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

Zhiyu Liang
Harbin Institute of Technology
Harbin, China
zyliang@hit.edu.cn

Dongrui Cai[*]
Chenyuan Zhang[*]
2021113243@stu.hit.edu.cn
2022113454@stu.hit.edu.cn
Harbin Institute of Technology
Harbin, China

Zheng Liang
Chen Liang
lz20@hit.edu.cn
23B903050@stu.hit.edu.cn
Harbin Institute of Technology
Harbin, China

Bo Zheng
CnosDB Inc.
Beijing, China
harbour.zheng@cnosdb.com

Shi Qiu
Jin Wang
{sheldon.qiu,jerryW}@csu.edu.cn
Central South University
Changsha, China

Hongzhi Wang[†]
Harbin Institute of Technology
Harbin, China
wangzh@hit.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 the highly heterogeneous time series in real-world applications. However, despite the success of existing model selection solutions that train a classification model (especially neural network, NN) using historical data as a selector to predict the correct TSAD model for each series, the NN-based selector learning methods used by existing solutions do not make full use of the knowledge in the historical data and require iterating over all training samples, which limits the accuracy and training speed of the selector. To address these limitations, we propose KDSelector, a novel knowledge-enhanced and data-efficient framework for learning the NN-based TSAD model selector, of which three key components are specifically designed to integrate available knowledge into the selector and dynamically prune less important and redundant samples during the learning. We develop a TSAD model selection system with KDSelector as the internal, to demonstrate how users improve the accuracy and training speed of their selectors by using KDSelector as a plug-and-play module.

CCS Concepts

• **Mathematics of computing → Time series analysis**; • **Computing methodologies → Machine learning**.

Keywords

Model selection, Time series, Anomaly detection

SIGMOD-Companion '25, Berlin, Germany
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0422-2/24/06
https://doi.org/10.1145/3626246.3654733

ACM Reference Format:
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 (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3626246.3654733

1 Introduction

Time series anomaly detection (TSAD) is an important technique for many real-world applications [8]. However, as widely shown, there is ***no single best TSAD method (a.k.a. model) when applied to different time series***, due to the highly heterogeneous nature of the data in terms of the types, numbers, and lasting time of the anomalies, etc [8, 9]. A straightforward thought is to combine all TSAD models through ensembling. Nevertheless, such solutions require running multiple TSAD methods, causing excessive computational costs that are prohibitive for large time series collections.

To overcome the above issues, recent work [9] proposes ***model selection methods to automatically select the best TSAD model for different time series based on their data characteristics***. This is usually achieved by training (a.k.a. learning) a time series classification [4] (TSC) model as a **selector** to classify time series into discrete categories that represent the TSAD models to select, using the historical data such as the previously seen time series and the corresponding correct TSAD models as training samples. By such solutions, only the selected TSAD model is run for each time series to detect, which is more scalable than the aforementioned ensemble methods that require running all candidates.

Challenges. Among existing approaches, neural network (NN)-based selectors (i.e., TSC models) have shown superior accuracy [9] due to their ability to learn complex relationships between time series and TSAD models. However, the selector learning methods used by the existing solutions face two main challenges.

Firstly, existing methods only use the time series and the hard labels that represent the best TSAD models for the corresponding series as training data, ***ignoring that there usually exists additional knowledge in the historical data***, such as the detection

[*] Equal contribution.
[†] Corresponding author.

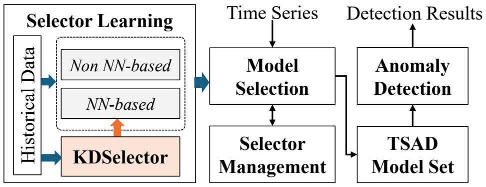


Figure 1: System architecture.

performance of all TSAD model candidates used for identifying the hard labels [9], and the metadata that reflects the characteristics of the time series, anomalies, TSAD scenarios, etc [8]. Thus, the learned selector can be weak in choosing a good TSAD model.

Secondly, NN-based selector learning in existing solutions, following the widely used stochastic gradient descent (SGD) scheme [3], ***requires iterating over all training samples, which is data-inefficient and time-consuming.*** Although there are advanced acceleration methods [6] that can dynamically prune less important training samples for each training epoch while guaranteeing little loss of accuracy, these approaches have not considered the specific data characteristics in the TSAD model selection problem, which causes them to provide only limited speedup.

Contributions. To address the first challenge, we propose ***two knowledge enhancement modules*** to integrate the knowledge into the selector to improve its accuracy. To utilize the detection performance of all TSAD models, we design a ***performance-informed selector learning (PISL)*** module that transforms the performance scores of different TSAD models into the probabilities of selecting the corresponding models, which is taken as a soft label to better train the selector. To gain knowledge from diverse metadata, we propose a ***meta-knowledge integration (MKI)*** module. The module takes natural language as input, so that any type of metadata can be flexibly incorporated. It transforms the input text into unified embedding (i.e., feature vector) via a pre-trained large language model (LLM) [2]. Then, it integrates the knowledge within the input into the selector by maximizing the mutual information between the feature vectors of the text and the time series.

To cope with the second challenge, we propose ***a novel pruning-based acceleration (PA) framework*** for NN-based selector training that can prune more training samples at each epoch with still nearly lossless model accuracy. Our key observation is that there are training samples that are very similar to each other and also contribute to almost equal training losses. Based on our theoretical analysis, these training samples 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 ensures that training on the pruned dataset can achieve a similar result as training on the original one.

To the best of our knowledge, our novel solution, which we name `KDSelector`***, is the first framework for NN-based TSAD model selector learning that aims to improve the accuracy and training speed via knowledge enhancement and data pruning***. It is noteworthy that the three proposed key components, including PISL, MKI, and PA, are all ***plug-and-play frameworks*** that are ***agnostic to NN architectures*** (e.g., ResNet or Transformer [9]) and ***independent of each other***. This means that users can flexibly integrate each of them into their own TSAD model selection tasks where any selector architecture can be used.

