Train, Test) split specified by the original authors.

Category	Dataset	# Classes	Series Length	Dataset Size
Device	HouseTwenty	2	2000	(40, 119)
	ElectricDevices	7	96	(8926, 7711)
	SmallKitchenAppliances	3	720	(375, 375)
ECG	ECG200	2	96	(100, 100)
	ECG5000	5	140	(500, 4500)
EOG	EOGVerticalSignal	12	1250	(362, 362)
EPG	InsectEPGSmallTrain	3	601	(17, 249)
HRM	Fungi	18	201	(18, 186)
Image	Crop	24	46	(7200, 16800)
	MedicalImages	10	99	(381, 760)
	ShapesAll	60	512	(600, 600)
	MiddlePhalanxOutlineCorrect	2	80	(600, 291)
	FaceAll	14	131	(560, 1690)
	FaceFour	4	350	(24, 88)
	MixedShapesRegularTrain	5	1024	(500, 2425)
	SwedishLeaf	15	128	(500, 625)
	Fish	7	463	(175, 175)
	Adiac	37	176	(390, 391)
	DiatomSizeReduction	4	345	(16, 306)
Motion	CricketX	12	300	(390, 390)
	CricketY	12	300	(390, 390)
	InlineSkate	7	1882	(100, 550)
	GunPointMaleVersusFemale	2	150	(135, 316)
	UWaveGestureLibraryZ	8	315	(896, 3582)
	GunPointAgeSpan	2	150	(135, 316)
Power	PowerCons	2	144	(180, 180)
Sensor	GesturePebbleZ2	6	Vary	(146, 158)
	StarLightCurves	3	1024	(1000, 8236)
	DodgerLoopWeekend	2	288	(20, 138)
	Wafer	2	152	(1000, 6164)
	Lightning2	2	637	(60, 61)
	Lightning7	7	319	(70, 73)
	PickupGestureWiimoteZ	10	Vary	(50, 50)
	FreezerRegularTrain	2	301	(150, 2850)
	Plane	7	144	(105, 105)
	Car	4	577	(60, 60)
Simulated	Mallat	8	1024	(55, 2345)
	BME	3	128	(30, 150)
	CBF	3	128	(30, 900)
Spectro	Coffee	2	286	(28, 28)
	Strawberry	2	235	(613, 370)
	Beef	5	470	(30, 30)
Spectrum	SemgHandGenderCh2	2	1500	(300, 600)
Traffic	MelbournePedestrian	10	24	(1194, 2439)
Trajectory	GestureMidAirD1	26	Vary	(208, 130)

Table 9: List of 28 meta-features used for describing time-series datasets

Category	Features
Statistical	Absolute energy, Entropy, Histogram mode, Interquartile range, Kurtosis, Max, Mean, Mean absolute deviation, Median, Median absolute deviation, Min, Peak to peak distance, Root mean square, Skewness, Standard deviation, Variance
Temporal	Autocorrelation, Mean absolute difference, Mean difference, Median absolute difference, Median difference, Negative turning points, Neighborhood peaks, Positive turning points, Signal distance, Slope, Sum absolute difference, Zero crossing rate

$d_{agg} \in \mathbb{R}^{7K}$, as the N variables will be marginalized out during aggregation with the aforementioned operations. As indicated by Brazdil et al. (2022), the main reason for using different aggregation functions is to ensure the elimination of any possible over-smoothing effect and also provide more information about the distribution of the underlying datasets.

While the general idea for extracting the meta-features for different tasks are similar in essence, there's an additional step for encoding classification datasets. A classification time-series dataset consists of M samples, each of which is a time-series in $\mathbb{R}^{L \times N}$. Thus, once the aggregated meta-features are calculated per sample based on the procedure illustrated in Figure 1, a second level of aggregation is applied over the meta-feature vectors across all samples to summarize the whole encoding into a unified vector using the same aggregation operations as before. This will result in a vector $d'_{agg} \in \mathcal{R}^{49K}$.

D Architectures and Hyperparameters

In the following, we provide a complete list of training and model-specific hyperparameters used in our HPO and meta-prediction experiments for each time-series task. The number of training epochs and batch size are kept fixed throughout the experiments for each task. Table 10 and Table 11 provide a full list of training HPs for the time-series models used for classification and forecasting tasks, respectively. Similarly, Table 12 and Table 13 present the model-specific HPs for the time-series models used in classification and forecasting tasks, respectively.

Table 10: Training hyperparameters for classification experiments

Hyperparameter	Value
Learning Rate	$\{1e-05, 5e-05, 1e-04, 5e-04, 1e-03, 5e-03, 1e-02, 5e-02\}$
Optimizer	$\{'Adam', 'AdamW', 'RAdam', 'SGD'\}$
Momentum	$\{0.9, 0.95, 0.97, 0.99\}$
Weight Decay	$\{1e-06, 1e-05, 1e-04, 1e-03, 1e-02, 1e-01\}$
Batch Size	16
Training Epochs	30
Early Stopping Patience	3

Table 11: Training hyperparameters for forecasting experiments

Hyperparameter	Value
Learning Rate	$\{1e-05, 5e-05, 1e-04, 5e-04, 1e-03, 5e-03, 1e-02, 5e-02\}$
Optimizer	$\{'Adam', 'AdamW', 'RAdam', 'SGD'\}$
Momentum	$\{0.9, 0.95, 0.97, 0.99\}$
Weight Decay	$\{1e-06, 1e-05, 1e-04, 1e-03, 1e-02, 1e-01\}$
Learning Rate Scheduler	$\{'cosine', 'exponential', 'none'\}$
Batch Size	16
Training Epochs	10

