PASTA: Neural Architecture Search for Anomaly Detection in Multivariate Time Series

Patara Trirat, *Member, IEEE* and Jae-Gil Lee, *Senior Member, IEEE*

***Abstract*—Time-series anomaly detection uncovers rare errors or intriguing events of interest that significantly deviate from normal patterns. In order to precisely detect anomalies, a detector needs to capture intricate underlying temporal dynamics of a time series, often in multiple scales. Thus, a fixed-designed neural network may not be optimal for capturing such complex dynamics as different time-series data require different learning processes to reflect their unique characteristics. This paper proposes a *P*rediction-based neural *A*rchitecture *S*earch for *T*ime series *A*nomaly detection framework, dubbed *PASTA*. Unlike previous work, besides searching for a connection between operations, we design a novel search space to search for optimal connections in the temporal dimension among recurrent cells within/between each layer, i.e., *temporal connectivity*, and encode them via *multi-level configuration encoding* networks. Experimental results from both real-world and synthetic benchmarks show that the discovered architectures by *PASTA* outperform the second-best state-of-the-art baseline by around 13.6% in the enhanced time-series aware F_1 score on average, confirming that the design of temporal connectivity is critical for time-series anomaly detection.**

***Index Terms*—Neural architecture search, AutoML, time-series anomaly detection, temporal connectivity encoding.**

I. Introduction

TIME-SERIES anomaly detection (TSAD) aims at determining abnormal patterns or deviant behaviors that are extremely rare in a time series. For half a century [1], it has served as one of the fundamental tasks in data mining and has been an active research area with various applications, e.g., fraud detection and fault diagnosis. Early endeavors [2], [3] solved this problem using statistical or machine learning methods. However, the massive amount of *multivariate* time series generated nowadays makes the problem more challenging for the traditional approaches. In addition to the well-known challenges, e.g., high complexity of data and diverse types of anomalies, a model for multivariate TSAD also needs to capture intricate temporal dynamics and relationships within/between multiple variables, i.e., sensors [4], [5].

In recent years, unsupervised deep learning (DL) models have achieved state-of-the-art performance in TSAD, surpassing traditional methods by a large margin. Specifically, recurrent-based layers [6], [7], [8], [9], [10] were adopted with generative networks, such as autoencoder (AE), to capture the nonlinear complexity of high-dimensional multivariate time series. Like other domains, ensemble learning [11], [12] was also adopted to enhance the model's robustness when deciding whether a particular instance is anomalous. Moreover, multi-resolution learning[1] [13], [14], [15] was used to enhance the representations of normal patterns on different scales and learn long-range and diverse temporal dependencies. However, such efficient DL-based detectors rely laboriously on human experts to design the neural architecture, tune the hyperparameters, and select a proper anomaly scoring function, impeding its wider adoption. These efforts are usually time-consuming, less systematic trial-and-error, and the resulting solutions may still be suboptimal given that different time series have different unique characteristics to be considered.

Neural architecture search (NAS) was proposed to reduce human efforts by automatically designing deep neural networks and sometimes selecting their hyperparameters accordingly [16]. Despite the substantial progress of NAS techniques in several tasks, NAS for time-series data is still underexplored with only a few papers on classification [17], [18], [19] and forecasting [20], [21], [22] problems. In addition, we cannot directly use them for TSAD due to the following challenges.

(1) Lack of a suitable search space for TSAD. Most existing search spaces [23] consist of convolutional networks (CNN), whereas a few include recurrent networks (RNN). Still, the RNN-based spaces are designed by following the CNN-based spaces to find a new connection between operations within a micro-level motif. Thus, other essential settings for RNN-based AEs, such as macro configurations and layer hyperparameters, are ignored. Moreover, to achieve state-of-the-art results, information on *how recurrent cells are connected in the time dimension* is vital for TSAD using RNN-based architectures to model multi-scale temporal dependency [11], [13], [14]. We call such connection *temporal connectivity*, i.e., the connection between different time steps. As in Fig. 1, we argue that the search space for TSAD should cover not only architecture configurations, e.g., layer hyperparameters, but also the TSAD task-specific settings for global configurations and temporal connectivity because the operation-only search space is insufficient to build high-performance TSAD [4], [24].

