KDSelector: A Framework of Knowledge-Enhanced and Data-Efficient Selector Learning for Anomaly Detection Model Selection in Time Series

Zhiyu Liang[1], Dongrui Cai[1], Chenyuan Zhang[1], Zheng Liang[1], Chen Liang[1], Shi Qiu[2], Jin Wang[2], Hongzhi Wang[1*]

[1]Harbin Institute of Technology, Harbin, China, [2]Central South University, Changsha, China

{zyliang,lz20,wangzh}@hit.edu.cn, {25S103186,cyzhang,23B903050}@stu.hit.edu.cn, {sheldon.qiu,jerryW}@csu.edu.cn

ABSTRACT

Model selection has been raised as an essential problem in the area of time series anomaly detection (TSAD), because there is no single best TSAD model for highly heterogeneous time series in real-world applications. However, despite the success of existing model selection solutions, which usually learn (a.k.a. train) a classification model (especially neural network, NN) using historical data as a *selector* to predict the correct TSAD model for each time series to detect, the existing NN-based selector learning method cannot utilize the auxiliary knowledge in the historical data and requires iterating over all training samples, which limits the model selection ability and training speed of the selector. The latter data efficiency problem can be partially solved by existing data pruning methods designed for general NN training, but with suboptimal speedup or degraded selection ability due to disregarding intrinsic data properties in TSAD model selector training. To address these limitations, we propose KDSelector, to the best of our knowledge, the first framework customized for knowledge-enhanced and data-efficient learning of NN-based TSAD model selectors, of which we design three plug-and-play modules that are agnostic to NN architectures (e.g., ResNet and Transformer) and can be seamlessly integrated into the existing selector learning framework. Specifically, we propose two knowledge enhancement mechanisms to improve the selection ability of the selector with any architecture by integrating the auxiliary knowledge in a unified way. We further design a novel data pruning framework with theoretical guarantees to achieve state-of-the-art training acceleration for the NN-based selector with almost lossless selection ability. Extensive experiments demonstrate the superior performance of our proposals in terms of model selection ability and selector learning efficiency.

PVLDB Reference Format:

Zhiyu Liang, Dongrui Cai, Chenyuan Zhang, Zheng Liang, Chen Liang, Shi Qiu, Jin Wang, and Hongzhi Wang. KDSelector: A Framework of Knowledge-Enhanced and Data-Efficient Selector Learning for Anomaly Detection Model Selection in Time Series. PVLDB, 19(9): 1935-1948, 2026.

doi:10.14778/3819518.3819525

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at https://github.com/chenyuanTKCY/KDSelectorAlgorithm.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 9 ISSN 2150-8097.
doi:10.14778/3819518.3819525

1 INTRODUCTION

Extensive real-world infrastructures [16, 24, 54, 66, 75], such as power grids [16] and intelligent manufactories [66], employ sensors to record key metrics of interested entities (e.g., temperature of a machine) to monitor their status. These recordings are real-valued data points that vary over time, commonly referred to as *time series* [28, 57, 65]. Time series anomaly detection (TSAD) [44, 59, 67], that is, identifying data points that do not correspond to expected behaviors (e.g., abnormal temperature values of a thermostat system), is one of the most important problems in time-series data management that is crucial for many applications [64, 83].

Although numerous TSAD techniques have been proposed in recent years [6, 44, 59, 67, 82], multiple evaluation studies [30, 59, 67] consistently show that there is *no single best* anomaly detection method (a.k.a. model) when applied to various time series collected from different entities or domains, partially due to their highly heterogeneous nature in terms of data characteristics (e.g., stationary or non-stationary [70]), anomaly types (e.g., point- or sequence-based [59, 70]) and numbers, and pattern variations (e.g., one, multiple similar or different anomalies/normality [59]).

A straightforward solution is to combine all existing TSAD methods through ensembling [1]. However, as shown in [70], such a solution requires running all TSAD methods for each time series to detect, causing excessive computational costs that are prohibitive for real-world applications, especially for large-scale time series collections. Meanwhile, bottlenecked by a portion of inferior TSAD models of the combination, the ensemble method usually performs worse than the model that has the best detection performance.

Another line of work proposes *trial-based model selection* solutions [20] that estimate the detection performance of different TSAD models using predefined surrogate metrics and choose the model with the best estimated performance. However, it still has to run all TSAD models to compute the surrogate metrics for each time series to detect. It is also difficult for surrogate metrics to accurately evaluate performance over time series of different data properties, causing suboptimal selection (see Exp-2 in § 5.2).

To overcome the above issues, recent work [64, 70] proposes *meta-learning-based model selection methods* to adaptively select the best TSAD model for different time series. This is usually achieved by learning (a.k.a. training) a time series classification [38, 50] (TSC) model as a **selector** to classify time series into discrete categories that represent the TSAD models to select, using historical data such as the previously seen time series and the corresponding correct TSAD models as training data. By modeling the relationships between time series and TSAD models, such solutions can better identify suitable TSAD models (i.e. with high detection performance),

and only the selected model is run for the target time series, which is more scalable than running all models as above solutions.

Among existing meta-learning solutions, *neural network (NN)-based selectors* have shown superiority in selecting the correct TSAD models [70], due to their ability to automatically learn complex relationships between time series and TSAD models [33, 49], rather than relying on complicated and difficult-to-generalize feature engineering as traditional non-NN-based methods [70]. However, despite the advantages of NN-based selectors, the selector learning methods used by existing solutions face two main challenges.

Challenge 1. Following the standard TSC training scheme [38, 49], existing methods can only use the time series and the labels that represent the best TSAD models for the corresponding series as training data, *ignoring that there usually exists auxiliary knowledge in the historical data*, such as the detection performance of all TSAD model candidates used for identifying the labels [70], and the diverse metadata that reflects the characteristics of the time series, anomalies, TSAD scenarios, etc [68]. As a result, the learned selector can still be weak in choosing a good TSAD model (i.e., model selection ability). Considering that the available knowledge and selector architecture (e.g., ResNet [21] or Transformer [73]) can be different across applications, it is difficult yet essential to effectively and efficiently utilize the heterogeneous knowledge in a unified and general way to improve the selectors of different architectures.

Challenge 2. NN-based selector learning in existing solutions, with the widely used stochastic gradient descent (SGD) scheme [33], *requires iterating over all training samples, which is data-inefficient and time-consuming*. Although there exist advanced training data pruning methods [27, 60, 61, 63] that can remove less essential samples for training acceleration, current general approaches disregard the intrinsic data characteristics of TSAD model selector training. Some of them also incur non-negligible computational costs for the pruning and can degrade the performance of the NNs. These issues cause the existing methods to provide *limited training speedup and suboptimal selection ability* (see Exp-3 in § 5.2).

To address these challenges, we propose KDSelector [36], to the best of our knowledge, the first framework for improving selection ability and learning efficiency of NN-based TSAD model selectors via knowledge-enhanced and data-efficient selector learning.

Addressing challenge 1. We propose two knowledge enhancement modules to integrate the aforementioned knowledge into the selector to improve its selection ability.

To utilize the detection performance of all TSAD models, we propose a simple yet effective *performance-informed selector learning* (PISL) module. It transforms the performance scores of different TSAD models into the probabilities of selecting the models, which is taken as a soft target to better guide the learning of the selector.

In parallel, we propose a unified *meta-knowledge integration* (MKI) framework to gain knowledge from various metadata. Specifically, to flexibly incorporate different types of metadata, MKI first uses a simple yet general text template to describe metadata in natural language. On this basis, to understand the knowledge within the metadata and integrate it into the selector, MKI transforms the text into unified embedding (i.e., feature vector) via a pre-trained language model (PLM) [17, 53], taking advantage of the powerful language understanding capability of the PLM. Then, it introduces a novel objective for selector learning, by encouraging the mutual information between the text embedding and the feature vector of the corresponding time series extracted by the selector to be maximized, which ensures that the learned selector can share the auxiliary information (i.e., knowledge) of the metadata.

Addressing challenge 2. We propose a novel framework for accelerating the training of NN-based selectors, named *similarity and difficulty guided pruning* (SDGP), which can efficiently prune more inessential training samples at each epoch with nearly lossless selection ability. Our key observation is that there are training samples that are very similar to each other and also have almost equal learning difficulty. Based on our theoretical analysis, such samples may have redundant information for selector learning. Therefore, we randomly prune them and rescale the gradients of the remaining samples, which not only improves the training speed, but also theoretically guarantees that training on the pruned dataset can achieve a similar result as training on the original one.

It should be noted that the three proposed key components, including PISL, MKI and SDGP, are all *plug-and-play frameworks* and *agnostic to NN architectures*. This means that users can flexibly integrate them into different TSAD model selection applications.

Contributions. We make the following notable contributions.

- We propose KDSelector, to the best of our knowledge, the first framework for knowledge-enhanced and data-efficient learning of NN-based TSAD model selectors (§ 2.3).
- We design PISL and MKI, two plug-and-play and architecture-agnostic modules for selector learning, that make use of the auxiliary knowledge in historical data in a unified and general way to improve the model selection ability (§ 3).
- We design SDGP, a novel data pruning framework with theoretical guarantees to remove redundant samples to achieve data-efficient selector learning and almost lossless model selection ability (§ 4).
- Our extensive experiments using various types of datasets demonstrate the superior performance of KDSelector in terms of model selection ability and selector learning efficiency (§ 5).

2 PRELIMINARIES AND OVERVIEW

2.1 Notations and Problem Formulation

Time series. A time series is a sequence of values ordered by time, denoted $T = (t_1, t_2, \ldots, t_n)$, where $n = |T|$ is the length of T. Denote the l-length subsequence of T that starts from position i as $T_{i,l} \in \mathbb{R}^l$, i.e., $T_{i,l} = (t_i, t_{i+1}, \ldots, t_{i+l-1})$, where $i \in [1, n-l+1]$.

TSAD model and anomaly score. Given an input time series T, a TSAD model, denoted by M, can be uniformly defined [70] as a function that maps from T to a sequence of length $|T|$, i.e., $M(T) = (s_1, s_2, \ldots, s_n)$, where $s_i \in [0, 1]$ represents the anomaly score of t_i (a higher score indicates more anomalous).

Detection performance. Let $L = (L_1, L_2, \ldots, L_n)$ be the ground truth (a.k.a. label) of T, where $L_i = 0$ if t_i is a normal point and 1 otherwise. The detection performance of a TSAD model M is referred to as a function $P(M(T), L)$ that indicates how accurate M is when applied to T and accordingly to the ground truth L, i.e., how close s_i is to L_i for all $i \in [1, n]$ (the closer, the better). Standard metrics used for evaluating the detection performance include the area under the precision-recall curve (AUC-PR) [12], volume under the surface (VUS) [58], F1-score [68], etc. Since the label L depends only on T, we abbreviate $P(M(T), L)$ to $P(M(T))$ for explanation.

TSAD model selection. Given the above notations, we can formally define the problem of TSAD model selection as follows.

Definition 2.1 (TSAD model selection)**.** *Given a TSAD model set, denoted as $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$, where $m = |\mathcal{M}|$ is the number of*

TSAD model candidates, the goal of TSAD model selection is to build a function f to determine the model in $\mathcal{M}$ that has the best detection performance for an input time series T, i.e.,

$$f(T) = \arg\max_{i=1,\ldots,m} P(M_i(T)), \tag{1}$$

where P can be any interested metric, such as the standard AUC-PR or a customized metric that combines many key performance indicators.

Note that *the ground-truth anomalies (a.k.a. the label L) are unknown for a series T to detect, which means that it is impossible to accurately evaluate $P(M_i(T))$*. Thus, the key to solving the problem in Definition 2.1 is to design a function f that can select the TSAD model with the detection performance as good as possible. In general, f can be built in two different ways, by trial or by meta-learning. The former uses surrogate metrics to estimate $P(M_i(T))$ for each candidate M_i and chooses the one with the best estimated performance, while the latter builds f with a selector trained using historical data and directly predicts the best model for an input T.

This study focuses on meta-learning solutions due to their advantages discussed in § 1. The general framework of such methods is presented in § 2.2, which serves as the basis of our KDSelector.