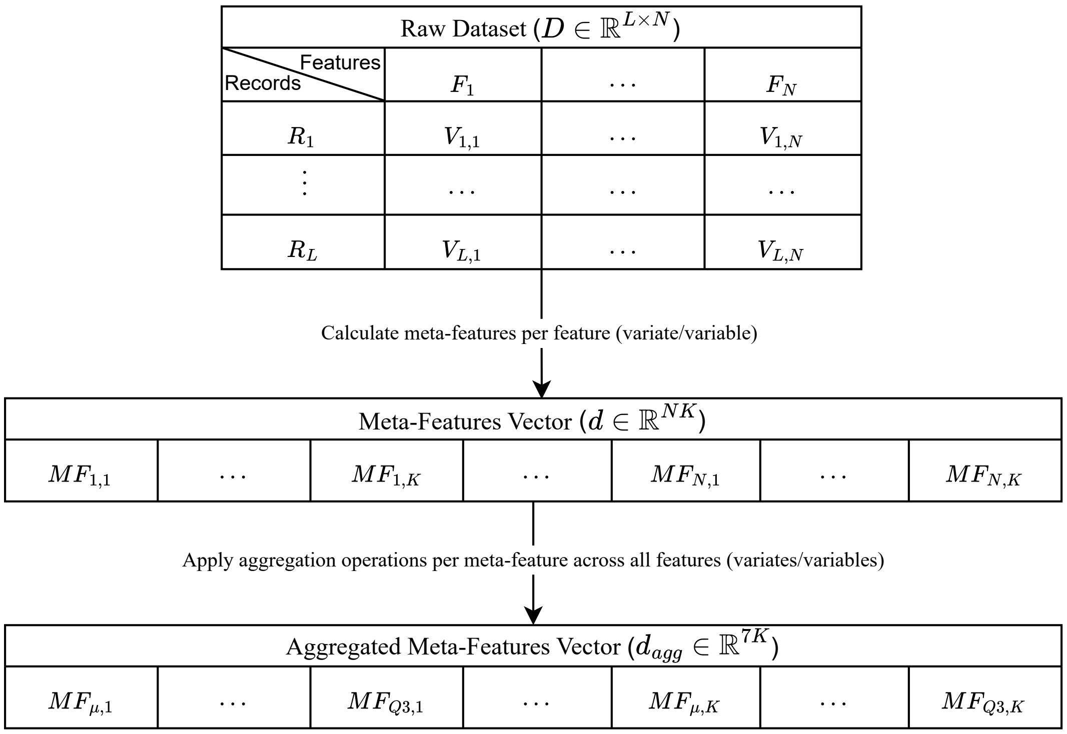

Raw Dataset ($D \in \mathbb{R}^{L \times N}$)
Features
Records
F_1
$\cdots$
F_N
R_1
$V_{1,1}$
$V_{1,N}$
R_L
$V_{L,1}$
$V_{L,N}$
Calculate meta-features per feature (variate/variable)
Meta-Features Vector ($d \in \mathbb{R}^{NK}$)
$MF_{1,1}$
$MF_{1,K}$
$MF_{N,1}$
$MF_{N,K}$
Apply aggregation operations per meta-feature across all features (variates/variables)
Aggregated Meta-Features Vector ($d_{agg} \in \mathbb{R}^{7K}$)
$MF_{\mu,1}$
$MF_{Q3,1}$
$MF_{\mu,K}$
$MF_{Q3,K}$

Figure 1: Overview of meta-feature encoding strategy for time-series datasets. Starting from the raw dataset D (table on the top) with temporal length L (with R_i representing the record at the i-th time step) and N variables (denoted as F_j), i.e., $D \in \mathbb{R}^{L \times N}$, the meta-features are initially calculated for each variable dimension (based on values $V_{i,j}$, denoting the value of the j-th feature for the i-th record). The outcome of the first phase will be a raw meta-feature vector (middle table), where $MF_{j,k}$ denotes the k-th meta-feature calculated for the j-th variable. In the second phase, all meta-features corresponding to the same variable are aggregated through a list of aggregation operations, resulting in the final meta-feature encoding vector d_{agg} (table at the bottom), where $MF_{o,k}$ denotes the aggregated value for the k-th meta-feature based on aggregation operation o.

Table 12: Complete list of hyperparameters along with their value domains for time-series models used for classification experiments

Model	Hyperparameter	Value
Autoformer (Wu et al. 2021)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Moving Average Window Size	{5, 15, 25, 35, 45}
	Attention Heads	{2, 4, 8}
	C (Auto-Correlation Factor)	{1, 2, 3, 5}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Encoder Layers	{2, 3, 4, 6, 8}
Crossformer (Zhang and Yan 2023)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Encoder Layers	{1, 2, 3, 4}
	Number of Routers	{3, 5, 10, 15, 20}
	Segment Length (DSW Embedding)	{4, 6, 8, 12, 24}
	Dropout	{0.0, 0.05, 0.1, 0.2}
DLinear (Zeng et al. 2023)	Moving Average Window Size	{5, 15, 25, 35, 45}
	Channel Independence	{0, 1}
	Dropout	{0.0, 0.05, 0.1, 0.2}
ETSformer (Woo et al. 2022)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Encoder Layers	{2, 3, 4, 6, 8}
	K (Top-K Frequency Amplitudes)	{1, 2, 3, 4, 5}
	Dropout	{0.0, 0.05, 0.1, 0.2}
FiLM (Zhou et al. 2022)	HiPPO Projection Order	{16, 32, 64, 128}
	Frequency Mode Selection Method	{'low', 'random'}
	Number of Frequency Modes	{4, 8, 16, 32}
	Dropout	{0.0, 0.05, 0.1, 0.2}
Informer (Zhou et al. 2021)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Encoder Layers	{2, 3, 4, 6, 8}
	C (ProbSparse Attention Factor)	{1, 2, 3, 5, 8, 10}
	Dropout	{0.0, 0.05, 0.1, 0.2}
LMU (Voelker, Kajic, and Eliasmith 2019)	LMU Version	{'lmu', 'lmufft'}
	LMU Layers	{1, 2, 3, 4, 6}
	Input/Hidden Size	{16, 32, 64, 128, 256}
	Memory Size	{16, 32, 64, 128, 256}
	Dropout	{0.0, 0.05, 0.1, 0.2}
LightTS (Zhang et al. 2022)	Feature Dimension	{16, 32, 64, 128, 256, 512, 1024, 2048}
	Chunk Size/Sub-sequence Length	{4, 6, 8, 12, 24, 36, 48, 96}
	Dropout	{0.0, 0.05, 0.1, 0.2}
ModernTCN (Donghao and Xue 2024)	Block Input Channels	{16, 32, 64, 128}
	Number of Stages	{1, 2}
	Number of Blocks	{1, 2}
	Downsampling Ratio	{1, 2, 4}
	Feedforward Ratio	{1, 2, 4}
	Small Kernel Size	{3, 5, 7}
	Large Kernel Size	{13, 27, 55}
	Moving Average Window Size	{5, 15, 25, 35, 45}
	Dropout	{0.0, 0.05, 0.1, 0.2}