This paper aims to ***demonstrate our KDSelector in two aspects***, including (i) guiding audiences to learn a selector and apply it to TSAD model selection on their own data, and (ii) showcasing the effectiveness and superiority of the proposed methods in improving the accuracy and training speed of the selectors. To achieve our goal, we develop ***an end-to-end system*** to enable TSAD model selection using different TSC methods (i.e., selectors), where our ***KDSelector can be flexibly used for training any NN-based selector***. Currently, we have implemented ***12 TSAD models*** and ***15 selectors***, and provided ***16 different datasets*** to facilitate evaluation. Our code is available at ***https://github.com/chenyuanTKCY/KDSelector***.

2 System Overview

Preliminaries. Formally, the problem of TSAD model selection is defined as follows.

Definition 2.1 (TSAD model selection). *Given a set of TSAD models, denoted as $\mathcal{M} = \{M_1, \ldots, M_m\}$, TSAD model selection aims to build a function (i.e., selector) f to predict the model in $\mathcal{M}$ that has the best detection performance for an input time series $T \in \mathbb{R}^L$, i.e.,*

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

where P can be any interested metric, such as AUC-PR or F1-Score.

Definition 2.1 can be seen as a specific TSC problem [4], where the selector f is a TSC model that maps a time series T to a class that represents the TSAD model to select from $\mathcal{M}$. Therefore, we can adopt any existing TSC method to build f, using available historical data such as the previously seen time series and the corresponding correct TSAD models as training samples [9]. To address real-world time series of variable lengths, we follow [9] to preprocess each raw time series by *extracting fixed-length subsequences* using a window of size L. The selector f predicts a TSAD model for each subsequence, while majority voting is used to *select one model for each time series from the predicted models of all its subsequences.*

System architecture. Fig. 1 shows the architecture of our TSAD model selection system, which includes five main components.

The **Selector Learning** module, which is the key to model selection, aims to train TSC models as selectors using historical data. Currently, the system supports 15 different selectors, including both NN-based and non-NN-based, where ***our novel KDSelector, as illustrated in Sect. 3, is used as a plug-and-play framework to improve the learning of any NN-based selector***. For evaluation and demonstration purposes, we have prepared the 16 different TSAD datasets used in [9] as historical data. Audiences can also test on their own data using our system.

The system provides a **Selector Management** module for users to easily save, manage, and load their learned selectors. Given a learned selector, the **Model Selection** module predicts the best model among the **TSAD Model Set** for each time series to detect. The selected model is run by the **Anomaly Detection** module and the detection results (e.g., anomaly score and overall performance) are visually shown to the users. We have now implemented 12 representative models in the TSAD model set following [9]. More models [8] can be integrated in the same way in future work.

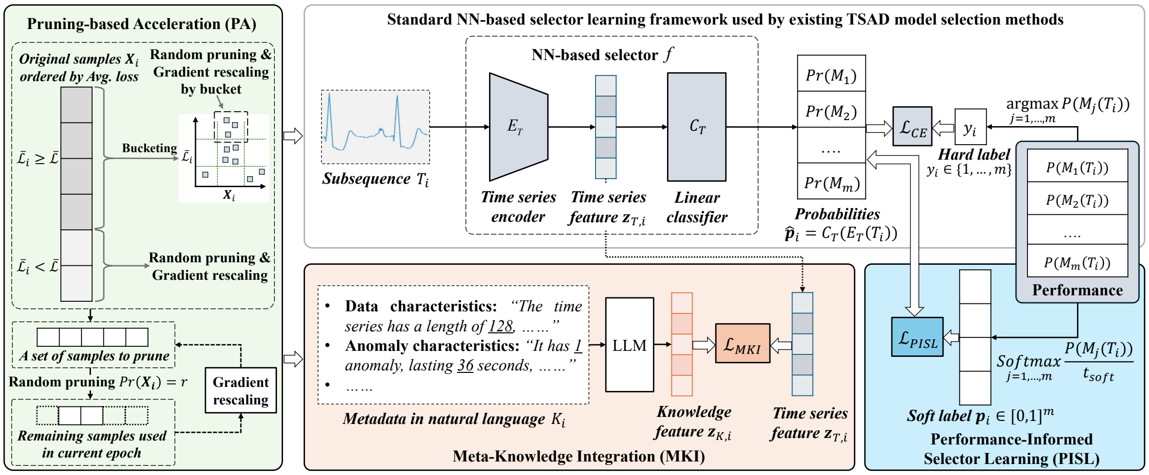

Pruning-based Acceleration (PA)
Original samples X_i ordered by Avg. loss
Random pruning & Gradient rescaling by bucket
Bucketing
$\bar{\mathcal{L}}_i \geq \bar{\mathcal{L}}$
$\bar{\mathcal{L}}_i < \bar{\mathcal{L}}$
$\bar{\mathcal{L}}_i$
X_i
Random pruning & Gradient rescaling
A set of samples to prune
Random pruning $Pr(X_i) = r$
Gradient rescaling
Remaining samples used in current epoch
Standard NN-based selector learning framework used by existing TSAD model selection methods
NN-based selector f
Subsequence T_i
E_T
Time series encoder
Time series feature $z_{T,i}$
C_T
Linear classifier
$Pr(M_1)$
$Pr(M_2)$
....
$Pr(M_m)$
Probabilities $\hat{p}_i = C_T(E_T(T_i))$
$\mathcal{L}_{CE}$
y_i
Hard label $y_i \in \{1, ..., m\}$
$\arg\max_{j=1,...,m} P(M_j(T_i))$
$P(M_1(T_i))$
$P(M_2(T_i))$
....
$P(M_m(T_i))$
Performance
Data characteristics: "The time series has a length of 128, ..."
Anomaly characteristics: "It has 1 anomaly, lasting 36 seconds, ..."
......
Metadata in natural language K_i
LLM
$\mathcal{L}_{MKI}$
Knowledge feature $z_{K,i}$
Time series feature $z_{T,i}$
Meta-Knowledge Integration (MKI)
$\mathcal{L}_{PISL}$
$Softmax_{j=1,...,m} \frac{P(M_j(T_i))}{t_{soft}}$
Soft label $\mathbf{p}_i \in [0,1]^m$
Performance-Informed Selector Learning (PISL)

Figure 2: Overall framework of KDSelector

3 System Internals: KDSelector

Next, we introduce the proposed KDSelector, which is a general NN-based selector learning framework that aims to address the aforementioned challenges and serves as the internal of our system.