2.2 Meta-learning-based TSAD Model Selection

The basic idea of meta-learning-based methods is to learn a predictive function f from historical data to directly predict the correct TSAD model to be chosen for a time series T, avoiding computationally intensive execution of all candidates on T as in ensembling and trial-based approaches. This corresponds to a specific TSC problem, where f is a TSC model (referred to as a **selector**) that classifies T into a class $i \in \{1, 2, \ldots, m\}$, representing selecting the model M_i from the set of candidates $\mathcal{M}$. Thus, given available historical data, mainly including a collection of time series (denoted by $\mathcal{T}$) and the corresponding anomaly labels, the selector f can be learned using many existing TSC approaches, by identifying the class label y that represents the correct TSAD model to select for $T \in \mathcal{T}$, i.e., $y = \arg\max_{i=1,\ldots,m} P(M_i(T))$, and then using the preprocessed labeled time series $\{(T, y) | T \in \mathcal{T}\}$ as a training set to train f.

However, since the series T in TSAD applications usually have variable lengths (i.e., $|T|$ is not a constant) and can be very long, building f on top of the original time series as mentioned above is very limited, because not all TSC methods can handle variable-length input and some have poor scalability *w.r.t.* the length [2], and batches of time series of variable length cannot be treated in parallel for acceleration by hardware like GPUs.

To address these limitations, Sylligardos et al. [70] proposes a general TSAD model selection pipeline, where the original time series are preprocessed into equi-length subsequences for training f. Specifically, for a given length l, each original time series T of length $|T|$ ($|T| \geq l$) is divided into a set of $\lceil |T|/l \rceil$ non-overlapping subsequences of length l, denoted as $\mathcal{T}_l$. In particular, when the length of T cannot be divided evenly with the length l, the remainder, denoted $\Delta = \lceil \frac{|T|}{l} \rceil \cdot l - |T|$, is added with an overlap between the first two subsequences. Formally, $\mathcal{T}_l$ is defined as follows.

$$\mathcal{T}_l = \begin{cases} \left\{ T_{1+i \cdot l,\, l},\ i \in \left\{0, 1, \ldots, \left\lceil \frac{|T|}{l} \right\rceil - 1\right\}\right\}, & \left\lceil \frac{|T|}{l} \right\rceil = \frac{|T|}{l}, \\ \left\{ T_{1,l} \right\} \cup \left\{ T_{1+l-\Delta+i \cdot l,\, l},\ i \in \left\{0, 1, \ldots, \left\lceil \frac{|T|}{l} \right\rceil - 2\right\}\right\}, & \left\lceil \frac{|T|}{l} \right\rceil > \frac{|T|}{l}. \end{cases} \tag{2}$$

Since all subsequences in $\mathcal{T}_l$ originate from the time series T, they share the same label $y = \arg\max_{i=1,\ldots,m} P(M_i(T))$ as T. All historical time series in $\mathcal{T}$ are divided and labeled in the same way above to construct the training set. Then, any TSC approach can be used to train a selector f over the training subsequences and class labels. In the inference (i.e., model selection) stage, the learned f predicts which TSAD model to use for each subsequence $T_{i,l} \in \mathcal{T}_l$ of a given time series T, and majority voting is used to select a model to run on T from all the predictions, as visually illustrated in Fig. 1.

For ease of explanation, in the remainder of this paper, we abbreviate a subsequence $T_{i,l}$ as T_i, denote its class label and the time series it originates from as y_i and T, respectively. And we denote the full dataset that contains all training subsequences by $\mathcal{D}$.

2.3 Overview of KDSelector

Motivation. While the meta-learning method in § 2.2 is a general framework for which any TSC model can be used as a selector, this paper focuses on NN-based selectors due to their superior model selection ability as discussed in § 1 and verified in Exp-2 of § 5.2.

Generally, *an NN-based selector f consists of two learnable components, a time series encoder E_T appended by a linear classifier C_T*. Given an input subsequence T_i, the selector first transforms it into a feature vector $\boldsymbol{z}_i^T \in \mathbb{R}^{D_T}$ as

$$\boldsymbol{z}_i^T = E_T(T_i). \tag{3}$$

Then, the selector maps $\boldsymbol{z}_i^T$ to a vector $\hat{\boldsymbol{p}}_i \in \mathbb{R}^m$ that represents the predicted probabilities of selecting different TSAD models, i.e.,

$$\hat{\boldsymbol{p}}_i = C_T(\boldsymbol{z}_i^T) = (Pr(M_1), \ldots, Pr(M_m)), \tag{4}$$

where $\hat{\boldsymbol{p}}_{i,j} = Pr(M_j)$ denotes the probability of selecting the candidate M_j with $\hat{\boldsymbol{p}}_{i,j} \in [0, 1]$ and $\sum_{j=1}^{m} \hat{\boldsymbol{p}}_{i,j} = 1$, and the model with the highest predicted probability is chosen by f for T_i, i.e.,

$$f(T_i) = \arg\max_{j=1,\ldots,m} \hat{\boldsymbol{p}}_{i,j}. \tag{5}$$

However, despite the advantages of NN-based selectors, the *standard selector learning framework used by existing TSAD model selection approaches* (as Fig. 2 shows), which follows the general training procedure of a TSC model [70], only uses the time series T_i and the label y_i that represents the model of the best performance for T_i (referred to as the "hard label") to learn f. This is typically achieved by iteratively adjusting the parameters of f using SGD to minimize a classification loss between each prediction $\hat{\boldsymbol{p}}_i$ and the hard label y_i, e.g., the commonly used cross-entropy loss defined as $\mathcal{L}_i^{CE} = -\log \hat{\boldsymbol{p}}_{i,y_i}$. Meanwhile, it requires iterating over all training samples $T_i \in \mathcal{D}$ at each epoch (see Algorithm 1). As discussed in § 1, these issues limit the performance of existing NN-based selectors.

Our proposal. To address the aforementioned limitations, our KDSelector introduces three independent and plug-and-play modules that can be seamlessly integrated into the standard NN-based selector learning procedure, including PISL and MKI that make use of the auxiliary knowledge in historical data in a unified way to improve the selection ability, and SDGP that dynamically prunes less important samples during training with theoretical guarantees to achieve state-of-the-art (SOTA) speedup with nearly lossless selection ability. The fully decoupled and modular design of KDSelector allows flexible composition of every component to generalize to diverse TSAD applications that have specified available knowledge (i.e., performance scores or metadata) and request (e.g., precision- or efficiency-first). Although we adopt existing techniques as building blocks, e.g., PLM-based embedding, we make notable contributions in customizing the first knowledge-enhanced and data-efficient framework for NN-based selector learning, based on our novel and essential insights. We visually illustrate the KDSelector framework in Fig. 2, and detail our novel technical designs in § 3 and § 4.

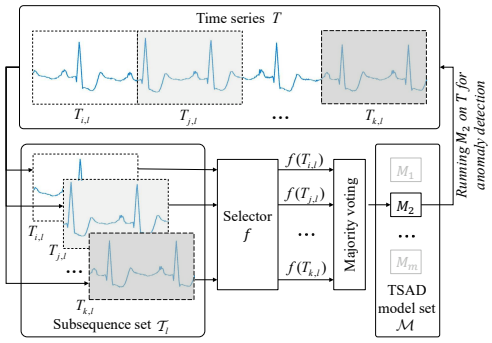


Figure 1: Overall pipeline of meta-learning-based TSAD model selection (Inference stage).

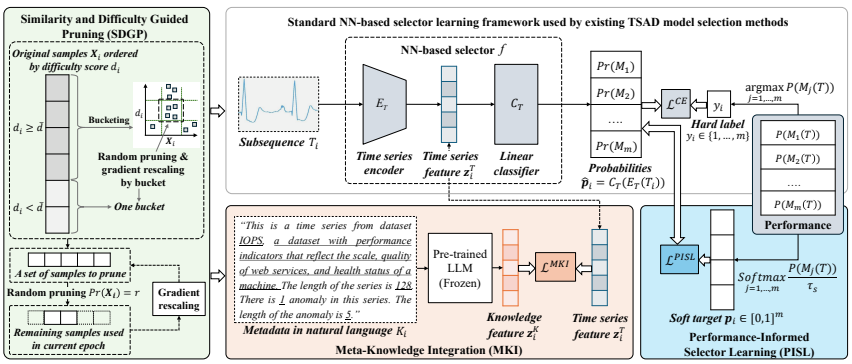


Figure 2: Framework overview of KDSelector (Training stage).

3 KNOWLEDGE ENHANCEMENT: PISL & MKI

As mentioned in § 1, there are generally two types of auxiliary knowledge in historical data that cannot be utilized by existing meta-learning-based TSAD model selection methods following the standard selector learning framework presented in § 2.3, including (1) the detection performance of all TSAD model candidates used for identifying the hard label, i.e., $P(M_1(T)), \ldots, P(M_m(T)), \forall T \in \mathcal{T}$, and (2) the various metadata reflecting the characteristics of the time series, anomalies, TSAD scenarios, etc (detailed in § 3.2). Next, we elaborate on how KDSelector integrates these two types of knowledge into NN-based selectors of any architecture via the proposed PISL and MKI modules, in § 3.1 and § 3.2, respectively.

3.1 Performance-Informed Selector Learning

Observation. In principle, the performance scores $P(M_j(T)), j = 1, 2, \ldots, m, \forall T \in \mathcal{T}$ consist of rich information about how different models perform and how their detection performance is related to each other on the historical time series, much more than just the best TSAD model as indicated by the hard label. Such information can be beneficial for selectors in distinguishing different model candidates, providing knowledge on how to make a better choice.

For example, using the hard label $y = \arg\max_{j=1,\ldots,m} P(M_j(T))$ for selector learning (e.g., by minimizing $\mathcal{L}_i^{CE}$ where $y_i = y$) is equivalent to encourage the probability of selecting the model M_j $(j = y_i)$ to be 1 and the probabilities for all other candidates M_j $(j \neq y_i)$ to be 0, meaning that the TSAD models other than the best one are indistinguishable for the selectors. However, these models are almost impossible to be equivalent because their performance scores on each time series are usually different (see Exp-7 in § 5.2). For this reason, models with better detection performance should be more likely to choose than those perform worse.

Methodology. Based on our observation, we propose a simple yet effective performance-informed selector learning (PISL) module to make use of the detection performance. The *core idea* is to transform the performance scores into probabilities of selecting the corresponding TSAD models for each time series, and use the probabilities as a soft target to enhance the learning of the selectors.

Specifically, we use a vector $\boldsymbol{p}_i \in \mathbb{R}^m$ to indicate the probabilities of selecting different models for the input subsequence T_i. To ensure comparability with the predicted probabilities of the selector f and correlation to the detection performance, the vector should satisfy:

(1) *Probabilistic:* Each entry of $\boldsymbol{p}_i$, i.e., $\boldsymbol{p}_{i,j}, \forall j$, indicates the probability of choosing M_j, satisfying $\boldsymbol{p}_{i,j} \in [0, 1]$ and $\sum_{j=1}^m \boldsymbol{p}_{i,j} = 1$.

(2) *Performance-informed:* The probability of selecting M_j for T is proportional to the performance of M_j, i.e., $\boldsymbol{p}_{i,j} \propto P(M_j(T))$.

To satisfy the above conditions, inspired by knowledge distillation [22], we define $\boldsymbol{p}_i$ as the softmax of the performance scores smoothed by a constant τ_s, formulated as

$$\boldsymbol{p}_{i,j} = \frac{\exp\left(P(M_j(T)) \,/\, \tau_s\right)}{\sum_{j=1}^m \exp\left(P(M_j(T)) \,/\, \tau_s\right)}, \; j = 1, 2, \ldots, m. \tag{6}$$

Note that $\boldsymbol{p}_i$ can be computed in conjunction with y_i during preprocessing with negligible costs (see Exp-1 in § 5.2).

To guide the learning of a selector f using the soft target $\boldsymbol{p}_i$, we define the objective of PISL as minimizing the cross-entropy between the probabilities predicted by f (i.e., $\hat{\boldsymbol{p}}_i$) and target $\boldsymbol{p}_i$, i.e.,

$$\mathcal{L}_i^{PISL} = -\sum_{j=1}^{m} \boldsymbol{p}_{i,j} \log \hat{\boldsymbol{p}}_{i,j}. \tag{7}$$

PISL can be seamlessly integrated into existing NN-based selector learning methods, by optimizing the objective $(1-\alpha)\mathcal{L}_i^{CE} + \alpha\mathcal{L}_i^{PISL}$ for each input T_i, where α is a constant that controls the relative importance of the soft target $\boldsymbol{p}_i$ and the hard label y_i.

3.2 Meta-Knowledge Integration

Motivation. Generally speaking, there are many different types of metadata of the historical time series that can be informative for model selection, including their data characteristics such as the length and statistics of the series, their anomaly characteristics such as the number, length, and type (e.g., an abnormal point or an anomalous subsequence) of the anomalies, and the application scenarios (e.g., monitoring a server or recognizing a human's activity) the time series come from. It can further improve the model selection ability by integrating the related knowledge within the metadata into the selectors. Unfortunately, this essential idea has never been explored by existing TSAD model selection approaches.