	Patch Size	{4, 8, 12, 16}
	Patch Stride	{4, 8, 12, 16}
	Channel Independence	{0, 1}
	Seasonal/Trend Decomposition	0
PatchTST (Nie et al. 2023)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Patch Length	{4, 8, 12, 16}
	Patch Stride	{4, 8, 12, 16}
	Encoder Layers	{2, 3, 4, 6, 8}
Reformer (Kitaev, Kaiser, and Levskaya 2020)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Encoder Layers	{2, 3, 4, 6, 8}
	Attention Heads	{2, 4, 8}
	LSH Bucket Size	{2, 4, 8, 16, 32, 64}
	Number of Hashes	{2, 4, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}
S4 (Hasani et al. 2023)	Model Embedding Dimension	{16, 32, 64, 128}
	State Dimension	{64, 128, 256, 512}
	Liquid Kernel	{'KB', 'PB', 'none'}
	Liquid Order/Degree	{2, 3, 4}
	LRA Rank	{1, 2, 3, 4, 5}
	Number of Layers	1
	Dropout	{0.0, 0.05, 0.1, 0.2}
SegRNN (Lin et al. 2023)	RNN Model Input/Hidden Size	{16, 32, 64, 128, 256, 512}
	Segment Length	{4, 6, 8, 12, 24, 36, 48, 96}
	Dropout	{0.0, 0.05, 0.1, 0.2}
TS2Vec (Yue et al. 2022)	Representation Dimension	{16, 32, 64, 128, 256, 512}
	Encoder Layers	{1, 2, 3, 4, 6, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Hidden Dimension	{16, 32, 64, 128}
TSMixer (Chen et al. 2023)	Mixing Layers	{1, 2, 3, 4, 6}
	Feature Dimension	{16, 32, 64, 128}
	Dropout	{0.0, 0.05, 0.1, 0.2}
TimesNet (Inception) (Wu et al. 2023)	TimesBlock	Inception
	K (Top-K Frequency Amplitudes)	{1, 2, 3, 4, 5}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Model Embedding Dimension	{16, 32, 64}
	TimesBlock Hidden Dimension	{16, 32, 64}
	Number of Stacked TimesBlocks	{1, 2, 3}
	Multi-scale Kernels	{1, 2, 3, 4}
TimesNet (ResNeXt) (Wu et al. 2023)	TimesBlock	ResNeXt
	K (Top-K Frequency Amplitudes)	{1, 2, 3, 4, 5}
	Base Width	{4, 14, 24, 40, 64}
	Cardinality (Group Convolution)	{1, 2, 4, 8, 32}
	Expansion Factor	{2, 4}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Model Embedding Dimension	{64, 128}
	TimesBlock Hidden Dimension	{32, 64, 128}
	Number of Stacked TimesBlocks	{1, 2, 3, 4}
Transformer (Vaswani et al. 2017)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}

	Encoder Layers	{2, 3, 4, 6, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}
iTransformer (Liu et al. 2024)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Encoder Layers	{2, 3, 4, 6, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}

Table 13: Complete list of hyperparameters along with their value domains for time-series models used for forecasting experiments

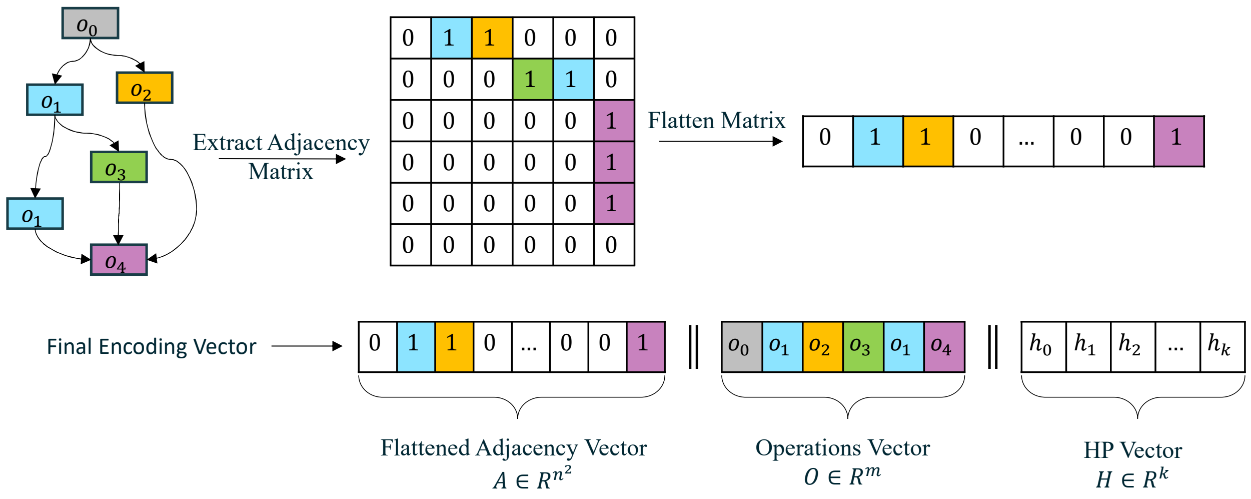
Model	Hyperparameter	Value
Autoformer (Wu et al. 2021)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Moving Average Window Size	{5, 15, 25, 35, 45}
	Attention Heads	{2, 4, 8}
	C (Auto-Correlation Factor)	{1, 2, 3, 5}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Encoder Layers	{2, 3, 4, 6}
	Decoder Layers	{2, 3, 4, 6}
Crossformer (Zhang and Yan 2023)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Encoder Layers	{1, 2, 3, 4}
	Number of Routers	{3, 5, 10, 15, 20}
	Segment Length (DSW Embedding)	{4, 6, 8, 12, 24}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Decoder Layers	{2, 3, 4, 5}
DLinear (Zeng et al. 2023)	Moving Average Window Size	{5, 15, 25, 35, 45}
	Channel Independence	{0, 1}
ETSformer (Woo et al. 2022)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Encoder Layers	{2, 3, 4, 6, 8}
	K (Top-K Frequency Amplitudes)	{1, 2, 3, 4, 5}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Decoder Layers	{2, 3, 4, 6, 8}
Informer (Zhou et al. 2021)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Encoder Layers	{2, 3, 4, 6, 8}
	C (ProbSparse Attention Factor)	{1, 2, 3, 5, 8, 10}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Decoder Layers	{2, 3, 4, 6}
	Distil	{0, 1}
LightTS (Zhang et al. 2022)	Feature Dimension	{16, 32, 64, 128, 256, 512, 1024, 2048}
	Chunk Size/Sub-sequence Length	{4, 6, 8, 12, 24, 36, 48, 96}
Mamba (Gu and Dao 2024)	Embedding Dimension	{16, 32, 64, 128}
	State Dimension	{64, 128, 256, 512}
	Conv Kernel Size	{2, 3, 4}
	Expansion Factor	{1, 2, 4}
	Mamba Layers	1