Framework overview. Fig. 2 illustrates the framework of KDSelector. Generally, *an NN-based selector f (i.e., TSC model) is a time series encoder E_T appended by a linear classifier C_T*. For each input subsequence T_i, the selector first transforms it into a feature vector $z_{T,i} = E_T(T_i)$. Then, the selector maps $z_{T,i}$ to a vector $\hat{\boldsymbol{p}}_i = (Pr(M_1), \ldots, Pr(M_m)) = C_T(z_{T,i})$ that represents the predicted probabilities of selecting the corresponding TSAD models, where the model with the highest probability is chosen.

As Fig. 2 shows, the ***standard NN-based selector learning framework used by existing approaches*** [9] only uses the *hard label* $y_i = \arg\max_{j=1,\ldots,m} P(M_j(T_i))$ *that represents the model with the best performance* to learn f, by minimizing the commonly used cross-entroy loss (denoted as $\mathcal{L}_{CE}$) between $\hat{\boldsymbol{p}}_i$ and y_i. Meanwhile, it requires iterating over all training samples at each epoch. As discussed in Sect. 1, the above issues limit the performance of existing TSAD model selectors in terms of accuracy and training speed. To tackle these limitations, we design three plug-and-play modules that can be seamlessly integrated into the standard NN-based selector learning framework, which are described as follows.

Performance-informed selector learning (PISL). Considering that the detection performance $P(M_j(T_i))$, $j = 1, \ldots, m$ not only indicates which model is the best for T_i (i.e., y_i), but also reflects the complex relationship between the performance of all different models, we design PISL to make full use of the latter information to better train the selector. In principle, the TSAD model with better performance should have a higher probability of being selected. Thus, PISL *transforms the performance scores into a probability distribution of selecting the corresponding models* using the *Softmax* function, i.e., $\boldsymbol{p}_i = Softmax_{j=1,\ldots,m}\ P(M_j(T_i))/t_{soft}$, where t_{soft} is a hyperparameter that controls the smoothness of the distribution. The distribution $\boldsymbol{p}_i$ is used as a *soft target (a.k.a. label)* to train f, which is achieved by *minimizing the cross-entropy between the predicted distribution $\hat{\boldsymbol{p}}$ and the target $\boldsymbol{p}_i$*. Formally, the objective function is defined as $\mathcal{L}_{PISL} = \sum_{j=1}^{m} \boldsymbol{p}_{i,j} \log \hat{\boldsymbol{p}}_{i,j}$.

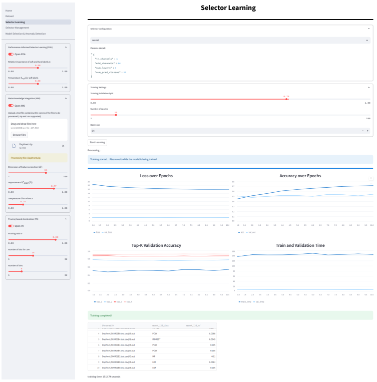
PISL can be integrated into existing NN-based selector learning methods regardless of NN architectures, by optimizing the objective $(1-\alpha)\mathcal{L}_{CE} + \alpha\mathcal{L}_{PISL}$, where α controls the relative importance of the soft label $\boldsymbol{p}_i$ and the hard label y_i.

Meta-knowledge integration (MKI). To gain knowledge from diverse metadata, MKI is designed to take natural languages (i.e., texts) as input to allow flexible and easy description of all kinds of metadata (e.g., the data and anomaly characteristics shown in Fig. 2). The input, denoted as K_i, is then fed into a pre-trained LLM (e.g., BERT [2]) to *transform the text into a unified feature vector $z_{K,i}$*, to take advantage of the superior ability of LLMs in natural language understanding. To integrate the knowledge in the metadata into the selector, from the perspective of information theory, we design a learning objective to *maximize the mutual information (MI) between the features of the time series and the metadata*. It is achieved by mapping $z_{T,i}$ and $z_{K,i}$ into a shared space $\mathbb{R}^H$ using two projections h_T and h_K, respectively, and then minimizing the InfoNCE loss [5] (denoted as $\mathcal{L}_{InfoNCE}$) that represents the opposite of a lower bound of MI between two random variables. The objective function is denoted as $\mathcal{L}_{MKI} = \mathcal{L}_{InfoNCE}(\{h_T(z_{T,i}), h_K(z_{K,i})|\forall i\})$.

Similar to PISL, users can integrate MKI just by adding $\lambda\mathcal{L}_{MKI}$ to the total loss, where λ is used to control the importance of MKI.

Pruning-based acceleration (PA). To achieve data-efficient NN training, the state-of-the-art method, namely *InfoBatch* [6], evaluates the importance of each sample $\boldsymbol{X}_i$ ($\boldsymbol{X}_i = \{T_i, z_{K,i}\}$ with MKI and T_i otherwise) using its average loss in the past epochs (denoted as $\bar{\mathcal{L}}_i$), and randomly prunes each less important sample (i.e., $\boldsymbol{X}_i$ if $\bar{\mathcal{L}}_i < \bar{\mathcal{L}}$ where $\bar{\mathcal{L}}$ is the average loss of all samples) with a probability r. It then iterates over the remaining samples in the current epoch, where for the samples of $\bar{\mathcal{L}}_i < \bar{\mathcal{L}}$, the gradients used for SGD are rescaled by multiplying $1/(1-r)$ to maintain that training on the pruned dataset is similar to training on the original one.

However, in the TSAD model selection problem, there can be redundant samples $\boldsymbol{X}_i$ for $\bar{\mathcal{L}}_i \geq \bar{\mathcal{L}}$ that cannot be pruned by InfoBatch. In specific, there may exist *samples that are similar to each other and have similar average losses.* By our theoretical analysis (detailed in Sect. A.1 of our technical report [7], TR), these samples have *almost identical contributions for training the selector.* Thus, we


Home
Dataset
Selector Learning
Selector Management
Model Selection & Anomaly Detection
Performance-Informed Selector Learning (PISL)
Open PISL
Meta-Knowledge Integration (MKI)
Open MKI
Drag and drop files here
Browse files
Pruning-based Acceleration (PA)
Open PA
Start Learning

Selector Learning

Selector Configuration

resnet

Params detail:

```
{
  "in_channels" : 1
  "mid_channels" : 64
  "num_layers" : 3
  "num_pred_classes" : 12
}
```

Training Settings

Training/Validation Split: 0.70

Number of epochs: 10

Batch size: 64

Upload a text file containing the names of the files to be processed (.zip and .rar supported)

Daphnet.zip

Processing file: Daphnet.zip

Processing...