Challenges. However, integrating knowledge within metadata into selectors is still challenging due to the following reasons.

(1) *Heterogeneity of metadata.* The type of metadata can vary significantly for different historical TSAD tasks. For example, metadata for certain tasks may contain detailed descriptions of root causes and underlying mechanisms of anomalies, e.g., memory overflow, disk failure, and network congestion that lead to abnormal crashes of a database, while in other tasks, only basic statistics of the anomalies can be obtained (e.g., derived from anomaly labels), such as their frequency of occurrence and duration. Obviously, it is infeasible to design specific methods for different types of metadata.

(2) *The need of universality.* A natural thought of knowledge integration is to expand the selector f (i.e., add additional NN blocks to f) to allow a sample consisting of both a subsequence T_i and its metadata as input, so that the selector can learn combined

features from time series and metadata for predicting a correct model. However, expanding selectors of different architectures may need customized approaches, and the added blocks and parameters will also increase the model complexity of the selectors. Moreover, unlike in historical data, metadata may be *unavailable* in the online model selection stage for a new TSAD task, thus the selector must make a choice *solely* based on the time series to detect.

Solution. To address the above challenges, the proposed MKI module is a general framework that can integrate the knowledge of various metadata into selectors of any architecture in a unified way. Our *core idea* is to first represent the knowledge from different types of metadata as unified embedding (i.e., knowledge features) by making use of the flexibility of natural language and the powerful language understanding ability of a PLM. Then, we propose a novel MKI objective to encourage semantic consistency between the time series feature extracted by any selector f and the knowledge feature corresponding to the series by maximizing their mutual information, so that f *can learn information relative to the knowledge feature without modifying its model structure and choose the TSAD models* ***solely*** *based on the series to detect (i.e.,* ***without any metadata****) at the inference stage.* We detail the method below.

First, to enable flexible representation of various metadata, we use a simple yet general text template to describe metadata in natural language. Specifically, the text for the metadata relative to each input T_i, denoted as K_i, consists of sentences that describe key properties related to model selection and their values derived from the metadata. Considering the most basic properties available in the metadata of existing TSAD datasets [59], including the name and description of the dataset that correlate with the application scenario, the length of T_i and the number and lengths of anomalies that occur in T_i (which can be easily derived from the anomaly label L), we use the following template by default in MKI, but one can easily extend the template to incorporate more useful information.

Definition 3.1 (Text template for metadata description)**.** *The text for describing the metadata of T_i is defined as:*

K_i = "This is a time series from dataset [Dataset name of T_i], [Dataset description]. The length of the series is [Length of T_i]. There are [Number of anomalies in T_i] anomalies in this series. (The lengths of the anomalies are [Lengths of the anomalies in T_i].)"

Note that the placeholder "[]" denotes the required fields, while the sentence in "()" is optional and exists only when the number of anomalies is greater than 0. In this paper, we mainly use the TSB-UAD datasets [59] for evaluation, where the *description in a short paragraph is available for every training set*, as shown in Table 4 in Appendix B [55]. We show an example of K_i for a subsequence T_i from the IOPS dataset [59] in the dashed box of MKI in Fig. 2.

Next, we adopt a PLM to encode K_i into unified embedding $\boldsymbol{z}_i^K \in \mathbb{R}^{D_K}$, denoted as $\boldsymbol{z}_i^K = PLM(K_i)$, which are the knowledge features extracted from the metadata by taking advantage of the power of PLMs in natural language understanding. To integrate the extracted knowledge into a selector f while agnostic to its architecture, from the perspective of information theory [74], we design a learning objective to encourage the maximization of mutual information (MI) between the features of the time series and the metadata. It is achieved by mapping $\boldsymbol{z}_i^T$ and $\boldsymbol{z}_i^K$ into a shared space $\mathbb{R}^H$ using two learnable projections h_T and h_K, respectively, and then minimizing the InfoNCE loss [42] that represents the opposite of a lower bound of MI between two random variables [78]. The MKI objective is formally defined as follows.

$$\mathcal{L}_i^{MKI} = -\log \frac{\exp\left(\text{sim}(h_T(\boldsymbol{z}_i^T), h_K(\boldsymbol{z}_i^K))/\tau\right)}{\sum_{T_j \in \mathcal{D}} \exp\left(\text{sim}(h_T(\boldsymbol{z}_i^T), h_T(\boldsymbol{z}_j^T))/\tau\right)}, \tag{8}$$

where $\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u} \cdot \boldsymbol{v} / (||\boldsymbol{u}|| \cdot ||\boldsymbol{v}||)$ is the cosine similarity, and τ is a constant controlling the strength of penalties on hard samples [56].

Similarly, MKI is also a plug-and-play module that can be seamlessly integrated into existing NN-based selector learning methods, by optimizing the objective $\mathcal{L}_i^{CE} + \lambda \mathcal{L}_i^{MKI}$ for each input T_i, where $\lambda > 0$ is a constant that controls the importance of MKI. Note that the PLM parameters are *frozen* (a.k.a. without update) during training, while the projection h_K can be seen as an *adapter* (i.e., a lightweight model with trainable parameters added to the PLM) [48] to adaptively extend the PLM to the selector learning task. Thus, we encode all the text K_i in the preprocessing stage and just use the embedding $\boldsymbol{z}_i^K$ for training, which not only significantly reduces training overhead, but also avoids tuning the well-pretrained PLM that may degrade its performance (see Exp-4 in § 5.2).

It is also noteworthy that metadata in industrial applications may be of low quality, limiting the effectiveness of MKI. Although it is beyond the scope of this paper to study how to evaluate and improve metadata quality, our ablation study in Exp-4 in § 5.2 shows that MKI can still improve selection ability even using incomplete K_i with missing fields. Moreover, benefiting from the plug-and-play design, we can easily make a fallback by disabling MKI when deploying KDSelector in cases where metadata quality is undesirable.

3.3 Summary and Complexity Analysis

Summary. Based on the discussion in § 3.1 and § 3.2, the objective of selector learning with the proposed PISL and MKI modules is to minimize the loss $\mathcal{L}_i$ defined in Eq. (9) for each subsequence $T_i \in \mathcal{D}$, which is achieved by using the popular stochastic gradient descent (SGD) and back-propagation algorithm [33] to update the trainable parameters of the selector f and the projections h_T and h_K (denoted as $\boldsymbol{\Theta}_f$, $\boldsymbol{\Theta}_{h_T}$ and $\boldsymbol{\Theta}_{h_K}$, respectively) over each minibatch.

$$\mathcal{L}_i = (1-\alpha)\mathcal{L}_i^{CE} + \alpha \mathcal{L}_i^{PISL} + \lambda \mathcal{L}_i^{MKI}. \tag{9}$$

The main learning procedure is illustrated in lines 1-4 and 14-26 of Algorithm 1. For ease of explanation, we denote a training sample as $\boldsymbol{X}_i$ and the full training set as $\mathcal{X}$, where $\boldsymbol{X}_i = [T_i, \boldsymbol{z}_i^K]$ if MKI is used (lines 1-2) and $\boldsymbol{X}_i = T_i$ otherwise (lines 3-4). For each minibatch, the algorithm first computes the total loss $\mathcal{L}$ over the samples in the minibatch based on Eq. (9) (lines 19, 20-21 and 22-23), where c_i is a rescaling factor for the loss of the sample $\boldsymbol{X}_i$ (i.e., $\mathcal{L}_i$) used in SDGP that we will discuss in § 4. To unify the expression, it is set to 1 for all samples when SDGP is not enabled (lines 13-15). Next, the algorithm updates the parameters with the gradients of $\mathcal{L}$ *w.r.t.* the parameters using SGD (lines 24-25 and 26).

After training, *only the learned selector f is used for inference (i.e., model selection)* following the method discussed in § 2.2, regardless of whether PISL and MKI are enabled during training.

Complexity analysis. The time complexity of the standard selector learning method without PISL and MKI depends on the architecture of f. Suppose that the time to compute $f(T_i)$ for one time is T_f. Since the model set size $|\mathcal{M}|=m$ can be seen as a constant, the standard algorithm takes $O(E|\mathcal{X}|T_f)$ time to learn f, where E represents the number of training epochs. The additional time caused by PISL is to compute $\mathcal{L}_i^{PISL}$ for each sample, which requires $O(1)$ time. Thus, the learning algorithm with PISL still takes $O(E|\mathcal{X}|T_f)$ time. The selector learning time with MKI is $O\big(E|\mathcal{X}|(T_f + T_h)\big)$, where

T_h is the time to compute $h_T(\boldsymbol{z}_i^T)$ and $h_K(\boldsymbol{z}_i^K)$ for one time. Using two lightweight models for h_T and h_K, the time incurred by MKI is almost negligible (see Exp-4 in § 5.2).

4 DATA-EFFICIENT LEARNING: SDGP

This section elaborates on SDGP, our novel similarity and difficulty guided pruning framework with theoretical guarantees for NN-based selector learning, which aims to achieve SOTA training acceleration without sacrificing selection ability. We present our theoretical observation in § 4.1 and detail our SDGP method in § 4.2. Finally, a theoretical analysis is performed in § 4.3.

4.1 Redundancy of Training Samples

Our key observation behind SDGP is that there are training samples that can be similar in themselves and also in their difficulty (usually indicated by training losses of the samples where higher values suggest harder to learn [4]) in the TSAD model selector learning problem. The main reason is that the training samples contain subsequences generated by dividing the original time series (see § 2.2), of which subsequences may represent repeated patterns (e.g., motifs [72] and repeated anomalies [59]), which can lead to similarity in terms of samples and difficulty. In the following, we formally show that such training samples may have redundant contributions to parameter gradients used by SGD for selector learning, which serves as the basis for our pruning method.

Let $\mathcal{L}_i = \mathcal{L}(F(\boldsymbol{X_i}; \boldsymbol{\Theta}), Y_i)$ be the loss of $\boldsymbol{X_i}$ in the current epoch, where $F(\boldsymbol{X}; \boldsymbol{\Theta})$ is the full model including the selector f and the projections h_T and h_K, $\boldsymbol{Y}_i$ is the one-hot encoding of the label y_i (concatenated by $\boldsymbol{p}_i$ if PISL is enabled), and $\boldsymbol{\Theta}$ is the set of all trainable parameters. Recall that the NN-based selector is learned using SGD, by which the learning result (i.e., update of the parameters) depends on the gradient of $\mathcal{L}_i$ *w.r.t.* $\boldsymbol{\Theta}$, denoted as $\nabla_{\boldsymbol{\Theta}} \mathcal{L}_i$. Abbreviate $F(\boldsymbol{X}_i; \boldsymbol{\Theta})$ as F_i. According to the chain rule, we have

$$\nabla \mathcal{L}_i = \nabla_F \mathcal{L}_i \cdot \nabla_{\Theta} F_i. \tag{10}$$

Using SGD, we can make a practical assumption that the gradient in Eq. (10) is bounded (e.g., by gradient clipping), i.e., $||\nabla_F \mathcal{L}_i|| \leq B_L$ and $||\nabla_{\Theta} F_i|| \leq B_F$, where B_L and B_F are finite constants. We also make a commonly used assumption that the functions F, $\mathcal{L}$ and the derivative $\nabla_{\boldsymbol{\Theta}} F$, $\nabla_F \mathcal{L}$ are *Lipschitz*. Then, we have Theorem 4.1.

Theorem 4.1. *Suppose $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ are two training samples that are similar in themselves and in their difficulty (i.e., losses), denoted as $||\boldsymbol{X}_i - \boldsymbol{X}_j|| < \delta_X$ and $|\mathcal{L}_i - \mathcal{L}_j| < \delta_L$, where $\delta_X > 0$ and $\delta_L > 0$ are two small values. The difference in the contributions of the samples to the selector learning, denoted by $\delta_\nabla = ||\nabla_{\boldsymbol{\Theta}} \mathcal{L}_i - \nabla_{\boldsymbol{\Theta}} \mathcal{L}_j||$, satisfies*

$$\begin{aligned} \delta_\nabla < B_F M_Y \Big(2 \cdot \mathbb{I}_{\Delta_{XY} > \delta_L} + \mathbb{I}_{\Delta_{XY} \leq \delta_L} \cdot \min(2, \frac{\delta_L - L_F C_F \delta_X}{L_Y}) \Big) \\ + \Big(B_F M_F C_F + B_L M_X \Big) \delta_X, \end{aligned} \tag{11}$$

where $\Delta_{XY} = L_Y ||\boldsymbol{Y}_i - \boldsymbol{Y}_j|| + L_F C_F \delta_X$, $\mathbb{I}_s$ is an indicator with $\mathbb{I}_s = 1$ if the statement s is satisfied and 0 otherwise. C_F, M_F, M_Y, M_X, L_F, L_Y are the Lipschitz constants of F w.r.t. $\boldsymbol{X}$, $\nabla_F \mathcal{L}$ w.r.t. F and $\boldsymbol{Y}$, $\nabla_{\boldsymbol{\Theta}} F$ w.r.t. $\boldsymbol{X}$, $\mathcal{L}$ w.r.t. F and $\boldsymbol{Y}$, respectively.