	Number of Heads	8
ModernTCN (Donghao and Xue 2024)	Block Input Channels	{16, 32, 64, 128}
	Number of Stages	{1, 2}
	Number of Blocks	{1, 2}
	Downsampling Ratio	{1, 2, 4}
	Feedforward Ratio	{1, 2, 4}
	Small Kernel Size	{3, 5, 7}
	Large Kernel Size	{13, 27, 55}
	Moving Average Window Size	{5, 15, 25, 35, 45}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Patch Size	{4, 8, 12, 16}
	Patch Stride	{4, 8, 12, 16}
	Channel Independence	{0, 1}
	Seasonal/Trend Decomposition	{0, 1}
PatchTST (Nie et al. 2023)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Attention Heads	{2, 4, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Patch Length	{4, 8, 12, 16}
	Patch Stride	{4, 8, 12, 16}
	Encoder Layers	{2, 3, 4, 6, 8}
Reformer (Kitaev, Kaiser, and Levskaya 2020)	Model Embedding Dimension	{16, 32, 64, 128}
	Feedforward Dimension	{64, 128, 256, 512}
	Encoder Layers	{2, 3, 4, 6, 8}
	Attention Heads	{2, 4, 8}
	LSH Bucket Size	{2, 4, 8, 16, 32, 64}
	Number of Hashes	{2, 4, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}
S4 (Hasani et al. 2023)	Model Embedding Dimension	{16, 32, 64, 128}
	State Dimension	{64, 128, 256, 512}
	Liquid Kernel	{'KB', 'PB', 'none'}
	Liquid Order/Degree	{2, 3, 4}
	LRA Rank	{1, 2, 3, 4, 5}
	Number of Layers	1
	Dropout	{0.0, 0.05, 0.1, 0.2}
SegRNN (Lin et al. 2023)	RNN Model Input/Hidden Size	{16, 32, 64, 128, 256, 512}
	Segment Length	{4, 6, 8, 12, 24, 36, 48, 96}
	Dropout	{0.0, 0.05, 0.1, 0.2}
TS2Vec (Yue et al. 2022)	Representation Dimension	{16, 32, 64, 128, 256, 512}
	Encoder Layers	{1, 2, 3, 4, 6, 8}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Hidden Dimension	{16, 32, 64, 128, 256}
TSMixer (Chen et al. 2023)	Mixing Layers	{1, 2, 3, 4, 6}
	Feature Dimension	{16, 32, 64, 128, 256, 512}
TimesNet (Inception) (Wu et al. 2023)	TimesBlock	Inception
	K (Top-K Frequency Amplitudes)	{1, 2, 3, 4, 5}
	Dropout	{0.0, 0.05, 0.1, 0.2}
	Model Embedding Dimension	{16, 32, 64, 128}
	TimesBlock Hidden Dimension	{16, 32, 64, 128}
	Number of Stacked TimesBlocks	{1, 2, 3, 4}
	Multi-scale Kernels	{1, 2, 3}
TimesNet (ResNeXt) (Wu et al. 2023)	TimesBlock	ResNeXt
	K (Top-K Frequency Amplitudes)	{1, 2, 3, 4, 5}
	Base Width	{4, 14, 24, 40, 64}

Cardinality (Group Convolution)	{1, 2, 4, 8, 32}
Expansion Factor	{2, 4}
Dropout	{0.0, 0.05, 0.1, 0.2}
Model Embedding Dimension	{64, 128, 256}
TimesBlock Hidden Dimension	{32, 64, 128, 256}
Number of Stacked TimesBlocks	{1, 2, 3}

E Arch-Hyper Encoding

As already mentioned, the representation of architectures along with their hyperparameters is another significant factor that impacts the performance of the meta-predictor. One frequently-used approach is to represent a neural architecture as a directed acyclic graph, where the nodes represent the operation types and their inter-connections define the information flow between them. However, there also exists other conventions where the roles of nodes and edges are exchanged. This class of encoding captures the structural attributes of neural architectures based on the representations of the adjacency and operation matrices. The adjacency matrix encodes information about nodes' connectivity, and the operation matrix (also referred to as feature matrix) represents the operation of each node. The method that we consider in this category is the *one-hot adjacency matrix encoding*, where the final encoding vector is obtained by flattening the adjacency matrix and concatenating it with the operations vector. Figure 2 illustrates an example of a neural architecture directed acyclic graph (DAG) along with its one-hot encoding. In order to jointly encode information about the HPs, we also concatenate the min-max normalized k-dimensional HP vector to the previously obtained one-hot encoding vector.



$$\begin{array}{cccccc}
0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 2: Overview of joint one-hot encoding of neural architectures and hyperparameters (c.f. White et al. (2020)). Operation types o_i are distinguished through a color-encoding scheme. $\|$ denotes the concatenation operator.

Similar to the work by (Lukasik et al. 2021), we apply the following slight modification to neural architectures of different sizes when encoding them with the specified scheme: we zero-pad the one-hot encoding vector such that all architectures are represented with the same dimensionality. Moreover, the operations vector indicates the functions that are applied on intermediate feature maps while passing time-series datasets through the building blocks of neural architectures. The vector consists of normalized categorical values for the corresponding operations of each architecture. We reserve $NOOP = 0$ to denote no operation for padding the vector operations O to a fixed dimension of size m. Similarly, different architectures are associated with different HPs. In order to ensure consistency between architectures in terms of the dimensionality of their HP vector, we maintain a fixed size k that represents the total number of all recognized HP configurations across all benchmarked architectures in our experiments. Consequently, we need to reserve a specific value to denote irrelevance between the neural architectures and certain HPs. For instance, while the dimension of the feed-forward module might be a relevant HP for transformer-based architectures, it might be ignored by architectures with other backbones. We define this reserved value to be $NOHP = -1$ since it's not contained in the value domain of any of the identified HPs.