Training started... Please wait while the model is being trained.

Loss over Epochs

Accuracy over Epochs

Top-K Validation Accuracy

Train and Validation Time

Training completed!

training time: 1512.74 seconds

Figure 3: The system interfaces.

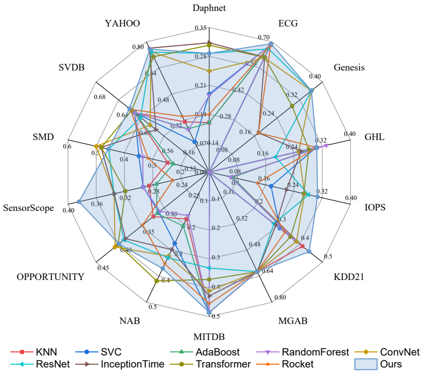


Figure 4: AUC-PR of different solutions.

Table 1: Results of PISL and MKI. (AUC-PR/total training time on all test/train sets. The same below.)

Method	Standard	+ PISL	+ MKI	+ PISL & MKI
AUC-PR	0.421	0.449	0.424	**0.461**
Time (mins)	281.90	**280.42**	282.05	282.03

Table 2: Results of PA on all datasets.

Method	Full data	+ InfoBatch	+ PA (Ours)
AUC-PR	**0.461**	$0.455_{\downarrow 0.006}$	$0.452_{\downarrow 0.009}$
Time (mins)	282.03	$171.73_{\downarrow 39.1\%}$	$\mathbf{117.72}_{\downarrow 58.3\%}$

Table 3: Results of KDSelector on different architectures (all datasets).

Architecture	ResNet	InceptionTime	Transformer
Improved AUC-PR	0.040	0.046	0.015
Saved time (%)	58.3%	70.96%	74.17%

propose ***a novel strategy to prune these redundant samples to speed up the selector learning***. The core idea is to *divide the samples X_i of $\bar{\mathcal{L}}_i \geq \bar{\mathcal{L}}$ into different buckets, where the samples within each bucket are similar in both themselves and their average losses, and then perform pruning by bucket.* Considering that the values of the training samples are invariant during the training, we use local sensitive hashing [1] (LSH) to *efficiently hash all similar samples to the same hash tables before the training starts.* At each training epoch, we *divide X_i of $\bar{\mathcal{L}}_i \geq \bar{\mathcal{L}}$ into p equi-depth bins according to the current $\bar{\mathcal{L}}_i$, divide the samples that fall into the same hash table and bin into one bucket, and perform random pruning and gradient rescaling as InfoBatch for the buckets with more than one sample.* The samples of $\bar{\mathcal{L}}_i < \bar{\mathcal{L}}$ are pruned as InfoBacth without bucketing. We show that training using the proposed PA can still achieve a similar result to training without pruning (see Sect. A.2 of our TR [7]).

Again, PA is general for accelerating the training of any NN-based selector. It is used as a plug-and-play module in our system.

4 Demonstration Scenarios

In our demonstration, we intend to show how audiences achieve TSAD model selection using our system, and how the proposed KDSelector helps them improve the accuracy and training speed of the NN-based selectors. We have prepared the 16 TSB-UAD [8] benchmark datasets used in [9] for the audiences. They can also test on their own data using our system.

Pipeline for TSAD model selection. Fig. 3 shows the interfaces of our system. In a nutshell, the user takes the following three steps to perform TSAD model selection by using the system.

(1) Selector learning. In this step, the user first uploads the historical data and configures the selector learning method. The system provides both NN-based and non-NN-based selectors. If the user chooses an NN-based selector, he/she can flexibly integrate the proposed modules in KDSelector into the learning. Once the user clicks on the "Start Learning" button, the system runs the learning method on the training data to learn the selector. The system provides visualization and evaluation functions for the user to validate the selector. It also allows the user to save and manage the learned selectors for easy reuse.

(2) Model selection. At this stage, users upload their time series of interest and apply the selector learned above to predict the best TSAD model for each series. The system also shows the votes of different models to help users understand how the selection is made.

(3) Anomaly detection. After selection, the user can interact with the system to run the selected model on the corresponding time series and visually assess the detection results, such as the anomaly scores predicted by the TSAD model and the overall performance evaluated using a metric of interest. The user can also run alternative models for a comparative analysis to validate the effectiveness of the model selection.

Superiority of KDSelector. The user can thoroughly evaluate our proposed KDSelector. For example, by experimenting using the train/test data and settings following the benchmark [9] and reporting the average accuracy (e.g., AUC-PR) and total training time, the user can draw three main conclusions: **(i) Effective knowledge-enhanced and data-efficient learning:** PISL and MKI can improve the accuracy of the learned selector with negligible training time overhead (see Table 1), while PA can save more training time than using full data and InfoBatch with almost lossless accuracy (see Table 2). **(ii) Architecture-agnostic:** KDSelector is effective for different selector architectures (see Table 3). **(iii) Better model selection solution:** By integrating KDSelector into existing NN-based selectors, e.g., ResNet as evaluated in `Ours` in Fig. 4, the user can obtain a model selection solution (i.e., `Ours`) that achieves superior performance across different domains (i.e., datasets) compared to existing solutions (see Fig. 4). We refer interested readers to Sect. B of our TR [7] for the detailed setups and full results.

References

[1] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*.

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

[4] 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 *SIGMOD*.

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

[6] Ziheng Qin, Kai Wang, Zangwei Zheng, Jianyang Gu, Xiangyu Peng, xu Zhao Pan, Daquan Zhou, Lei Shang, Baigui Sun, Xuansong Xie, and Yang You. 2024. InfoBatch: Lossless Training Speed Up by Unbiased Dynamic Data Pruning. In *ICLR*.

[7] KDSelector Technical Report. 2025. https://github.com/chenyuanTKCY/KDSelector/report.

[8] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. 2022. Anomaly detection in time series: a comprehensive evaluation. In *PVLDB*.

[9] 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. In *PVLDB*.

A Theoretical Analysis

This section shows the theoretical analysis results mentioned in Sect. 3 in detail, including the redundancy of the training samples that are similar to each other and have similar training losses in terms of their contributions to the selector learning, and the effectiveness of the proposed pruning-based acceleration in terms of achieving a similar result as training on full data.