Please refer to Appendix A.1 [55] for the proof of Theorem 4.1.

Theorem 4.1 indicates that with appropriate δ_X and δ_L, the difference in the contribution to selector learning can be minimal. In particular, when $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ originate from similar domains or entities that can make $||\boldsymbol{Y}_i - \boldsymbol{Y}_j||$ arbitrarily small, we have $\delta_\nabla \to 0$ as $\delta_X \to 0$ and $\delta_L \to 0$. It provides a formal intuition that we can identify redundant samples for selector learning based on their δ_X and δ_L. Next, in § 4.2, we clarify how we take advantage of the redundancy of training samples to accelerate selector learning.

Algorithm 1: KDSelector Framework

Input: Training subsequences $\mathcal{D}$, class label set $\{y_i | T_i \in \mathcal{D}\}$, parameters of selector $\boldsymbol{\Theta}_f$, learning rate η, batch size B; soft target set $\{\boldsymbol{p}_i | T_i \in \mathcal{D}\}$ and importance coefficient α if PISL is enabled; knowledge feature set $\{\boldsymbol{z}_i^K | T_i \in \mathcal{D}\}$, parameters of projections $\boldsymbol{\Theta}_{h_T}$, $\boldsymbol{\Theta}_{h_K}$, temperature τ and importance coefficient λ if MKI is enabled; pruning ratio r, number of LSH bits q and number of bins w if SDGP is enabled.

Output: Learned parameters $\boldsymbol{\Theta}_f$ of selector f.

```
if MKI is enabled then
    X ← {X_i = [T_i, z_i^K] | T_i ∈ D};
else
    X ← D;
if SDGP is enabled then
    HashTables ← LSH(X, q);
    d ← {d_i = 1 | X_i ∈ X};
while not converged do
    /* Iterate over epochs                                   */
    if SDGP is enabled then
        S_1, S_2 ← SDGP(X, HashTables, d, r, w);
        X_train ← S_1 ∪ S_2;
        C ← {c_i = 1/(1−r) | X_i ∈ S_1} ∪ {c_i = 1 | X_i ∈ S_2};
    else
        X_train ← X;
        C ← {c_i = 1 | X_i ∈ X};
    n_b = ⌊ |X_train| / B ⌋;
    B_1, B_2, ..., B_{n_b} ← Randomly split X_train into n_b minibatches of size B;
    for B = B_1, B_2, ..., B_{n_b} do
        /* Iterate over minibatches                          */
        L ← (1/B) Σ_{T_i ∈ B} c_i L_i^CE;
        if PISL is enabled then
            L ← (1 − α) L + α (1/B) Σ_{T_i ∈ B} c_i L_i^PISL;
        if MKI is enabled then
            L ← L + λ (1/B) Σ_{T_i ∈ B} c_i L_i^MKI;
            Θ_{h_T} ← Θ_{h_T} − η ∇_{Θ_{h_T}} L;
            Θ_{h_K} ← Θ_{h_K} − η ∇_{Θ_{h_K}} L;
        Θ_f ← Θ_f − η ∇_{Θ_f} L;
        if SDGP is enabled then
            d ← (d \ {d_i | X_i ∈ B}) ∪ {L_i | X_i ∈ B};
```

4.2 Similarity and Difficulty Guided Pruning

Key idea. Based on Theorem 4.1, the *core idea* of SDGP is to divide the training samples into different buckets at the beginning of each training epoch, where the samples within each bucket are similar in themselves and in their difficulty (i.e., losses), and then perform random pruning by bucket to reduce redundancy. Moreover, note that training samples that have small loss values indicate that they have been well learned by the selector [4, 11]. Thus, we divide samples with losses less than a threshold into one bucket regardless of their similarity. To theoretically guarantee that using the remaining samples can achieve a similar result as using the full training data, we rescale the gradients of the remaining samples before updating the parameters using them. The details are as follows.

Training sample bucketing. Considering that the training samples $\boldsymbol{X}_i \in \mathcal{X}$ are invariant during selector learning while their losses vary with the update of parameters, we first partition $\mathcal{X}$ into groups of similar samples before training starts. Then, we dynamically divide the samples into different buckets at each epoch based on their losses and the groups they fall into. To avoid additional costs, instead of specifically computing the losses of all samples in

Algorithm 2: $SDGP(\mathcal{X}, HashTables, d, r, w)$

Input: Full training set $\mathcal{X}$, Hash tables $HashTables$, difficulty score set d, pruning ratio r, number of bins w.
Output: Set of remaining samples in the pruned buckets $\mathcal{S}_1$, set of samples in the buckets that do not need to prune $\mathcal{X}_2$.

1 $\bar{d} \leftarrow \frac{1}{|\mathcal{X}|} \sum_{\boldsymbol{X}_i \in \mathcal{X}} d_i$;
2 $Bin_1, Bin_2, \ldots, Bin_w \leftarrow$ Divide samples in $\{\boldsymbol{X}_i \in \mathcal{X} \mid d_i \geq \bar{d}\}$ into w equi-depth bins according to d_i;
3 $Buckets \leftarrow \emptyset$;
4 **for** $\boldsymbol{X}_i \in Bin_1 \cup Bin_2 \ldots \cup Bin_w$ **do**
5 $idx_1 \leftarrow$ Bin index of $\boldsymbol{X}_i$;
6 $idx_2 \leftarrow$ Hash table index of $\boldsymbol{X}_i$;
7 **if** $Bucket_{idx_1,idx_2} \in Buckets$ **then**
8 $Bucket_{idx_1,idx_2} \leftarrow Bucket_{idx_1,idx_2} \cup \{\boldsymbol{X}_i\}$;
9 **else**
10 $Bucket_{idx_1,idx_2} \leftarrow \{\boldsymbol{X}_i\}$;
11 $Buckets \leftarrow Buckets \cup Bucket_{idx1,idx2}$;
12 $Buckets \leftarrow Buckets \cup \{\boldsymbol{X}_i \in \mathcal{X} \mid d_i < \bar{d}\}$;
13 $\mathcal{X}_1 \leftarrow \emptyset, \mathcal{X}_2 \leftarrow \emptyset$;
14 **for** $Bucket \in Buckets$ **do**
15 **if** $|Bucket| > 1$ **then**
16 $\mathcal{X}_1 \leftarrow \mathcal{X}_1 \cup Bucket$;
17 **else**
18 $\mathcal{X}_2 \leftarrow \mathcal{X}_2 \cup Bucket$;
19 $\mathcal{S}_1 \leftarrow$ Randomly prune each $\boldsymbol{X}_i \in \mathcal{X}_1$ with probability r;
20 **return** $\mathcal{S}_1, \mathcal{X}_2$;

$\mathcal{X}$ before pruning them in each epoch, we divide them according to the loss value of each $\boldsymbol{X}_i$ *necessarily computed from the previous update that uses* $\boldsymbol{X}_i$, which we refer to as the *difficulty score* of $\boldsymbol{X}_i$, denoted as d_i. Formally, let $\mathcal{S}$ denote the pruned dataset used for learning at any epoch t. The difficulty score is defined as

$$d_i^{(t+1)} = \begin{cases} \mathcal{L}_i^{(t)}, & \boldsymbol{X}_i \in \mathcal{S}, \\ d_i^{(t)}, & \boldsymbol{X}_i \in \mathcal{X} \setminus \mathcal{S}. \end{cases} \tag{12}$$

Since the loss in the early training stage is usually unstable and cannot well reflect sample difficulty, and therefore pruning based on it can introduce bias, we initialize all scores as a small value of 1, i.e. $d_i^{(0)} = 1$. As such, no score is less than the threshold defined below. Thus, the samples are pruned based on only similarity in the first epoch, avoiding using the unreliable loss to ensure pruning robustness. The difficulty scores for the remaining samples which are essential to learn are updated on time, while the scores of the less important samples also have a probability of being updated due to the random pruning strategy discussed below. We empirically show that this scoring approach is more efficient and robust and does not sacrifice selection ability (see Exp-4 in § 5.2).

Specifically, to efficiently group $\mathcal{X}$, we use locality-sensitive hashing (LSH) [8] to map similar samples into the same hash tables. This is achieved using q random hyperplanes $\{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_q\}$ to map each sample $\boldsymbol{X}_i$ to a hash table encoded with q bits, where the j-th bit ($j = 1, 2, \ldots, q$) is 1 if $\boldsymbol{h}_j \cdot \boldsymbol{X}_i > 0$ and 0 otherwise. Next, at the beginning of each epoch, we partition training samples whose difficulty scores are no less than a threshold into w equi-depth bins [51] based on their score values d_i and divide the samples that fall into the same bin and hash table into one bucket, while the other samples (i.e., with d_i less than the threshold) are grouped into a separate bucket. Since the difficulty scores change over epoch, it is difficult to identify the easy and hard samples with a fixed threshold. Inspired by [61], we adaptively set the threshold as the average score of all samples, denoted $\bar{d} = \sum_{\boldsymbol{X}_i \in \mathcal{X}} d_i / |\mathcal{X}|$.

With the above method, we can control δ_X and δ_L of the samples in different buckets by adjusting the number of bits q and bins w.

Random pruning and gradient rescaling by bucket. After bucketing, we randomly remove redundant samples that fall into the same bucket with a given pruning ratio $r \in (0, 1)$. To be specific, each sample is pruned with a probability of r for the buckets that have more than one training sample, while the samples are retained for the buckets that have only one sample. Formally, we can represent the original set $\mathcal{X}$ using two subsets, i.e., $\mathcal{X}_1$ that combines all the buckets that need to be pruned and $\mathcal{X}_2$ otherwise, where $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2$ and $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$. Then, we formulate the pruning as

$$Pr(\boldsymbol{X}_i) = \begin{cases} r, & \boldsymbol{X}_i \in \mathcal{X}_1, \\ 0, & \boldsymbol{X}_i \in \mathcal{X}_2, \end{cases} \tag{13}$$

where $Pr(\boldsymbol{X}_i)$ denotes the probability of pruning $\boldsymbol{X}_i$. Intuitively, this *soft* pruning strategy ensures that *any* sample can be retained for training to avoid full discard of potentially important samples.

However, compared to training on the full data, directly using the remaining samples to update the selector parameters will change the overall gradient, denoted $G = \sum_{\boldsymbol{X}_i \in \mathcal{X}} \nabla_{\boldsymbol{\Theta}} \mathcal{L}_i$, which indicates the direction of parameter update that minimizes the objective $\mathcal{L}$. To be specific, assume that all samples $\boldsymbol{X}$ are drawn from a continuous distribution $\rho(\boldsymbol{X})$. The objective of selector learning on the full dataset $\mathcal{X}$ can be formulated as

$$\arg\min_{\boldsymbol{\Theta}} \mathop{\mathbb{E}}_{\boldsymbol{X} \in \mathcal{X}} [\mathcal{L}(\boldsymbol{X}; \boldsymbol{\Theta})] = \arg\min_{\boldsymbol{\Theta}} \int_{\boldsymbol{X} \in \mathcal{X}} \mathcal{L}(\boldsymbol{X}; \boldsymbol{\Theta}) \rho(\boldsymbol{X}) d\boldsymbol{X}. \tag{14}$$

However, updating the selector parameters by directly using the gradients of the remaining samples in $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{X}_2$ (where $\mathcal{S}_1 \subseteq \mathcal{X}_1$ contains the remaining samples in the pruned buckets), denoted $G_S = \sum_{\boldsymbol{X}_i \in \mathcal{S}} \nabla_{\boldsymbol{\Theta}} \mathcal{L}_i$, the learning objective becomes

$$\begin{aligned} \arg\min_{\boldsymbol{\Theta}} \mathop{\mathbb{E}}_{\boldsymbol{X} \in \mathcal{S}} [\mathcal{L}(\boldsymbol{X}; \boldsymbol{\Theta})] &= \arg\min_{\boldsymbol{\Theta}} \int_{\boldsymbol{X} \in \mathcal{X}} (1 - Pr(\boldsymbol{X})) \mathcal{L}(\boldsymbol{X}; \boldsymbol{\Theta}) \rho(\boldsymbol{X}) d\boldsymbol{X} \\ &\leq \arg\min_{\boldsymbol{\Theta}} \mathop{\mathbb{E}}_{\boldsymbol{X} \in \mathcal{X}} [\mathcal{L}(\boldsymbol{X}; \boldsymbol{\Theta})], \end{aligned} \tag{15}$$

where the equality satisfies only when $r = 0$, meaning that it is inconsistent with the original objective in Eq. (14) for a ratio $r > 0$.