F Ablation Study: Generalization Capability to Unseen Architectures and Datasets

This ablation study aims to evaluate the generalization capability of the surrogate model to unseen architectures and datasets. In this study, we excluded the models with the same architecture (i.e., transformer, CNN, RNN, and MLP) at each round from the training set and tested the surrogate's performance on these unseen models. Additionally, we conducted cross-domain validation by training the meta-predictor on the datasets of all but one domain and testing it on the datasets from the excluded domain. This approach helps to assess the generalization ability of the meta-predictor across time-series data of different subject categories. Finally, we also evaluated the combined effect of excluding both unseen architectures and datasets during training to observe the surrogate's performance in a more challenging scenario. In the *No Cutout* scenario, all the models and datasets were included. In the *Dataset Cutout* scenario, we excluded datasets from one domain during training. In the *Model Cutout* scenario, we excluded models of one architecture during training. In the *Dataset + Model Cutout* scenario, we excluded both datasets from one domain and models of one architecture during training. The results are summarized in the table below. Except for the No Cutout scenario (the baseline), we report the change in performance compared to the baseline for each metric. we provide the mean and standard deviation of each metric across different rounds of exclusion with 3 seeds to ensure the robustness of the results.

Table 14: Cutout table

Cutout	Kendall's Tau		Spearman's Rank Correlation		MSE		MAE	
	Value	Change	Value	Change	Value	Change	Value	Change
No Cutout	0.695 ± 0.005	-	0.868 ± 0.003	-	0.015 ± 0.001	-	0.085 ± 0.002	-
Dataset Cutout	0.589 ± 0.098	$\%14.72 \downarrow$	0.759 ± 0.115	$\%12.26 \downarrow$	0.020 ± 0.003	$\%26.32 \uparrow$	0.102 ± 0.014	$\%17.31 \uparrow$
Model Cutout	0.534 ± 0.102	$\%22.71 \downarrow$	0.618 ± 0.153	$\%28.51 \downarrow$	0.020 ± 0.003	$\%27.43 \uparrow$	0.107 ± 0.015	$\%22.59 \uparrow$
Dataset+Model Cutout	0.438 ± 0.073	$\%36.63 \downarrow$	0.560 ± 0.083	$\%35.27 \downarrow$	0.022 ± 0.003	$\%40.89 \uparrow$	0.121 ± 0.015	$\%38.89 \uparrow$

G Implementation Details

In this section, we present the underlying technologies used for the implementation of the proposed frameworks. Moreover, we discuss our method for choosing and splitting data, the third-party libraries used for extracting time-series meta-features, and the method for extracting the computation DAG of neural architectures.

G.1 Development Environment

Both of the proposed frameworks are implemented using the *Python 3.10* programming language and *PyTorch 2.2* deep-learning framework. Moreover, the implementation has been tested in *CUDA*-aware environments on *NVIDIA A100* and *H100* GPUs using *CUDA 12.1* Toolkit and *cuDNN 8.0* runtime package for GPU acceleration.

G.2 Time-Series Models and Hyperparameter Optimization

We utilize the publicly available *Time-Series-Library* [1] to interact with a wide range of time-series algorithms and datasets. It implements the logic for various tasks such as classification, forecasting, imputation, and anomaly detection. The HPO framework uses this library as its core module. Additionally, the *Ray* framework [2] is integrated to apply state-of-the-art HPO algorithms and also integrate custom optimization strategies. In our initial HPO experiments as the baseline, we utilize *HEBO* algorithm with 128 trials per model-dataset pair.

G.3 Third-Party Libraries for Dataset and Architecture Encoding

To extract time-series features, we use the *TSFEL* [3] library, which offers a large set of functional routines for this purpose. However, as already mentioned, we only use a subset of these features (see Table 9) to represent time-series datasets.

To extract the structural representation of neural architectures, we utilize the openly accessible *torchview* library [4]. It records all the modules, functional units, tensors and their shapes during the forward propagation of the networks, and allows access to the captured information through a *Computation Graph* object. Additionally, the Computation

[1] See https://github.com/thuml/Time-Series-Library

[2] See https://www.ray.io

[3] See https://github.com/fraunhoferportugal/tsfel

[4] See https://github.com/mert-kurttutan/torchview

Graph represents the topological structure of the network, based on which we extract the adjacency matrix and node operations for its corresponding *DAG*.

Finally, the meta-learning framework gathers all the outputs extracted from the previous modules to train a meta-aware network that can later be integrated into *Ray* as a custom search strategy. This requires defining a search space, an objective function, a search/suggestion function, and a metric to optimize for.

G.4 Data Selection and Splitting Strategy

We adopt a leave-one-out strategy for training the meta-predictor, i.e., when analysing the prediction performance of the meta-predictor for one dataset, we initially exclude it from the list of all datasets that we used for HPO experiments. Then, we utilize the weights of the trained meta-predictor to predict the performance of the architectures and HP configurations over the dataset which was initially excluded during the training phase. Figure 3 illustrates the meta-predictor data splitting strategy during the training phase and the cutout data that serves to evaluate the performance of the trained meta-predictor.