A.1 Redundancy of Training Samples

Denote $\mathcal{L}_i = \mathcal{L}(F(\boldsymbol{X_i}; \Theta))$ the loss of $\boldsymbol{X_i}$ at current epoch, where $F(\boldsymbol{X}; \Theta)$ is the full model including both the selector f and the projections h_T and h_K. Θ is the set of parameters. Recall that NN-based selector is learned using SGD, by which the learning result (i.e., the update of the parameters) of each iteration depends on the gradient of $\mathcal{L}$ with respect to Θ, i.e.,

$$\Theta^{t+1} = \Theta^t - \eta \sum_i \nabla_\Theta \mathcal{L}_i, \tag{2}$$

where t is the current epoch, η is the learning rate, and $\nabla_\Theta \mathcal{L}_i$ is the gradient of $\mathcal{L}_i$ respect to Θ. According to the chain rule, we have

$$\nabla \mathcal{L}_i = \nabla_F \mathcal{L}_i \cdot \nabla_\Theta F_i, \tag{3}$$

where F_i represents $F(\boldsymbol{X}_i)$.

Suppose $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ are two training samples that are similar in themselves and in their losses, i.e.,

$$||\boldsymbol{X}_i - \boldsymbol{X}_j|| < \delta_X, \tag{4}$$

and

$$|\mathcal{L}_i - \mathcal{L}_j| < \delta_L, \tag{5}$$

where $\delta_X > 0$ and $\delta_L > 0$ are two small values. Based on Eq. (3), the difference of their contributions to the learning is

$$\begin{aligned} &||\nabla_\Theta \mathcal{L}_i - \nabla_\Theta \mathcal{L}_j|| \\ =&||\nabla_F \mathcal{L}_i \cdot \nabla_\Theta F_i - \nabla_F \mathcal{L}_j \cdot \nabla_\Theta F_j|| \\ =&||\nabla_F \mathcal{L}_i(\nabla_\Theta F_i - \nabla_\Theta F_j) + \nabla_\Theta F_j(\nabla_F \mathcal{L}_i - \nabla_F \mathcal{L}_j)||. \end{aligned} \tag{6}$$

By using the triangle inequality, we have

$$\begin{aligned} ||\nabla_\Theta \mathcal{L}_i - \nabla_\Theta \mathcal{L}_j|| \leq& ||\nabla_F \mathcal{L}_i|| \cdot ||\nabla_\Theta F_i - \nabla_\Theta F_j|| \\ &+ ||\nabla_\Theta F_j|| \cdot ||\nabla_F \mathcal{L}_i - \nabla_F \mathcal{L}_j)||. \end{aligned} \tag{7}$$

By using SGD, we have to ensure that the gradient in Eq. (3) is bounded (e.g., by gradient clipping), i.e.,

$$||\nabla_F \mathcal{L}_i|| \leq B_L, \tag{8}$$

and

$$||\nabla_\Theta F_i|| \leq B_F, \tag{9}$$

where B_L and B_F are the corresponding bounds.

Assume that $\nabla_\Theta F$ and $\nabla_F \mathcal{L}$ are *Lipschitz* for $\boldsymbol{X}$. According to Eq. 4 we have

$$||\nabla_\Theta F_i - \nabla_\Theta F_j|| \leq C_F ||\boldsymbol{X}_i - \boldsymbol{X}_i|| < C_F \delta_X, \tag{10}$$

and

$$||\nabla_F \mathcal{L}_i - \nabla_F \mathcal{L}_j)|| \leq C_L ||\boldsymbol{X}_i - \boldsymbol{X}_i|| < C_L \delta_X, \tag{11}$$

where $C_F > 0$ and $C_L > 0$ are two constants.

Substituting Eqs (8)-(11) back to Eq. (7), we get

$$||\nabla_\Theta \mathcal{L}_i - \nabla_\Theta \mathcal{L}_j|| \leq (B_L C_F + B_F C_L)||\boldsymbol{X}_i - \boldsymbol{X}_j|| < (B_L C_F + B_F C_L)\delta_X. \tag{12}$$

From Eq. (12), we can see that $||\nabla_\Theta \mathcal{L}_i - \nabla_\Theta \mathcal{L}_j|| \rightarrow 0$ when $\delta_X \rightarrow 0$, indicating that the samples close to each other have a similar contribution for updating the parameters of F (also including the selector f).

Moreover, assume that the loss is *strongly convex* (e.g., by L2 regularization). We have

$$\nabla_F \mathcal{L}_j \nabla_\Theta F_j(\boldsymbol{X}_i - \boldsymbol{X}_j) + \frac{\mu}{2}||\boldsymbol{X}_i - \boldsymbol{X}_j||^2 \leq \mathcal{L}_i - \mathcal{L}_j, \tag{13}$$

where $\mu > 0$ is a constant. Recall that $\nabla_F \mathcal{L}_j$ and $\nabla_\Theta F_j$ are bounded as Eqs. (8)-(9). Note that the second-order term with respect to $\boldsymbol{X}$ in Eq. (13) can be ignored when $\delta_X \rightarrow 0$. Thus, based on Eq.(5), when $|\nabla_F \mathcal{L}_j \nabla_\Theta F_j(\boldsymbol{X}_i - \boldsymbol{X}_j)| \leq |\mathcal{L}_i - \mathcal{L}_j|$ we get

$$||\boldsymbol{X}_i - \boldsymbol{X}_j|| \leq A\delta_L, \tag{14}$$

where

$$A = \frac{1}{||\nabla_F \mathcal{L}_j \nabla_\Theta F_j|| \cdot |\cos < \nabla_F \mathcal{L}_j \nabla_\Theta F_j, \boldsymbol{X}_i - \boldsymbol{X}_j > |}. \tag{15}$$

By substituting Eq. (14) to Eq. (12), we see that the similarity condition in the loss may provide a tighter bound for $||\nabla_\Theta \mathcal{L}_i - \nabla_\Theta \mathcal{L}_j||$ when $A\delta_L < \delta_X$.

In conclusion, the samples that are similar both in themselves and in their losses have almost ***identical contributions to the training***. Note that in our PA module, we use the average loss over the last $t - 1$ training epochs (i.e., $\bar{\mathcal{L}}_i$) to approximate the loss in the current epoch (i.e., $\mathcal{L}_i$) following [6] to achieve efficient pruning.