To address this issue, we rescale the gradients (equivalent to rescaling the loss values) for the samples that remain in the pruned buckets by multiplying a factor $1/(1-r)$, which is formulated as

$$\nabla_{\boldsymbol{\Theta}} \mathcal{L}_i' = \begin{cases} \frac{1}{1-r} \nabla_{\boldsymbol{\Theta}} \mathcal{L}_i = \nabla_{\boldsymbol{\Theta}} \frac{1}{1-r} \mathcal{L}_i, & \boldsymbol{X}_i \in \mathcal{S}_1, \\ \nabla_{\boldsymbol{\Theta}} \mathcal{L}_i, & \boldsymbol{X}_i \in \mathcal{X}_2. \end{cases} \tag{16}$$

We theoretically show in § 4.3 that using the rescaled gradients (i.e., loss values) of samples in the pruned dataset $\mathcal{S}$, the objective is equivalent to that of using the full data.

Summary. Based on the above discussion, we show the pseudocode of SDGP, in lines 5-7, 9-12, 18-28 of Algorithm 1, and Algorithm 2.

As illustrated in Algorithm 1, all training samples are mapped into hash tables and their difficulty scores are initialized as 1 before training starts (lines 5-7). Next, the full training set $\mathcal{X}$ is pruned at the beginning of each epoch using Algorithm 2 (line 10). The remaining samples in $\mathcal{S}_1$ and $\mathcal{X}_2$ are used for training in the current epoch (line 11), where the factors are set based on Eq. (16) (line 12) to rescale the gradients (i.e., losses) of the remaining samples (lines 19-23) used for updating the parameters (lines 24-26). Finally, the computed losses are used to update the difficulty scores (line 28).

As Algorithm 2 shows, to prune training samples in $\mathcal{X}$, SDGP first divides the samples with $d_i \geq \bar{d}$ into w equi-depth bins (line 1-2) and those falling into the same bins and hash tables into one bucket (lines 4-11). Meanwhile, it divides the samples of $d_i < \bar{d}$ into an additional bucket (line 12). Then, the buckets that contain

more than one sample are combined into a set $\mathcal{X}_1$ (lines 15-16) and the other buckets are combined into another set $\mathcal{X}_2$ (lines 17-18). According to Eq. (13), only the samples in $\mathcal{X}_1$ are randomly pruned to produce $\mathcal{S}_1$ (line 19). Finally, the remaining samples in $\mathcal{S}_1$ and $\mathcal{X}_2$ are returned (line 20) to Algorithm 1 for selector training.

4.3 Theoretical Analysis

Effectiveness guarantee. Recall that in SDGP, we use the remaining samples in $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{X}_2$ to train the selector with the gradients rescaled according to Eq. (16), where $\mathcal{S}_1$ is a subset of $\mathcal{X}_1$ produced by randomly pruning each sample with probability $r \in (0, 1)$ as in Eq. (13). Thus, we have the following theorem.

Theorem 4.2. *Using the rescaled gradients* $G'_S = \sum_{X_i \in \mathcal{S}} \nabla_\Theta \mathcal{L}'_i$ *for parameter update, the objective of the selector learning, denoted* $\arg\min_\Theta \mathbb{E}_{X \in \mathcal{S}}[\mathcal{L}'(X;\Theta)]$*, satisfies the following equation for any* r*.*

$$\arg\min_\Theta \mathbb{E}_{X \in \mathcal{S}}[\mathcal{L}'(X;\Theta)] = \arg\min_\Theta \mathbb{E}_{X \in \mathcal{X}}[\mathcal{L}(X;\Theta)]. \tag{17}$$

Please refer to Appendix A.2 [55] for the proof of Theorem 4.2.

Theorem 4.2 indicates that the learning objective of using the data pruned by SDGP is always equivalent to that of using the full training set, theoretically showing the effectiveness of SDGP.

Complexity analysis. Dividing $\mathcal{X}$ using LSH takes $O(q|\mathcal{X}|D_X)$ time, linear to the number of bits q, the number of all training samples $|\mathcal{X}| = |\mathcal{D}|$ and the dimensionality of the sample D_X, and is only performed once before training starts. Algorithm 2 has a time complexity of $O(|\mathcal{X}|)$ that is run for each training epoch. Thus, the time complexity of selector learning (Algorithm 1) changes from $O\big(E|\mathcal{X}|T_{model}\big)$ to $O\big(E(|\mathcal{X}|+|\mathcal{S}|T_{model})+q|\mathcal{X}|D_X\big)$ with SDGP, where $T_{model} = T_f + T_h$ if MKI is enabled and $T_{model} = T_f$ otherwise (as discussed in § 3.3), and $|\mathcal{S}|$ is the number of remaining samples used for training. Since the training time is dominated by T_{model}, SDGP can achieve acceleration with $|\mathcal{S}| < |\mathcal{X}|$ by pruning.

5 EXPERIMENTS

5.1 Experimental Setups

Datasets. We mainly use the 16 TSB-UAD [68] datasets following the TSAD model selection benchmark [70]. The name, statistics and description used by K_i of the datasets are shown in Table 4 in Appendix B [55]. For a fair comparison, we use the recommended train/test split [70] by default for selector learning, where the training set is a combination of samples from the 16 datasets, while time series from 14 subsets are used for the test of model selection.

Baselines. We mainly evaluate our KDSelector in two aspects. For *model selection ability*, we consider three types of baseline methods:

(1) *Meta-learning-based model selection.* This includes:
- *Non-NN-based methods* extended by [70] including (i) *feature-based methods* that use the open-source tool TSFresh [10] to extract features from the input subsequences, and train classical classifiers on top of the features, containing K nearest neighbors [18] (KNN), support vector classifier [7] (SVC), AdaBoost [19], and random forest [23] (RF), and (ii) *kernel-based method* that refers to MultiRocket [71] (MR) that uses multiple convolutional kernels with a Ridge classifier.
- *NN-based methods* including (i) two representative convolution-based models, i.e., ResNet [21] (ResN) that uses convolutional layers with residual connections, and InceptionTime [26] (IncepT) that combines ResNets with kernels of multiple sizes, and (ii) an advanced Transformer (Trans) model corresponding to SiT-stem in [70]. As discussed in § 2.3, the NNs are learned using *the standard method* in existing solutions.

(2) *Trial-based model selection.* This refers to the method proposed in [20], denoted UMS, which selects TSAD models by running and ranking them based on computing surrogate metrics.

(3) *Ensembling.* The baseline solution that runs all TSAD models and combines them through average ensembling [70] (AvgEns).

For *learning efficiency* evaluation, we compare the pruning framework of KDSelector, i.e., SDGP, with the following methods.

(1) Full refers to the baseline that trains over the full dataset $\mathcal{X}$.
(2) GC [27] prunes the samples by maximizing the submodular function that considers diversity and information. The remaining samples are used for all training epochs.
(3) EL2N [60] is similar to GC, but estimates sample importance with the error-L2-norm and removes less important samples.
(4) UCB [63] prunes the samples at different training epochs using the upper confidence bound of the loss values.
(5) InfoBatch [61] prunes the samples of small loss values at random for each epoch and scales up the gradients of the remains.

We use their open-source implementations with the preferred parameters without specification for the baseline methods.

TSAD models. To achieve a fair comparison, we use the 12 representative TSAD models the same as [70] in our TSAD model set, namely *Isolation Forest (IForest), IForest1, Local Outlier Factor (LOF), Histogram-based Outlier Score (HBOS), Matrix Profile (MP), NORMA, Principal Component Analysis (PCA), Polynomial Approximation (POLY), One-class Support Vector Machines (OCSVM), LSTM-AD (LAD), CNN, and Autoencoder (AE)* (see details in [59, 70]).

Implementations. KDSelector is implemented using PyTorch 1.12. The experiment was run on a server with Platinum 8260 CPUs and Ubuntu 20.04 LTS, using a single NVIDIA GTX 3090 GPU. We use the base BERT [14] by default for text embedding with the specific [CLS] token of the last layer. The projections h_T and h_K used in MKI are implemented using two multi-layer perceptrons (MLPs), respectively. Each MLP has one hidden layer of 256 dimensions, with ReLU as the activation function. We optimize the weights α for PISL and λ for MKI as detailed in Exp-5 in § 5.2 and choose $D_K \in \{64, 256\}$, $\tau_s \in \{0.2, 0.22, 0.25\}$ through cross-validation. The temperature τ of InfoNCE is set to 0.1. For SDGP, we set the pruning ratio r to 0.8. The number of bits q used in LSH is set to 14, and the number of bins w is set to 8. Other settings are consistent with the baselines that use the same architectures for a fair comparison.

Evaluation metrics. We adopt the widely used AUC-PR [67, 70] by default to measure the detection performance of the TSAD model and the selection ability of the selector. The latter is obtained by running the selected model on the test time series and computing the metric score of the predicted anomaly scores. For evaluation of selector learning (a.k.a. training) efficiency, we report the total runtime of the training algorithm, including (i) pruning and (ii) training over the remaining data (i.e., pure training).

For a fair comparison, we exclude SDGP for selection ability evaluation because existing solutions do not use any pruning, while we keep PISL and MKI in use when comparing pruning methods.

5.2 Experimental Results

Exp-1: KDSelector vs. Standard NN-based selector learning. We compare our knowledge enhancement and data pruning modules with the standard selector learning method discussed in § 2.3

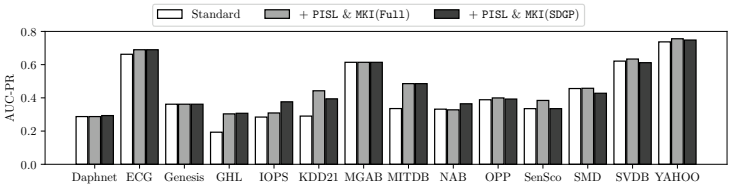
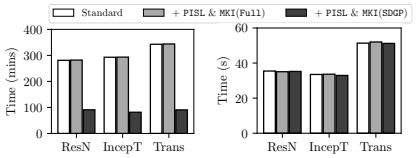


Figure 3: AUC-PR of KDSelector on different test sets. The selector architecture is ResN. Similar trends can be observed for IncepT and Trans.



(a) Selector learning. (b) Model selection.

Figure 4: Selector learning and model selection (inference) time for KDSelector using different architectures.

Table 1: Overall AUC-PR for KDSelector using different architectures.

Architecture	ResN			IncepT			Trans		
Method	Standard	+ PISL& MKI		Standard	+ PISL& MKI		Standard	+ PISL& MKI	
		Full	SDGP		Full	SDGP		Full	SDGP
Avg. AUC-PR	0.4213	**0.4609**	0.4572	0.4144	**0.4529**	0.4443	0.4353	**0.4500**	0.4402
Improved by PISL & MKI	↑ 0.0396		-	↑ 0.0385		-	↑ 0.0147		-
Reduced by SDGP	-	↓ 0.0037		-	↓ 0.0086		-	↓ 0.0098	

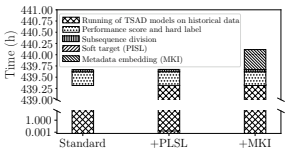


Figure 5: Preprocess time.

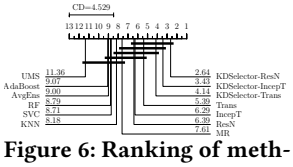


Figure 6: Ranking of methods based on AUC-PR.

for training selectors of different architectures. We show the average AUC-PR for all test series in Table 1 and the detailed results on different test sets for ResN in Fig. 3. We also show the time of selector learning and the total model selection time for all test series in Fig. 4. From the results, we can make the following observations.

(1) As shown in Table 1, learning using the knowledge enhancement modules (Full) is consistently better than the standard method for different architectures, with 0.0147-0.0396 higher on AUC-PR. Furthermore, we see from Fig. 3 that the selector learned with PISL and MKI performs better or equally in terms of AUC-PR for all test sets, except NAB where the standard method is slightly better. Meanwhile, Fig. 4 shows that the selector learning and model selection time are almost the same with and without the two modules, indicating negligible overhead incurred by them.