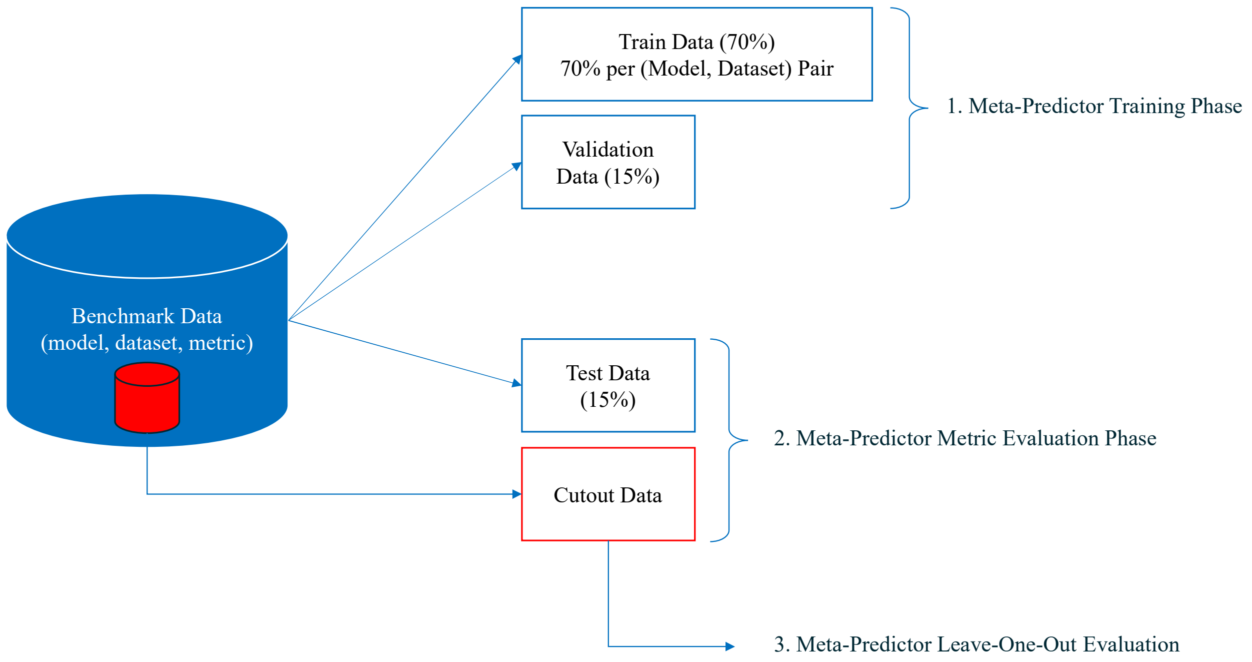

Train Data (70%)
70% per (Model, Dataset) Pair
Validation Data (15%)
1. Meta-Predictor Training Phase
Benchmark Data (model, dataset, metric)
Test Data (15%)
Cutout Data
2. Meta-Predictor Metric Evaluation Phase
3. Meta-Predictor Leave-One-Out Evaluation

Figure 3: Overview of the meta-predictor data splitting and cutout strategy

G.5 Performance Metrics

To compare the performance of different time-series algorithms, we adopt accuracy for the classification tasks, and mean squared error (MSE) and mean absolute error (MAE) for the forecasting tasks. Moreover, to measure the performance of the meta-predictors and compare them against each other, we use the regression metrics such as MSE, MAE, and R^2 score. Ultimately, we measure the ordering capability of the meta-predictors through Spearman's rank correlation coefficient (SRC) and Kendall's Tau (τ), which can be mathematically formalized as below:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}, \qquad \tau = \frac{C-D}{C+D},$$

where d_i in the SRC formula denotes the difference between the true and predicted ranks, and C and D in the τ formula refer to the number of concordant and discordant pairs, respectively.

Table 15 summarizes the performance of meta-predictors with different data and architecture encoding schemes. The results are reported by running the experiment using 3 different seed.

H Algorithm Implementation

In this section, we provide the implementation of most important algorithms covered in this paper using a *Pythonic* syntax to ease reproducibility. The following algorithms are provided in this section: (i) time-series data encoding

Table 15: Performance of meta-predictors trained with different encodings; the best performance is underlined. The error bounds for the inference times are negligible.

Encoding		Spearman's Rank	Kendall's	MSE	MAE	Inference
Data	Architecture	Correlation	Tau			Time
MFE	arch2vec	0.848 $\pm$0.003	0.668 $\pm$0.004	0.018 $\pm$0.001	0.095 $\pm$0.002	$3.2e-04$
MFE	AdjOp+GCN	0.841 $\pm$0.005	0.658 $\pm$0.007	0.020 $\pm$0.001	0.101 $\pm$0.003	$8.0e-05$
MFE	AdjOp+GIN	0.856 $\pm$0.004	0.678 $\pm$0.005	0.017 $\pm$0.001	0.093 $\pm$0.003	$1.2e-04$
MFE	OHE	$\underline{0.868 \pm 0.003}$	$\underline{0.695 \pm 0.005}$	$\underline{0.015 \pm 0.001}$	0.085 $\pm$0.002	$\underline{5.0e-05}$
MFE	OHE+LSTM	0.866 $\pm$0.006	0.693 $\pm$0.009	0.015 $\pm$0.001	0.085 $\pm$ 0.005	$1.1e-04$
MFE	OHE+Transformer	$\underline{0.868 \pm 0.002}$	0.694 $\pm$0.004	$\underline{0.015 \pm 0.001}$	$\underline{0.084 \pm 0.003}$	$1.7e-04$
VAE	arch2vec	0.838 $\pm$0.007	0.656 $\pm$0.009	0.019 $\pm$0.001	0.099 $\pm$0.005	$3.2e-04$
VAE	AdjOp+GCN	0.837 $\pm$0.001	0.655 $\pm$0.002	0.020 $\pm$0.000	0.103$\pm$0.001	$9.0e-05$
VAE	AdjOp+GIN	0.854 $\pm$0.003	0.677 $\pm$0.004	0.018 $\pm$0.001	0.094 $\pm$0.002	$1.4e-04$
VAE	OHE	0.862 $\pm$0.005	0.688 $\pm$0.008	0.016 $\pm$0.001	0.089 $\pm$0.004	$6.0e-05$
VAE	OHE+LSTM	0.863 $\pm$0.002	0.690 $\pm$0.003	0.016 $\pm$0.000	0.088 $\pm$0.002	$1.2e-04$
VAE	OHE+Transformer	0.854 $\pm$0.022	0.676 $\pm$0.030	0.018 $\pm$0.004	0.096 $\pm$0.014	$1.8e-04$

algorithm using meta-features, (ii) joint architecture and HP (referred to as arch-hyper) one-hot encoding, (iii) meta-predictor building blocks and training loop, and (iv) meta-search strategy using a trained meta-predictor and all historical performance results.