A.2 Effectiveness of Pruning-based Acceleration

Recall that in our proposed PA, we perform random pruning and gradient rescale for the samples of $\bar{\mathcal{L}}_i \geq \bar{\mathcal{L}}$ and falling in the same buckets, and the samples of $\bar{\mathcal{L}}_i < \bar{\mathcal{L}}$. Therefore, we can divide the full dataset, denoted as $\mathcal{D}$, into 2 disjoint subsets, including $\mathcal{D}_1$ that combines all buckets that need to prune and $\mathcal{D}_2$ other. Formally, we have

$$\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2. \tag{16}$$

Denote S_1 the subset of $\mathcal{D}_1$ after pruning and $\mathcal{S}$ the pruned dataset used for the current training epoch. We have

$$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{D}_2. \tag{17}$$

For each sample $\boldsymbol{X}_i$ to prune, the probability of pruning it is r, which is formulated as

$$Pr(\boldsymbol{X}_i) = r. \tag{18}$$

With gradient rescaling, the losses for the remaining samples in the pruned subsets, i.e., $\mathcal{S}_1$, are multiplied by the factor $1/(1 - r)$, which is the same as multiplying the corresponding gradients by the factor. Assume that all samples $\boldsymbol{X}$ are drawn from a continuous distribution $\rho(\boldsymbol{X})$. The objective of selector learning on the full dataset can be formulated as

$$\arg\min_\Theta \mathop{\mathbb{E}}_{\boldsymbol{X} \in \mathcal{D}}[\mathcal{L}(\boldsymbol{X}; \Theta)] = \int_{\boldsymbol{X} \in \mathcal{D}} \mathcal{L}(\boldsymbol{X}; \Theta)\rho(\boldsymbol{X})d\boldsymbol{X}. \tag{19}$$

After pruning using the proposed PA, the selector learning objective becomes

$$\begin{aligned} &\arg\min_\Theta \mathop{\mathbb{E}}_{\boldsymbol{X} \in \mathcal{S}}[\mathcal{L}'(\boldsymbol{X}; \Theta)] \\ =& \int_{\boldsymbol{X} \in \mathcal{S}_1} \frac{1}{1 - r} \mathcal{L}(\boldsymbol{X}; \Theta)\rho(\boldsymbol{X})d\boldsymbol{X} + \int_{\boldsymbol{X} \in \mathcal{D}_2} \mathcal{L}(\boldsymbol{X}; \Theta)\rho(\boldsymbol{X})d\boldsymbol{X}, \end{aligned} \tag{20}$$

where the first term on the right-hand side can be derived to

$$\begin{aligned} &\int_{X\in\mathcal{S}_1} \frac{1}{1-r}\mathcal{L}(X;\Theta)\rho(X)dX \\ =&\int_{X\in\mathcal{D}_1} \frac{1-Pr(X)}{1-r}\mathcal{L}(X;\Theta)\rho(X)dX \\ =&\int_{X\in\mathcal{D}_1} \mathcal{L}(X;\Theta)\rho(X)dX. \end{aligned} \tag{21}$$

By substituting Eq. (21) to Eq. (20), we get

$$\begin{aligned} &\arg\min_{\Theta} \mathop{\mathbb{E}}_{X\in\mathcal{S}}[\mathcal{L}'(X;\Theta)] \\ =&\int_{X\in\mathcal{D}_1} \mathcal{L}(X;\Theta)\rho(X)dX + \int_{X\in\mathcal{D}_2} \mathcal{L}(X;\Theta)\rho(X)dX \\ =&\int_{X\in\mathcal{D}} \mathcal{L}(X;\Theta)\rho(X)dX, \end{aligned} \tag{22}$$

which is consistent with the objective of learning on the full data without pruning as shown in Eq. (19). Therefore, we conclude that ***learning using the proposed pruning strategy can achieve a similar result as learning on the full data***.

B Experiments

This section shows the experimental setups and results in detail.

B.1 Experimental Setups

Datasets. We use the 16 TSB-UAD [8] subsets following [9], as described in Table 4. For a fair comparison, we use the recommended train/test split [9], where the training set is a combination of samples from all 16 datasets, while the time series from 14 subsets are used for test as shown in Fig. 4.

Baselines. The baseline TSAD model selection solutions used for comparison, as shown in Fig. 4, are representative approaches chosen from [9] that have shown competitive model selection performance. These solutions can be divided into two categories.

- **Non-NN-based methods.** This includes (i) *feature-based methods* that use the open-source tool TSFresh to extract features from the input time series, and train traditional machine learning classifiers on top of the features. The classifiers contain K nearest neighbors (`KNN`), support vector classifier (`SVC`), `AdaBoost`, and random forest classifier (`RandomForest`), and (ii) *kernel-based method* that refers to MiniRocket (abbreviated as `Rocket`) that uses multiple convolutional kernels generated at random in conjunction with a Ridge regression classifier.
- **NN-based methods.** This includes three convolution-based models, i.e., `ConvNet` that uses convolutional layers to learn spatial features, `ResNet` that uses ConvNet with residual connections, and `InceptionTime` that combines ResNets with kernels of multiple sizes, and an advanced `Transformer` architecture that corresponds to SiT-stem in [9].

We directly use the implementations open-sourced by [9] with their default parameters without specification for the baseline solutions. To achieve strong baselines, we run each baseline method using different subsequence lengths as $L \in \{16, 32, 64, 128, 256, 512, 768, 1024\}$ and report the best result on each dataset. All the evaluated baseline methods have been implemented in the current system.

TSAD models. To achieve a fair comparison, we use the 12 representative TSAD models chosen by [9] as the candidates in our TSAD model set. The models are described in Table 5. We use the open-source implementations and the default settings following [9]. All the TSAD methods have been integrated into our system.

Implementation details of KDSelector. The proposed KDSelector is implemented using Python 3.8 and 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. The default selector architecture is ResNet during our experiments, while InceptionTime and Transformer are also used to validate the effectiveness on different architectures. We keep the settings of KDSelector consistent with its underlying selector architecture for a fair comparison.