(2) Compared to Full, learning with SDGP does not much sacrifice and even improves model selection ability in most datasets (Fig. 3), with the average AUC-PR reduced by less than 0.01 for different architectures, still outperforms the standard method (Table 1). The decrease rate for ResN is only 0.8% (0.0037/0.4609), which is nearly lossless. However, we observe from Fig. 4**??** that using SDGP dramatically reduces the runtime of selector learning, improving the efficiency 3.1-3.8× compared to Full.

In addition, we report the preprocessing time in Fig. 5. Consistent with the analysis in § 3, PISL almost has no additional overhead compared to existing standard methods, because the performance scores used by PISL are intermediate results needed by all meta-learning baselines for identifying the best models (i.e., hard labels). The time added by MKI for metadata embedding is just 0.1%, which is also negligible. Note that while preprocessing seems time-consuming, it is needed only once and allows for reuse by all meta-learning solutions to train and improve selectors, and it can be incrementally executed for newly added data, models, and metrics.

Exp-2: Comparison of model selection methods. We conduct an end-to-end evaluation on the model selection solutions.

For comparison of model selection ability, we plot the critical difference diagram in Fig. 6 based on the Wilcoxon-Holm test [13] to show the AUC-PR ranking of the methods on the test sets, where a higher ranking indicates better performance, and the methods within different thick horizontal lines indicate that there is a significant difference between their performance with a statistical level of 0.01. From the diagram, we observe that the meta-learning-based solutions are generally better than the trial-based UMS and the ensembling method AvgEns among the baselines, especially for the NN-based selectors ResN, IncepT and Trans. Compared to baseline methods, the selectors trained using KDSelector consistently perform better. The best method, namely KDSelector-ResN, even outperforms the non-NN-based, trial-based and ensembling solutions with statistical significance, showing the superior model selection ability of our NN-based selector learning framework.

To further show the advantages of our meta-learning-based solution, we evaluate the runtime of our best method KDSelector-ResN (KD-R) and the competitors UMS and AvgEns for online inference (i.e., model selection, MS) and anomaly detection (AD). The total time and the time distribution over all test series are shown in Table 2 and Fig. 7, respectively. We can observe that both AvgEns and UMS take a long time for AD because they run all TSAD models. In addition, UMS spends a lot of time on MS by computing the surrogate metrics. Our solution is much faster because the learned selector directly predicts the correct model without time-consuming evaluation, and only the chosen model is run for detection. Note that unlike meta-learning solutions, the ensemble-based (AvgEns) and trial-based (UMS) approaches do not need offline preprocessing and selector training on the historical data. However, ensemble- and trial-based methods suffer significantly lower MS & AD efficiency as discussed above due to the exhaustive evaluation of all candidates, and thus meta-learning solutions can be more applicable because end-to-end latency of online MS and AD is usually important for a deployed TSAD service.

Note that we also compare our KD-R with the solution of running a single TSAD model on all test series. The results show that *model selection is superior to all single models on almost all test sets*. We refer the reader to Fig. 17 in Appendix C [55] for full results.

Exp-3: Comparison of data pruning methods. We evaluate the model selection ability and the training efficiency of different data pruning methods. The selector architecture is ResN (the same as in the following experiments without specification). The pruning ratio is 0.8 for all methods. The results over all test sets are shown in Table 3, from which we make the following observations.

(1) UCB, InfoBatch, and SDGP, which dynamically prune different samples during training, outperform GC and EL2N, which select a static subset of samples in terms of AUC-PR. Notably, SDGP achieves the highest AUC-PR among pruning methods, the only one that has a performance loss less than 1% compared to Full, showing effectiveness of our method in removing redundant samples.

(2) GC and EL2N are faster in pure training because they remove the given ratio of samples, but they spend a lot of time on pruning, even longer than training. In comparison, the pruning time for UCB, InfoBatch and SDGP is negligible, but their pure training time is

Table 2: Overall inference (MS) and AD time (h) of different types of solutions.

Method	AvgEns	UMS	KDSelector-ResN
MS	-	79.59	**0.01**
AD	242.01	242.01	**2.82**
Total	242.01	321.60	**2.83**
Speedup	85.52×	113.64×	-

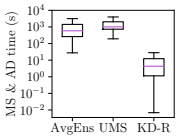


Figure 7: Time over all test series.

a few longer, since the actual ratio of samples removed is usually smaller than the given with their dynamic pruning mechanisms. SDGP, with the minimum loss of AUC-PR, is also the fastest in the total training phase, achieving 3.10× of speedup compared to Full .

We further assess the *scalability* of the pruning methods *w.r.t.* the total number of samples $|\mathcal{D}|=|\mathcal{X}|$ (default $600k$) and the subsequence length l (default 128). As Fig. 8 shows, our SDGP performs better under all sample sizes and lengths. The learning time is close for different methods when the sample size is small (e.g., $200k$), but SDGP is *increasingly more efficient as the sample size grows*, showing the significance of our SDGP for large-scale data collections.

Exp-4: Ablation study. We conduct ablation studies to evaluate the effectiveness of key designs of KDSelector. The average AUC-PR for all test sets and the selector learning time of the comparative methods are presented in Fig. 9. We discuss the results as follows.

(1) Study on loss functions. As shown in Fig. 9a *left*, we compare Full with its variants without the proposed losses of PISL, MKI, and both. We can see that both loss functions are effective in improving AUC-PR, while their impact on training speed is negligible.

(2) Study on gradient rescaling. We compare SDGP with its variant that does not rescale the gradients of the remaining samples used for training. As Fig. 9a *right* shows, the time cost of gradient rescaling is negligible, while the strategy significantly improves AUC-PR, which is consistent with our theoretical analysis in § 4.2-4.3.

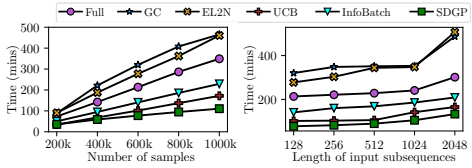
(3) Study on difficulty score. As Fig. 9a *right* shows, we replace $d_i^{(0)}$ of all samples by their loss values at the beginning of the training. The resulting variant (i.e., w/o init.) achieves significantly lower AUC-PR than SDGP, revealing the negative impact of the potentially unstable early-stage loss and thus demonstrating the effectiveness of our score initialization in improving pruning robustness. Next, we compare SDGP with its variant that does not use Eq. (12) but naïvely computes the losses of all samples per epoch to update $d_i^{(t+1)}$ (i.e., w/o update). The result shows that update as our SDGP is much more efficient without sacrificing AUC-PR.

(4) Study on PLM parameters. We make a comparison between freezing PLM parameters and fine-tuning them simultaneously during training. As Fig. 9b *left* shows, Frozen is much more efficient. It also has slightly higher AUC-PR, suggesting that fine-tuning may destroy the well-learned knowledge of the PLM.

(5) Study on PLM models. As Fig. 9b *right* shows, we replace the default BERT ($110M$) by alternative PLMs, including GPT-2 [62] ($124M$), DeepSeek [5] ($8B$) and LLama 3 [15] ($8B$). Since these models do not have the [CLS] token for text encoding as BERT, we use the mean pooling of the tokens of the last layer following LLM2Vec [3]. We observe that the default BERT outperforms the other three PLMs in terms of AUC-PR. The reason may be that BERT is encoder-only that is suitable for our classification-style tasks, while the others are mainly designed for text generation. DeepSeek and LLama 3 that are larger in parameter scales are inferior to GPT-2 and BERT, perhaps because the larger models are too general to fit the task of MKI without further adaptation (e.g., prompt engineering [43]).

Table 3: Overall results of data pruning methods.

Method		Full	GC	EL2N	UCB	InfoBatch	SDGP
Avg. AUC-PR		0.4609	0.3154	0.2877	0.4066	0.4551	**0.4572**
Time (mins)	Prune	-	351.21	291.96	0.37	**0.01**	0.51
	Train	282.02	58.48	**54.86**	124.48	173.71	90.41
	Total	282.02	409.69	346.82	124.85	173.72	**90.92**
Speedup		-	0.69×	0.81×	2.26×	1.62×	**3.10**×



(a) Varying sample size. (b) Varying input length.

Figure 8: Scalability of data pruning methods.

The selector learning time of using different PLMs is close because the metadata is embedded during preprocessing.

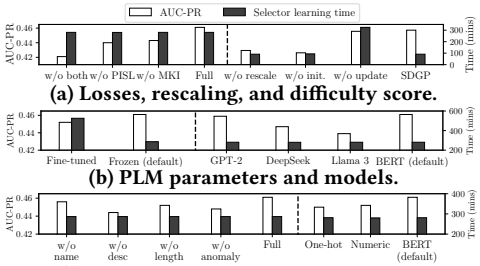
(6) Study on metadata template. We compare the metadata template K_i in definition 3.1 (Full) with its ablations without dataset name or description (desc), series length, and anomaly description. As Fig. 9c *left* shows, all these fields in K_i are effective in improving selection ability. Desc and anomaly have a higher impact than name and length, which is reasonable because the description of scenarios and anomaly patterns is intuitive to have more context related to choosing a suitable TSAD model. We also note that the above ablations achieve 0.001-0.013 higher AUC-PR than the variant w/o MKI shown in Fig. 9a *left*, implying the effectiveness of MKI even with incomplete templates that have one of the fields missed.

(7) Study on metadata embedding. To study the effectiveness of knowledge feature extraction using PLM, we compare MKI with its variants that use simple (i) One-hot encoding of text K_i, and (ii) Numeric features consisting of one-hot features of the name and description of the datasets, concatenated with the values of the series length, anomaly count and lengths, as the knowledge features $\boldsymbol{z}_i^K$. As Fig. 9c *right* shows, both simple features are inferior to the default PLM embedding in terms of AUC-PR. Numeric is slightly better than One-hot that also uses K_i, which indicates that although the representation of metadata in text is flexible and informative, it is essential to incorporate a powerful language understanding model (e.g., PLMs) to extract useful meta-knowledge from the text.

Exp-5: Impact of key parameters. We first analyze the parameters of SDGP, including the pruning ratio r, the number of LSH bits q and bins w. We report the total selector learning time and the average AUC-PR for all test sets in Fig. 10, where the blue area shows the 95% confidence interval (CI). Note that the width of the CI is almost constant with each varying parameter, because the variation of AUC-PR is generally consistent on each dataset.

(1) Effect of r. As Fig. 10a shows, training time decreases with increasing r. For $r \leq 0.8$, the AUC-PR drops very slightly as r increases and is almost lossless compared to training without pruning, indicating that $r \leq 0.8$ can be safe for training speedup without much sacrificing the selection ability for each dataset. However, the AUC-PR will degrade dramatically with a further increase in r.

(2) Effect of q and w. As Fig. 10b shows, the time increases with increasing q, because fine-grained buckets with more hash tables will preserve more samples from being pruned. The AUC-PR generally improves as q increases within a low-to-moderate range ($q \leq 14$), but plateaus to the level of training without pruning when q further increases, as all samples will remain with $q \to +\infty$. The effect of w is similar (Fig. 10c) as it also controls the granularity of the buckets.

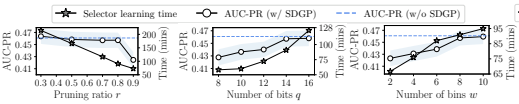


(a) Losses, rescaling, and difficulty score.

(b) PLM parameters and models.

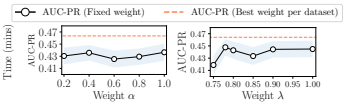
(c) Metadata template and embedding.

Figure 9: Ablation study results.



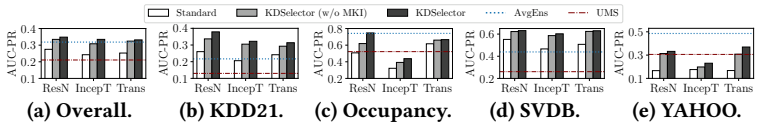
(a) Effect of r. (b) Effect of q. (c) Effect of w.

Figure 10: Impact of pruning parameters.



(a) Effect of α. (b) Effect of λ.

Figure 11: Impact of weights.



(a) Overall. (b) KDD21. (c) Occupancy. (d) SVDB. (e) YAHOO.

Figure 12: Out-of-domain evaluation. `Overall` means the average over all datasets.

Next, we study the weights α of PISL and λ of MKI. Since they mainly affect the selection ability, we only report the AUC-PR.