H.1 Data Encoding Procedure

```
# Input: D (MxLxN time-series dataset)
# Output: encoded vector of shape (7^2)K, where K is the number of meta-features for encoding time-series datasets using 7 aggregation operations.

def encode_time_series_dataset(D):
    # Step 1: Extract meta-features using TSFEL
    features = []  # list to store features for each sample
    for i in range(M):  # for each sample
        sample_features = []
        for j in range(N):  # for each variable
            ts = D[i][:][j]  # shape: L
            cfg = tsfel.get_features_by_domain(['temporal', 'statistical']) # Extracts the temporal and statistical feature sets.
            meta_feats = tsfel.time_series_feature_extractor(cfg, ts) # shape: K
            sample_features.append(meta_feats)
        features.append(concatenate(sample_features))  # shape: NK

    # features shape: M x NK

    # Step 2: Aggregate each meta-feature across variables using 7 functions
    # Aggregation functions: mean, std, min, max, Q1, Q2, Q3
    aggregated_samples = []
    for i in range(M):
        sample_feats = reshape(features[i], (N, K))  # shape: N x K
        agg_feats = []
        for k in range(K):
            meta_k = sample_feats[:, k]  # shape: N
            agg_k = [
                mean(meta_k),
                std(meta_k),
                min(meta_k),
                max(meta_k),
                quantile(meta_k, 0.25),
                quantile(meta_k, 0.5),
                quantile(meta_k, 0.75)
            ]
            agg_feats.extend(agg_k)
        aggregated_samples.append(agg_feats)

    # aggregated_samples shape: M x 7K

    # Step 3: Aggregate across samples to get final representation
    final_encoding = []
    for k in range(7*K):
        feature_k = [sample[k] for sample in aggregated_samples]  # shape: M
        final_agg = [
            mean(feature_k),
            std(feature_k),
            min(feature_k),
            max(feature_k),
            quantile(feature_k, 0.25),
            quantile(feature_k, 0.5),
            quantile(feature_k, 0.75)
        ]
        final_encoding.extend(final_agg)

    return final_encoding   # final_encoding shape: 7 x 7K = 49K
```

Listing 1: Pseudo-code for Time-Series Meta-feature Encoding

H.2 Architecture Encoding Procedure

```
def encode_model_architecture_with_hp(model, hyperparameters):
    # Step 1: Extract computation graph using torchview
    graph = get_computation_graph(model)  # via torchview
    nodes, edges = extract_nodes_edges(graph.DiGraph)

    N = len(nodes)
    MAX_NODES = 100 # Based on preliminary analysis of maximum node size among all
        candidate architectures in our experiments

    # Build adjacency matrix of size N x N
    adj_matrix = build_adjacency_matrix(nodes, edges, size=N)

    # Pad adjacency matrix to MAX_NODES x MAX_NODES
    padded_adj = zero_pad_matrix(adj_matrix, target_size=MAX_NODES)

    # Extract operations and normalize
    operations = [min_max_normalize(node.op_type) for node in nodes]
    padded_ops = zero_pad_vector(operations, target_size=MAX_NODES)

    # Step 2: Flatten and concatenate adjacency matrix and operations
    flat_adj = flatten(padded_adj)  # Size: MAX_NODES^2
    arch_vector = concatenate(flat_adj, padded_ops)  # Size: MAX_NODES^2 + MAX_NODES

    # Step 3: Process hyperparameters
    HP_KEYS = ...  # List of all possible training and model-specific HPs according to
         Tables 5, 6, 7, 8
    K = len(HP_KEYS)
    hp_vector = []

    for hp_key in HP_KEYS:
        if hp_key in hyperparameters:
            value = min_max_normalize(hyperparameters[hp_key])
        else:
            value = -1  # Indicator for irrelevant HP
        hp_vector.append(value)

    # Step 4: Concatenate architecture vector with HP vector
    final_encoding = concatenate(arch_vector, hp_vector)
    return final_encoding
```

Listing 2: Pseudocode: One-hot encode architecture and hyperparameters

H.3 Meta-Predictor Training Procedure

```
def train_meta_predictor(benchmark_dataset):
    # Step 1: Initialize Hyperparameters
    TRAIN_SPLIT, VAL_SPLIT, TEST_SPLIT = 0.7, 0.15, 0.15 # Split sizes
    N_EPOCHS = 200 # Number of training iterations/epochs
    BS = 256 # batch size
    LR = 0.0001 # learning rate
    ES_PATIENCE = 10 # early stopping patience
    LOSS_MARGIN = 1.0 # margin value of CRL loss function
    HIDDEN_SIZE = 2048 # initial hidden dimension

    # Step 1: Split dataset
    train_set, val_set, test_set = split_dataset(benchmark_dataset, [TRAIN_SPLIT,
        VAL_SPLIT, TEST_SPLIT])
    train_loader = DataLoader(train_set, batch_size=BS, shuffle=True)
    val_loader = DataLoader(val_set, batch_size=BS)

    # Step 2: Define model
    model = Sequential(
        Linear(input_dim, HIDDEN_SIZE), ReLU(),
        Linear(HIDDEN_SIZE, HIDDEN_SIZE // 2), ReLU(),
        Linear(HIDDEN_SIZE // 2, HIDDEN_SIZE // 4), ReLU(),
        Linear(HIDDEN_SIZE // 4, 1)  # regression output
    )

    # Step 3: Define optimizer, loss, and early stopping
    optimizer = RAdam(model.parameters(), lr=0.0001)
    loss_fn = CRL(margin=LOSS_MARGIN)
    early_stopper = EarlyStopping(patience=ES_PATIENCE)

    # Step 4: Training loop
    for epoch in range(N_EPOCHS):
        model.train()
        for batch in train_loader:
            x, ah, v = batch
            input_vec = concatenate(x, ah)
            pred = model(input_vec)
            loss = loss_fn(pred, v)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        # Validation and early stopping
        val_loss = evaluate(model, val_loader, loss_fn)
        if early_stopper.step(val_loss):
            break

    # Step 5: Return trained model
    return model
```