We use the base version of BERT [2] for text embedding, of which the parameters are frozen during the selector learning. The metadata used for MKI includes the length of the input series, the number of anomalies the series contains, the lasting time of these anomalies, and the description of application domain of the dataset (see Table 4). The following ***template is used to describe the metadata.***

"*This is a time series from dataset [Dataset name], [Description as Table 4]. The length of the series is [Length of series]. There are [Number of anomalies] anomalies in this series. The lengths of the anomalies are [Length of anomalies] (without this sentence if the number of anomalies is 0).*"

The projections h_T and h_K used in MKI are implemented using two multi-layer perceptions (MLPs), respectively. Each MLP has one hidden layer of 256 dimensions, with ReLU as the activation function. The output dimension H is selected from $\{64, 256\}$.

The hyper-parameters of PISL and MKI, including t_{soft}, α, and λ are selected from $\{0.2, 0.22, 0.25\}$, $\{0.2, 0.4, 1.0\}$, and $\{0.78, 1.0\}$, respectively. The temperature for the InfoNCE loss is set to 0.1.

For PA evaluation, we set the pruning ratio r to 0.8 (the same for InfoBatch). The number of bits used in LSH is set to 14, and the number of bins p is set to 8. Other settings are the same as InfoBatch.

Metrics. As shown in Sect. 4, we use AUC-PR to measure the accuracy of the selector. It is obtained by running the selected TSAD model on the corresponding time series and computing the metric using the true anomalies and the predicted anomaly scores of each data point. For training speed evaluation, we report the running time of the selector learning algorithm on the training dataset.

For a fair comparison, we exclude PA when comparing with existing solutions because they do not use any pruning strategy by default (Table 1, AUR-PR in Table 3, and Fig. 4), while to evaluate PA, we keep the proposed PISL and MKI in use to compare the learning results using different pruning strategies (Table 2 and saved time in Table 3).

B.2 Full Results

The full experimental results corresponding to Tables 1-3 and Fig. 4 are shown as follows.

Table 4: Dataset description [8].

Dataset	Description (Domain knowledge)
Dodgers	is a loop sensor data for the Glendale on-ramp for the 101 North freeway in Los Angeles and the anomalies represent unusual traffic after a Dodgers game.
ECG	is a standard electrocardiogram dataset and the anomalies represent ventricular premature contractions. We split one long series (MBA_ECG14046) with length about 1e7 to 47 series by first identifying the periodicity of the signal.
IOPS	is a dataset with performance indicators that reflect the scale, quality of web services, and health status of a machine.
KDD21	is a composite dataset released in a recent SIGKDD 2021 competition with 250 time series.
MGAB	is composed of Mackey-Glass time series with non-trivial anomalies. Mackey-Glass time series exhibit chaotic behavior that is difficult for the human eye to distinguish.
NAB	is composed of labeled real-world and artificial time series including AWS server metrics, online advertisement clicking rates, real time traffic data, and a collection of Twitter mentions of large publicly-traded companies.
SensorScope	is a collection of environmental data, such as temperature, humidity, and solar radiation, collected from a typical tiered sensor measurement system.
YAHOO	is a dataset published by Yahoo labs consisting of real and synthetic time series based on the real production traffic to some of the Yahoo production systems.
Daphnet	contains the annotated readings of 3 acceleration sensors at the hip and leg of Parkinson's disease patients that experience freezing of gait (FoG) during walking tasks.
GHL	is a Gasoil Heating Loop Dataset and contains the status of 3 reservoirs such as the temperature and level. Anomalies indicate changes in max temperature or pump frequency.
Genesis	is a portable pick-and-place demonstrator which uses an air tank to supply all the gripping and storage units.
MITDB	contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects studied by the BIH Arrhythmia Laboratory between 1975 and 1979.
OPPORTUNITY	is a dataset devised to benchmark human activity recognition algorithms (e.g., classiffication, automatic data segmentation, sensor fusion, and feature extraction). The dataset comprises the readings of motion sensors recorded while users executed typical daily activities.
Occupancy	contains experimental data used for binary classiffication (room occupancy) from temperature, humidity, light, and CO2. Ground-truth occupancy was obtained from time stamped pictures that were taken every minute.
SMD	is a 5-week-long dataset collected from a large Internet company. This dataset contains 3 groups of entities from 28 different machines.
SVDB	includes 78 half-hour ECG recordings chosen to supplement the examples of supraventricular arrhythmias in the MIT-BIH Arrhythmia Database.

Table 5: TSAD models used for model selection [8, 9].

TSAD model	**Description**
Isolation Forest (IForest)	This method constructs the binary tree based on the space splitting and the nodes with shorter path lengths to the root are more likely to be anomalies.
IForest1	same as IForest, but each data point (individually) are used as input.
The Local Outlier Factor (LOF)	This method computes the ratio of the neighboring density to the local density.
The Histogram-based Outlier Score (HBOS)	This method constructs a histogram for the data and the inverse of the height of the bin is used as the outlier score of the data point.
Matrix Profile (MP)	This method calculates as anomaly the subsequence with the most significant 1-NN distance.
NORMA	This method identifies the normal pattern based on clustering and calculates each point's effective distance to the normal pattern.
Principal Component Analysis (PCA)	This method projects data to a lower-dimensional hyperplane, and data points with a significant distance from this plane can be identified as outliers.
Autoencoder (AE)	This method projects data to the lower-dimensional latent space and reconstructs the data, and outliers are expected to have more evident reconstruction deviation.
LSTM-AD	This method build a non-linear relationship between current and previous time series (using Long-Short-Term-Memory cells), and the outliers are detected by the deviation between the predicted and actual values.
Polynomial Approximation (POLY)	This method build a non-linear relationship between current and previous time series (using polynomial decomposition), and the outliers are detected by the deviation between the predicted and actual values.
CNN	This method build a non-linear relationship between current and previous time series (using convolutional Neural Network), and the outliers are detected by the deviation between the predicted and actual values.
One-class Support Vector Machines (OCSVM)	This method fits the dataset to find the normal data's boundary.

Table 6: Full results of PISL and MKI.