(3) Effect of α and λ. As Fig. 11 shows, the overall AUC-PR is in slight variations for α (Fig. 11a), while a relatively larger value (e.g., $[0.78, 1]$) can be better for λ (Fig. 11b). We also note that AUC-PR at each specific weight is always lower than that of using the best weight for each dataset, as the selector performs better with different values of α and λ for different applications (i.e., datasets).

Guidance for parameter setting. Based on the above analysis, we provide a guide to the setting of key parameters for deploying our KDSelector. By default, we recommend $r = 0.8$, $q = 14$ and $w = 8$ for SDGP considering both learning efficiency and selection ability, but one can easily tune them for a preferred tradeoff. For the importance of PISL and MKI, it is practical to determine them from a few generally better values, e.g., $\alpha \in \{0.2, 0.4, 1\}$ and $\lambda \in \{0.78, 1\}$ as our default, for an overall good selection ability without much tuning effort. One can also optimize them more precisely and efficiently using well-established hyper-parameter tuning systems [34, 35].

Exp-6: Evaluation on out-of-domain generalization. We further evaluate the generalization ability of the model selection methods for unseen domains. We train selectors over *all training data except a target domain (i.e., dataset)* and then assess the selectors on the target domain. Fig. 12 shows the overall performance averaged over all 16 target datasets and the results of four representative domains (e.g. healthcare and IT infra), from which we observe:

(1) KDSelector consistently improves AUC-PR for different architectures. Note that existing meta-learning-based solutions that use the standard selector learning method perform similarly to or even worse than AvgEns and UMS that do not rely on historical training data, while the methods optimized by our KDSelector achieve an overall performance superior to UMS and competitive to AvgEns (Fig. 12a), and perform significantly better than both baselines in some cases (e.g, KDD21 in Fig. 12b and SVDB in Fig. 12d).

(2) The ablated KDSelector that uses only PISL (i.e., w/o MKI) consistently outperforms the standard method, while the full KDSelector including MKI can further improve selection ability. This indicates that both modules are essential for improving generalizability to unseen domains. The reason is that PISL is helpful to learn the relationship between different time series and TSAD models, which may contain more domain-shared knowledge than a simple mapping to the best model. Meanwhile, MKI provides the meta-knowledge to enhance general-purpose context for MS (e.g., relationships between anomaly characteristics and TSAD model performance). By encouraging the time series and meta-knowledge features to be close while the features of different series to be distant, MKI is also beneficial for learning the inherent features that distinguish all time series from each other, which can disentangle domain and class to achieve more transferability across domains [69].

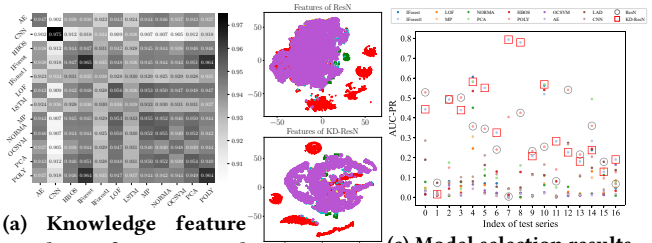
Note that the improvement of MKI seems limited compared to PISL because the meta-knowledge we use is still basic. Based on Exp-4 (6), MKI has the potential for further improvement in the real-world applications that have more useful meta-knowledge.

(3) In the extreme case where the target domain may be very different from the training data (e.g. YAHOO where many anomalies are isolated data points rather than segments [59]), the meta-learning-based methods with our KDSelector are still inferior to AvgEns in terms of AUC-PR. This indicates an optimization direction for future work that is to further improve the generalization ability of the selectors, to provide both MS and AD efficiency (shown in Exp-2) and detection performance with the meta-learning methods.

Exp-7: Study of the learned selector. To better understand how KDSelector improves the model selection ability of the learned selector, we first show in Fig. 13a the mean cosine similarity of the knowledge features corresponding to the time series (TS) of all 16 training sets that belong to the same (intra-) and different (inter-) classes (i.e., the best TSAD models). We observe that the intra-class similarity is higher than the inter-class similarity for 11 out of the 12 TSAD models, indicating that the PLM can extract semantic information from the metadata for distinguishing different models. Next, we conduct a study on IOPS, a data set containing performance indicators that reflect the scale, quality of web services, and health status of a machine. We visualize the TS features (i.e., z_i^T) of the ResN selectors in Fig. 13b for the test set using the two-dimensional t-SNE [47], where the features learned using the standard method (denoted ResN) and our KDSelector (denoted KD-ResN) are presented in *top* and *bottom*, respectively. Classes are distinguished using different colors. We observe that the TS features of KD-ResN form more separated clusters, suggesting low-entropy encoding that is easier to classify for the input subsequences, which explains why KDSelector can improve the model selection ability.

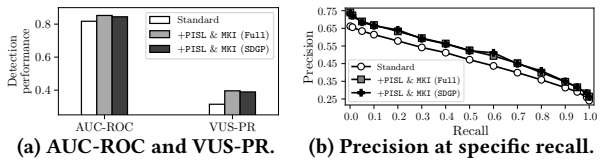
We further show the detection performance of the 12 TSAD models and the models chosen by ResN and KD-ResN for the test series in Fig. 13c. We see that both selectors correctly select the best models for 9/17 series. However, KD-ResN tends to choose a nearly optimal model even if it fails to identify the best one (e.g., for series 0, 3, 9, 13-15), but ResN selects the models of much lower AUC-PR compared to the best (e.g., 4, 7, 10, 11, 16). This indicates that the selector can better learn the relationships between time series and different TSAD models with the knowledge enhancement modules of KDSelector, resulting in better selection across different series.

Exp-8: Study of alternative metrics. To validate the generalization of our framework on metrics (used as P in Eq. (1)), we train selectors by replacing the default AUC-PR with alternative metrics commonly used in TSAD, including AUC-ROC and VUS-PR [58] (Fig. 14a), and precision at specific recall (Fig. 14b). The results show that KDSelector consistently improves the selection ability



(a) Knowledge feature similarity for intra- and inter-class TS. (b) TS features. (c) Model selection results for the test series.

Figure 13: Study of learned selectors.



(a) AUC-ROC and VUS-PR. (b) Precision at specific recall.

Figure 14: Study of alternative metrics.

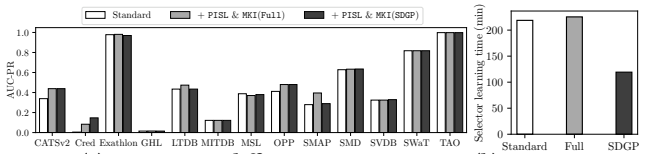
compared to the existing standard selector learning method, further indicating that KDSelector is a general framework that can be integrated into TSAD services that use different metrics of interest.

Exp-9: Generalization on multivariate time series (MTS). We further evaluate our framework using MTS. We use the recently published TSB-AD-M dataset [46] which contains heterogeneous MTS from different areas (see [76] in detail). For benchmarking, we use the data split recommended by [45] and ResN with the same settings as in § 5.1. Since IForest1, MP and POLY are not directly applicable to MTS, we exclude them and use the remaining TSAD models. The results are shown in Fig. 15. Similarly, learning with our PISL and MKI (Full) improves the selection ability on almost all MTS test sets with only 3% increase in training time, while pruning with SDGP saves 47% time with almost lossless or even improved selection ability. The average AUC-PR of Full has improved by 0.0303 compared to Standard. Although this is lower than the 0.0396 (see Table 1) achieved in the univariate scenario, it is still significant.

Exp-10: Study of selective model combination. Considering that even with improved selection ability, the selector can still fail to choose the best model, and there might be diverse top-performing models that complement each other, we study a practical solution to run the models of the top-k votes and combine their scores (e.g., based on *Average* or *Weighted Average according to their votes*) for detection. We observe that the performance of all selection variants improves after $k > 1$ and plateaus or degrades after k of 2-5 (Fig. 16a), while the detection time increases with k (Fig. 16b). It suggests an effective balance between accuracy and runtime with a proper k, compared to using only top-1 or a naïve ensemble of all models (e.g., AvgEns). WA is generally a better strategy than A because it takes into account the importance (i.e., votes) of different models. KDSelector outperforms Standard for both strategies, further verifying the effectiveness of our framework.

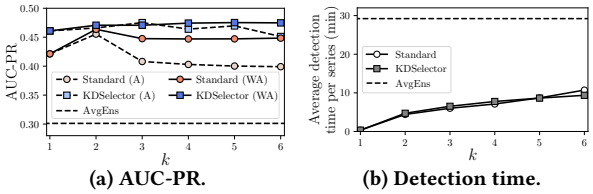
6 RELATED WORK

Model selection for TSAD. In general, existing studies on TSAD model selection are in two lines. Goswami et al. [20] propose a *trial-based* method that chooses the TSAD model using three carefully designed surrogate metrics. The method falls short of the high computational overhead required to run all TSAD models and calculate the metrics. It is also difficult to accurately estimate the detection performance using the hand-crafted metrics. On the other hand, Ying et al. [81] introduce the *meta-learning-based* model selection mechanism and show the effectiveness in their production.



(a) AUC-PR on different test sets. (b) Learning time.

Figure 15: Evaluation on multivariate time series.



(a) AUC-PR. (b) Detection time.

Figure 16: Study of top-k model combination.

Navarro et al. [52] design improved meta-learners for multivariate and dissimilar time series. Sylligardos et al. [70] propose a general meta-learning pipeline (see § 2.2) and adapt different TSC models to be selectors, showing advantages in selection ability and efficiency.

Data pruning for NN training. Data pruning emerges as a prevailing trend in both database and machine learning areas. Existing methods can be divided into two types. *Static pruning* methods [27, 29, 31, 60, 79, 80] remove less important samples using predefined metrics and train the model only on the remaining data. These methods suffer excessive computational costs for large-scale data collections, which is even higher than training. Moreover, the predefined metrics hardly perform well across different datasets and architectures. In contrast, *dynamic pruning* methods [61, 63] adaptively select subsets of samples during training based on sample information (i.e., loss values) at different epochs, which has lower pruning costs and is more generalized.

Improvement for TSC. There exist improvement methods for general-purpose TSC, such as self-supervised pre-training [9, 37, 38, 42, 84], data augmentation [25, 32, 77], privacy protection [39–41], etc, which can also be jointly used with KDSelector for selector learning, so our method is complementary to them.

7 CONCLUSIONS AND FUTURE WORK

This paper presents KDSelector, a novel framework for NN-based TSAD model selector learning. We design two knowledge enhancement modules to utilize the auxiliary knowledge in historical data to improve the model selection ability. We further propose a novel dynamic data pruning framework with theoretical guarantees to accelerate selector learning with almost lossless selection ability. Extensive experiments verify the effectiveness of our proposals.

In the future, it is interesting to explore model selection in a finer granularity, i.e., identifying the best TSAD model for each subsequence independently, which may further improve TSAD performance. Such cases face a non-trivial challenge in subsequence labeling, as anomalies are usually rare and thereby many subsequences contain only normal points, making standard metrics like AUC-PR, VUS-PR, and recall inapplicable for evaluation on those subsequences. Moreover, a realistic problem is how to construct rich and high-quality metadata to bring our method into full play.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their invaluable time and efforts in reviewing our paper. This paper was supported by the NSFC grant (62232005, 62502123), the Smart Grid-National Science and Technology Major Project (2024ZD0802900), and the China Postdoctoral Science Foundation (2024M764191).

REFERENCES

[1] Charu C Aggarwal and Saket Sathe. 2015. Theoretical foundations and algorithms for outlier ensembles. *Acm sigkdd explorations newsletter* 17, 1 (2015), 24–47.

[2] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. 2017. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery* 31, 3 (2017), 606–660.

[3] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. LLM2Vec: Large Language Models Are Secretly Powerful Text Encoders. arXiv:2404.05961

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. 41–48.

[5] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954* (2024).

[6] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. 2021. A review on outlier/anomaly detection in time series data. *ACM computing surveys (CSUR)* 54, 3 (2021), 1–33.

[7] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. 1992. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*. 144–152.

[8] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 380–388.

[9] Yuxuan Chen, Shanshan Huang, Yunyao Cheng, Peng Chen, Zhongwen Rao, Yang Shu, Bin Yang, Lujia Pan, and Chenjuan Guo. 2025. AimTS: Augmented Series and Image Contrastive Learning for Time Series Classification. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 1952–1965.

[10] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W Kempa-Liehr. 2018. Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing* 307 (2018), 72–77.

[11] Mirza Cilimkovic. 2015. Neural networks and back propagation algorithm. *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin* 15, 1 (2015), 18.