Listing 3: Pseudocode: Training a Meta-Predictor Model

H.4 Meta-Search Procedure

```
def search_best_arch_hypers(g, D_n, D, T, H, N, K):
    """
    g: Trained meta-predictor
    D_n: Input time-series dataset for which meta-search algorithm should run
    D: All historical datasets
    T: All historical observations including dataset, arch-hyper, and corresponding
        performance value
    H: Hyperparameter search space of all candidate architectures
    N: Number of top architectures to select
    K: Number of top hyperparameter configurations per architecture
    """

    # Step 1: Find most similar dataset D_s to D_n
    D_n_encoded = encode_time_series_dataset(D_n)
    min_dist = inf
    for D_i in D:
        D_i_encoded = encode_time_series_dataset(D_i)
        dist = normalized_L1_distance(D_n_encoded, D_i_encoded)
        if dist < min_dist:
            min_dist = dist
            D_s = D_i

    # Step 2: Get top-N architectures A_top based on performance on D_s
    arch_perf = {}
    for (arch, hp, dataset, perf) in T:
        if dataset == D_s:
            arch_perf.setdefault(arch, []).append(perf)
    avg_perf = {a: mean(arch_perf[a]) for a in arch_perf}
    A_top = top_N_keys(avg_perf, N)

    # Step 3: Predict performance for each arch and its HPs using g
    results = []
    for arch in A_top:
        hp_grid = generate_hyperparam_grid(H[arch])
        for hp in hp_grid:
            input_vec = concatenate(D_n_encoded, encode_model_architecture_with_hp(
                arch, hp))
            v_bar = g.predict(input_vec)
            results.append((arch, hp, v_bar))

    # Step 4: Select top-K HPs for each top-N arch
    top_arch_hypers = {}
    for arch in A_top:
        arch_results = [(hp, v_bar) for (a, hp, v_bar) in results if a == arch]
        top_k = top_K(arch_results, K, key=lambda x: x[1])  # sort by v_bar
        top_arch_hypers[arch] = top_k

    # Step 5: Return top-N architectures with top-K HPs each
    return top_arch_hypers
```

Listing 4: Pseudocode: Searching for Best Arch-Hypers Using Meta-Predictor

References

Bagnall, A.; Dau, H. A.; Lines, J.; Flynn, M.; Large, J.; Bostrom, A. G.; Southam, P.; and Keogh, E. J. 2018. The UEA multivariate time series classification archive, 2018. arXiv:1811.00075v1.

Brazdil, P.; van Rijn, J.; Soares, C.; and Vanschoren, J. 2022. *Metalearning: Applications to Automated Machine Learning and Data Mining*. Springer.

Chen, S.-A.; Li, C.-L.; Arik, S. O.; Yoder, N. C.; and Pfister, T. 2023. TSMixer: An All-MLP Architecture for Time Series Forecast-ing. *Transactions on Machine Learning Research*.

Dao, T.; and Gu, A. 2024. Transformers are SSMs: generalized models and efficient algorithms through structured state space duality. In *Proceedings of the International Conference on Machine Learning*. JMLR.

Dau, H. A.; Bagnall, A.; Kamgar, K.; Yeh, C.-C. M.; Zhu, Y.; Gharghabi, S.; Ratanamahatana, C. A.; and Keogh, E. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*.

Donghao, L.; and Xue, W. 2024. ModernTCN: A Modern Pure Convolution Structure for General Time Series Analysis. In *Proceedings of The International Conference on Learning Representations*.

Gu, A.; and Dao, T. 2024. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *Proceedings of the First Conference on Language Modeling*.

Hasani, R. M.; Lechner, M.; Wang, T.; Chahine, M.; Amini, A.; and Rus, D. 2023. Liquid Structural State-Space Models. In *Proceedings of the International Conference on Learning Representations*.

Kitaev, N.; Kaiser, L.; and Levskaya, A. 2020. Reformer: The Efficient Transformer. In *Proceedings of the International Conference on Learning Representations*.

Lin, S.; Lin, W.; Wu, W.; Zhao, F.; Mo, R.; and Zhang, H. 2023. SegRNN: Segment Recurrent Neural Network for Long-Term Time Series Forecasting. arXiv:2308.11200v1.

Liu, Y.; Hu, T.; Zhang, H.; Wu, H.; Wang, S.; Ma, L.; and Long, M. 2024. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting. In *Proceedings of The International Conference on Learning Representations*.

Lukasik, J.; Friede, D.; Stuckenschmidt, H.; and Keuper, M. 2021. Neural Architecture Performance Prediction Using Graph Neural Networks. In Akata, Z.; Geiger, A.; and Sattler, T., eds., *Pattern Recognition*, 188–201. Springer International Publishing. ISBN 978-3-030-71278-5.

Nie, Y.; H. Nguyen, N.; Sinthong, P.; and Kalagnanam, J. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *Proceedings of the International Conference on Learning Representations*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is All you Need. In *Proceedings of Advances in Neural Information Processing Systems*.

Voelker, A.; Kajic, I.; and Eliasmith, C. 2019. Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks. In *Proceedings of the International Conference on Neural Information Processing Systems*.

White, C.; Neiswanger, W.; Nolen, S.; and Savani, Y. 2020. A Study on Encodings for Neural Architecture Search. In *Proceedings of Advances in Neural Information Processing Systems*.

Woo, G.; Liu, C.; Sahoo, D.; Kumar, A.; and Hoi, S. 2022. ETSformer: Exponential Smoothing Transformers for Time-series Forecasting. arXiv:2202.01381v2.

Wu, H.; Hu, T.; Liu, Y.; Zhou, H.; Wang, J.; and Long, M. 2023. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. In *Proceedings of the International Conference on Learning Representations*.

Wu, H.; Xu, J.; Wang, J.; and Long, M. 2021. Autoformer: decomposition transformers with auto-correlation for long-term series forecasting. In *Proceedings of the International Conference on Neural Information Processing Systems*.

Yue, Z.; Wang, Y.; Duan, J.; Yang, T.; Huang, C.; Tong, Y.; and Xu, B. 2022. TS2Vec: Towards Universal Representation of Time Series. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zeng, A.; Chen, M.; Zhang, L.; and Xu, Q. 2023. Are transformers effective for time series forecasting? In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhang, T.; Zhang, Y.; Cao, W.; Bian, J.; Yi, X.; Zheng, S.; and Li, J. 2022. Less Is More: Fast Multivariate Time Series Forecasting with Light Sampling-oriented MLP Structures. arXiv:2207.01186v1.

Zhang, Y.; and Yan, J. 2023. Crossformer: Transformer Utilizing Cross-Dimension Dependency for Multivariate Time Series Forecasting. In *Proceedings of the International Conference on Learning Representations*.

Zhou, H.; Zhang, S.; Peng, J.; Zhang, S.; Li, J.; Xiong, H.; and Zhang, W. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhou, T.; Ma, Z.; Wang, X.; Wen, Q.; Sun, L.; Yao, T.; Yin, W.; and Jin, R. 2022. FiLM: Frequency improved Legendre Memory Model for Long-term Time Series Forecasting. In *Proceedings of the International Conference on Neural Information Processing Systems*.