Method	Standard	+ PISL	+ MKI	+ PISL & MKI
Daphnet	0.2873	0.2873	**0.3014**	0.2873
ECG	0.6624	**0.6897**	0.6259	**0.6897**
Genesis	**0.3617**	**0.3617**	**0.3617**	**0.3617**
GHL	0.1932	**0.3071**	0.2303	0.3035
IOPS	0.2843	0.2843	**0.309**	**0.309**
KDD21	0.2902	0.3875	0.3125	**0.4426**
MGAB	**0.614**	**0.614**	**0.614**	**0.614**
MITDB	0.3355	**0.4856**	0.3123	**0.4856**
NAB	**0.3319**	**0.3319**	0.3137	0.3279
OPPORTUNITY	0.3886	0.3886	**0.4031**	0.3995
SensorScope	0.335	0.335	0.335	**0.3844**
SMD	0.4561	0.4561	0.4501	**0.4576**
SVDB	0.6212	0.6212	0.6215	**0.6337**
YAHOO	0.737	0.737	0.7535	**0.7558**
Average AUC-PR	0.421	0.449	0.424	**0.461**
Total training Time (mins)	281.9	**280.42**	282.05	282.03

Table 7: Full results of PA.

Method	Full data	+ InfoBatch	+ PA (Ours)
Daphnet	0.2873	0.2724	**0.2933**
ECG	**0.6897**	**0.6897**	**0.6897**
Genesis	**0.3617**	**0.3617**	**0.3617**
GHL	0.3035	**0.3071**	**0.3071**
IOPS	0.3090	0.2843	**0.3762**
KDD21	**0.4426**	0.4304	0.3941
MGAB	**0.6140**	**0.6140**	**0.6140**
MITDB	**0.4856**	**0.4856**	**0.4856**
NAB	0.3279	0.3398	**0.3643**
OPPORTUNITY	**0.3995**	0.3725	0.3928
SensorScope	**0.3844**	**0.3844**	0.3132
SMD	0.4576	**0.4602**	0.4277
SVDB	**0.6337**	0.6213	0.6118
YAHOO	**0.7558**	0.7481	0.7070
Average AUC-PR	**0.461**	$0.455_{\downarrow 0.006}$	$0.452_{\downarrow 0.009}$
Total training time (mins)	282.03	$171.73_{\downarrow 39.1\%}$	$\mathbf{117.72}_{\downarrow 58.3\%}$

Table 8: Full results on different architectures.

Architecture	ResNet		InceptionTime		Transformer	
Method	Default	+ KDSelector	Default	+ KDSelector	Default	+ KDSelector
Daphnet	**0.2873**	**0.2873**	**0.3129**	0.2990	**0.3070**	0.2804
ECG	0.6624	**0.6897**	0.6187	**0.6917**	0.6176	**0.6897**
Genesis	**0.3617**	**0.3617**	0.1796	**0.3617**	0.2957	**0.3617**
GHL	0.1932	**0.3035**	0.2637	**0.3071**	0.2851	**0.3349**
IOPS	0.2843	**0.3090**	0.2235	**0.3045**	0.2714	**0.2845**
KDD21	0.2902	**0.4426**	0.2987	**0.4010**	0.3616	**0.3799**
MGAB	**0.6140**	**0.6140**	**0.6140**	**0.6140**	0.6140	**0.6140**
MITDB	0.3355	**0.4856**	0.4298	**0.5089**	0.3739	**0.4856**
NAB	**0.3319**	0.3279	0.3019	**0.3701**	**0.4195**	0.3530
OPPORTUNITY	0.3886	**0.3995**	**0.3849**	0.3842	**0.4013**	0.3656
SensorScope	0.3350	**0.3844**	0.3350	**0.3545**	0.3191	**0.3312**
SMD	0.4561	**0.4576**	**0.4832**	0.4684	**0.4819**	0.4444
SVDB	0.6212	**0.6337**	0.5940	**0.6391**	0.6369	**0.6372**
YAHOO	0.7370	**0.7558**	**0.7616**	0.7457	0.7091	**0.7384**
Average AUC-PR	0.4213	**0.4609**	0.4144	**0.4607**	0.4353	**0.4500**
Improved AUC-PR	0.0396		0.0463		0.0147	
Total training time (mins)	282.03	**117.72**	292.99	**85.09**	343.94	**88.85**
Saved time (%)	58.3%		70.96%		74.17%	

Table 9: Full results of different model selection solutions.

Method	KNN	SVC	AdaBoost	RandomForest	ConvNet	ResNet	InceptionTime	Transformer	Rocket	**Ours**
Daphnet	0.1197	0.1888	0.1197	0.1854	0.2445	0.2873	**0.3129**	0.3070	0.1397	0.2873
ECG	0.6842	0.6842	0.6842	0.6842	0.6154	0.6624	0.6187	0.6176	0.6842	**0.6897**
Genesis	0.0017	0.0017	0.0017	0.0017	**0.3617**	**0.3617**	0.1796	0.2957	0.1796	**0.3617**
GHL	0.3071	0.3042	0.3071	**0.3335**	0.2571	0.1932	0.2637	0.2851	0.3068	0.3071
IOPS	0.0734	0.1807	0.0734	0.0686	0.2732	0.2843	0.2235	0.2714	0.1430	**0.3090**
KDD21	0.4132	0.3154	0.3876	0.3808	0.3874	0.2902	0.2987	0.3616	0.3372	**0.4426**
MGAB	**0.6140**	0.0122	0.0122	0.0122	**0.6140**	**0.6140**	**0.6140**	**0.6140**	**0.6140**	**0.6140**
MITDB	**0.4856**	**0.4856**	**0.4856**	**0.4856**	0.4132	0.3355	0.4298	0.3739	0.4562	**0.4856**
NAB	0.1867	0.2783	0.2127	0.1739	0.3263	0.3319	0.3019	**0.4195**	0.3558	0.3730
OPPORTUNITY	0.3238	0.3116	0.3157	0.3092	**0.4080**	0.3886	0.3849	0.4013	0.3472	0.3995
SensorScope	0.2857	0.2941	0.2755	0.2941	0.3350	0.3350	0.3350	0.3191	0.2521	**0.3844**
SMD	0.2479	0.3497	0.2278	0.3358	**0.5004**	0.4561	0.4832	0.4819	0.3175	0.4576
SVDB	0.6317	0.6317	0.6317	0.6317	0.6043	0.6212	0.5940	0.6369	0.6391	**0.6408**
YAHOO	0.3144	0.1934	0.2718	0.2681	0.7385	0.7370	**0.7616**	0.7091	0.3474	0.7558