[12] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*. 233–240.

[13] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*. 4171–4186.

[15] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv e-prints* (2024), arXiv–2407.

[16] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. 2015. A time-series compression technique and its application to the smart grid. *The VLDB Journal* 24, 2 (2015), 193–218.

[17] Raul Castro Fernandez, Aaron J Elmore, Michael J Franklin, Sanjay Krishnan, and Chenhao Tan. 2023. How large language models will disrupt data management. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3302–3309.

[18] Evelyn Fix. 1985. *Discriminatory analysis: nonparametric discrimination, consistency properties*. Vol. 1. USAF school of Aviation Medicine.

[19] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139.

[20] Mononito Goswami, Cristian Ignacio Challu, Laurent Callot, Lenon Minorics, and Andrey Kan. [n.d.]. Unsupervised Model Selection for Time Series Anomaly Detection. In *The Eleventh International Conference on Learning Representations*.

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

[22] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[23] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[24] Julius Hülsmann, Jonas Traub, and Volker Markl. 2020. Demand-based sensor data gathering with multi-query optimization. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2801–2804.

[25] Romain Ilbert, Thai V Hoang, and Zonghua Zhang. 2024. Data augmentation for multivariate time series classification: An experimental study. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*. IEEE, 128–139.

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

[27] Rishabh Iyer, Ninad Khargoankar, Jeff Bilmes, and Himanshu Asanani. 2021. Submodular combinatorial information measures with applications in machine learning. In *Algorithmic Learning Theory*. PMLR, 722–754.

[28] Søren Kejser Jensen, Torben Bach Pedersen, and Christian Thomsen. 2017. Time series management systems: A survey. *IEEE Transactions on Knowledge and Data Engineering* 29, 11 (2017), 2581–2600.

[29] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, and Rishabh Iyer. 2021. Glister: Generalization based data subset selection for efficient and robust learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 8110–8118.

[30] Siwon Kim, Kukjin Choi, Hyun-Soo Choi, Byunghan Lee, and Sungroh Yoon. 2022. Towards a rigorous evaluation of time-series anomaly detection. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 36. 7194–7201.

[31] Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International conference on machine learning*. PMLR, 1885–1894.

[32] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. 2016. Data augmentation for time series classification using convolutional neural networks. In *ECML/PKDD workshop on advanced analytics and learning on temporal data*.

[33] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[34] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. 2020. A system for massively parallel hyperparameter tuning. *Proceedings of machine learning and systems* 2 (2020), 230–246.

[35] Yang Li, Yu Shen, Huaijun Jiang, Wentao Zhang, Jixiang Li, Ji Liu, Ce Zhang, and Bin Cui. 2022. Hyper-tune: towards efficient hyper-parameter tuning at scale. *Proceedings of the VLDB Endowment* 15, 6 (2022), 1256–1265.

[36] Zhiyu Liang, Dongrui Cai, Chenyuan Zhang, Zheng Liang, Chen Liang, Bo Zheng, Shi Qiu, Jin Wang, and Hongzhi Wang. 2025. KDSelector: A Knowledge-Enhanced and Data-Efficient Model Selector Learning Framework for Time Series Anomaly Detection. In *Companion of the 2025 International Conference on Management of Data*. 175–178.

[37] Zhiyu Liang, Chen Liang, Zheng Liang, Hongzhi Wang, and Bo Zheng. 2024. TimeCSL: Unsupervised Contrastive Learning of General Shapelets for Explorable Time Series Analysis. *Proceedings of the VLDB Endowment* 17, 12 (2024), 4489–4492.

[38] Zhiyu Liang, Chen Liang, Zheng Liang, Hongzhi Wang, and Bo Zheng. 2024. Units: A universal time series analysis framework powered by self-supervised representation learning. In *Companion of the 2024 International Conference on Management of Data*. 480–483.

[39] Zhiyu Liang, Zheng Liang, Hongzhi Wang, and Bo Zheng. 2025. FedDict: Towards Practical Federated Dictionary-Based Time Series Classification. *IEEE Transactions on Knowledge and Data Engineering* (2025).

[40] Zhiyu Liang and Hongzhi Wang. 2022. Fedtsc: a secure federated learning system for interpretable time series classification. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3686–3689.

[41] Zhiyu Liang and Hongzhi Wang. 2024. FedST: secure federated shapelet transformation for time series classification. *The VLDB Journal* 33, 5 (2024), 1617–1641.

[42] Zhiyu Liang, Jianfeng Zhang, Chen Liang, Hongzhi Wang, Zheng Liang, and Lujia Pan. 2023. A Shapelet-Based Framework for Unsupervised Multivariate Time Series Representation Learning. *Proceedings of the VLDB Endowment* 17, 3, 386–399.

[43] Yin Lin, Bolin Ding, and Jingren Zhou. 2025. Large Language Models as Pretrained Data Engineers: Techniques and Opportunities. *IEEE Data Eng. Bull.* 49, 1 (2025), 70–89.

[44] Qinghua Liu, Paul Boniol, Themis Palpanas, and John Paparrizos. 2024. Time-series anomaly detection: Overview and new trends. *Proceedings of the VLDB Endowment (PVLDB)* 17, 12 (2024), 4229–4232.

[45] Qinghua Liu, Seunghak Lee, and John Paparrizos. 2025. TSB-AutoAD: Towards Automated Solutions for Time-Series Anomaly Detection. *Proc. VLDB Endow.* 18, 11 (Sept. 2025), 4364–4379. https://doi.org/10.14778/3749646.3749699

[46] Qinghua Liu and John Paparrizos. 2025. The elephant in the room: towards a reliable time-series anomaly detection benchmark. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 3437, 31 pages.

[47] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[48] Xupeng Miao, Zhihao Jia, and Bin Cui. 2024. Demystifying data management for large language models. In *Companion of the 2024 International Conference on Management of Data*. 547–555.

[49] Navid Mohammadi Foumani, Lynn Miller, Chang Wei Tan, Geoffrey I Webb, Germain Forestier, and Mahsa Salehi. 2024. Deep learning for time series classification and extrinsic regression: A current survey. *Comput. Surveys* 56, 9 (2024), 1–45.

[50] Tianyu Mu, Hongzhi Wang, Shenghe Zheng, Zhiyu Liang, Chunnan Wang, Xinyue Shao, and Zheng Liang. 2023. Tsc-automl: Meta-learning for automatic

time series classification algorithm selection. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1032–1044.

[51] M Muralikrishna and David J DeWitt. 1988. Equi-depth multidimensional histograms. In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*. 28–36.

[52] Jose Manuel Navarro, Alexis Huet, and Dario Rossi. 2023. Meta-Learning for Fast Model Recommendation in Unsupervised Multivariate Time Series Anomaly Detection. In *Proceedings of the Second International Conference on Automated Machine Learning (Proceedings of Machine Learning Research)*, Aleksandra Faust, Roman Garnett, Colin White, Frank Hutter, and Jacob R. Gardner (Eds.), Vol. 224. PMLR, 24/1–19. https://proceedings.mlr.press/v224/navarro23a.html

[53] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology* (2023).

[54] Irene CL Ng and Susan YL Wakenshaw. 2017. The Internet-of-Things: Review and research directions. *International Journal of Research in Marketing* 34, 1 (2017), 3–21.

[55] Appendix of KDSelector. 2025. https://github.com/chenyuanTKCY/KDSelectorAlgorithm/blob/master/appendix/Appendix_of_KDSelector.pdf.

[56] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[57] Themis Palpanas. 2015. Data series management: The road to big sequence analytics. *ACM SIGMOD Record* 44, 2 (2015), 47–52.

[58] John Paparrizos, Paul Boniol, Themis Palpanas, Ruey S Tsay, Aaron Elmore, and Michael J Franklin. 2022. Volume under the surface: a new accuracy evaluation measure for time-series anomaly detection. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2774–2787.

[59] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. 2022. TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment* 15, 8 (2022), 1697–1711.

[60] Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. 2021. Deep learning on a data diet: Finding important examples early in training. *Advances in neural information processing systems* 34 (2021), 20596–20607.

[61] Ziheng Qin, Kai Wang, Zangwei Zheng, Jianyang Gu, Xiangyu Peng, Daquan Zhou, Lei Shang, Baigui Sun, Xuansong Xie, Yang You, et al. [n.d.]. InfoBatch: Lossless Training Speed Up by Unbiased Dynamic Data Pruning. In *The Twelfth International Conference on Learning Representations*.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[63] Ravi S Raju, Kyle Daruwalla, and Mikko Lipasti. 2021. Accelerating deep learning with dynamic data pruning. *arXiv preprint arXiv:2111.12621* (2021).

[64] Hansheng Ren, Bixiong Xu, Yujing Wang, Chao Yi, Congrui Huang, Xiaoyu Kou, Tony Xing, Mao Yang, Jie Tong, and Qi Zhang. 2019. Time-series anomaly detection service at microsoft. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 3009–3017.

[65] Sean Rhea, Eric Wang, Edmund Wong, Ethan Atkins, and Nat Storer. 2017. Littletable: A time-series database and its uses. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 125–138.

[66] Lei Rui, Xiangdong Huang, Shaoxu Song, Yuyuan Kang, Chen Wang, and Jianmin Wang. 2024. Time Series Representation for Visualization in Apache IoTDB. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–26.

[67] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2022. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment* 15, 9 (2022), 1779–1797.

[68] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2022. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment* 15, 9, 1779–1797.

[69] Kendrick Shen, Robbie M Jones, Ananya Kumar, Sang Michael Xie, Jeff Z HaoChen, Tengyu Ma, and Percy Liang. 2022. Connect, not collapse: Explaining contrastive learning for unsupervised domain adaptation. In *International conference on machine learning*. PMLR, 19847–19878.

[70] Emmanouil Sylligardos, Paul Boniol, John Paparrizos, Panos Trahanias, and Themis Palpanas. 2023. Choose wisely: An extensive evaluation of model selection for anomaly detection in time series. *Proceedings of the VLDB Endowment* 16, 11, 3418–3432.

[71] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I Webb. 2022. MultiRocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery* 36, 5 (2022), 1623–1646.

[72] Sahar Torkamani and Volker Lohweg. 2017. Survey on time series motif discovery. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 2 (2017), e1199.

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[74] Paul Viola and William M Wells III. 1997. Alignment by maximization of mutual information. *International journal of computer vision* 24, 2 (1997), 137–154.

[75] Chen Wang, Jialin Qiao, Xiangdong Huang, Shaoxu Song, Haonan Hou, Tian Jiang, Lei Rui, Jianmin Wang, and Jiaguang Sun. 2023. Apache iotdb: A time series database for iot applications. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[76] TSB-AD Website. 2025. https://thedatumorg.github.io/TSB-AD/.

[77] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. 2020. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478* (2020).

[78] Mike Wu, Chengxu Zhuang, Milan Mosse, Daniel Yamins, and Noah Goodman. 2020. On mutual information in contrastive learning for visual representations. *arXiv preprint arXiv:2005.13149* (2020).

[79] Xiaobo Xia, Jiale Liu, Jun Yu, Xu Shen, Bo Han, and Tongliang Liu. 2022. Moderate coreset: A universal method of data selection for real-world data-efficient deep learning. In *The Eleventh International Conference on Learning Representations*.

[80] Shuo Yang, Zeke Xie, Hanyu Peng, Min Xu, Mingming Sun, and Ping Li. [n.d.]. Dataset Pruning: Reducing Training Data by Examining Generalization Influence. In *The Eleventh International Conference on Learning Representations*.

[81] Yuanxiang Ying, Juanyong Duan, Chunlei Wang, Yujing Wang, Congrui Huang, and Bixiong Xu. 2020. Automated model selection for time-series anomaly detection. *arXiv preprint arXiv:2009.04395* (2020).

[82] Zahra Zamanzadeh Darban, Geoffrey I Webb, Shirui Pan, Charu Aggarwal, and Mahsa Salehi. 2024. Deep learning for time series anomaly detection: A survey. *Comput. Surveys* 57, 1 (2024), 1–42.

[83] Lianming Zhang, Wenji Bai, Xiaowei Xie, Liying Chen, and Pingping Dong. 2023. TMANomaly: Time-series mutual adversarial networks for industrial anomaly detection. *IEEE Transactions on Industrial Informatics* 20, 2 (2023), 2263–2271.

[84] Tian Zhou, Peisong Niu, Liang Sun, Rong Jin, et al. 2023. One fits all: Power general time series analysis by pretrained lm. *Advances in neural information processing systems* 36 (2023), 43322–43355.