(2) Lack of multi-level configuration encoding for RNN-based AEs. Current studies on architecture encoding for NAS

Manuscript received 18 December 2023; revised 29 July 2024; accepted 5 November 2024. Date of publication dd MM yyyy; date of current version dd MM yyyy. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2020-II200862, DB4DL: High-Usability and Performance In-Memory Distributed DBMS for Deep Learning and No. RS-2022-II220157, Robust, Fair, Extensible Data-Centric Continual Learning). *(Corresponding author: Jae-Gil Lee.)*

Patara Trirat and Jae-Gil Lee are with the School of Computing, KAIST, Daejeon 34141, Republic of Korea (e-mail: {patara.t, jaegil}@kaist.ac.kr).

Digital Object Identifier 10.1109/TETCI.2024.0000000

[1]In this paper, multi-resolution or multi-scale learning broadly refers to learning or extracting multiple time-scale information.

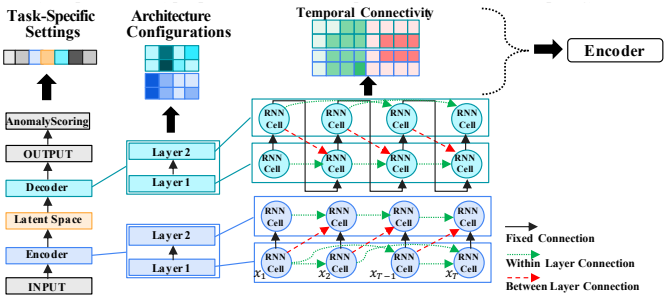


Fig. 1. Proposed multi-level search space and architecture encoding scheme.

mainly focus on layer-level or operation-level connections to distinguish architectures as a directed acyclic graph (DAG) in the search space [25], [26]. This simple single-level DAG-based encoding will not be able to differentiate the RNN-based AE models having different *cell-level*[2] configurations, e.g., temporal connectivity. Hence, with the hierarchical structure of our search space, a novel way to encode these *multi-level configurations*—from network to layer to cell—effectively is indispensable for the subsequent search process.

Motivated by the above TSAD challenges, we propose a novel **P**rediction-based neural **A**rchitecture **S**earch framework for **T**ime-series **A**nomaly detection called ***PASTA***. To tackle the first challenge, we design and construct a new search space based on the insights from deep TSAD [11], [13], [14], [15], [27] and preliminary experiments (see §III) that the temporal connectivity between recurrent cells and layers significantly affects TSAD performance. Accordingly, with the introduction of temporal connectivity subspace, the search method can systematically learn how to form the connections for capturing multiple resolutions of complex temporal dynamics in multivariate time series instead of relying solely on randomness [11], [14], which is significant for the anomaly detection task.

Although having temporal connectivity subspace provides better detection performance, it comes with the cost of having a larger search space. This motivates us to figure out how to incorporate it into the NAS setting effectively. As architecture encoding plays a vital role in NAS [28], [29], for the second challenge, we propose a *multi-level configuration encoding* based on unsupervised architecture representation learning to accompany the new search space by aggregating task-specific settings, network configurations, and temporal connectivity subspaces. The encoder will map these multi-level configurations into a learned latent space. It improves the embedding quality by learning the hierarchical structure of connections in the neural networks and boosts the search efficiency of a performance predictor by using high-quality low-dimensional learned latent spaces instead of massive raw representations.

Finally, we realize the *PASTA* framework by bridging the newly proposed two components with a performance predictor-guided search method. The predictor learns to predict the architecture performance based on the encoded representations from the multi-level configuration encoder and guides the search process with its predicted performance given a set of encoded architectures on the search space.

[2]Throughout this paper, the term "cell" refers to a recurrent cell at a time step within a recurrent layer.

Our **main contributions** are listed below.

- We propose a new search space tailored for TSAD, including TSAD-specific settings, architecture configurations, and various temporal connectivity types between recurrent cells. This search space allows a search method to search on different levels of an anomaly detection model more flexibly.
- We introduce a new architecture encoding technique for prediction-based NAS that learns the relationships between different levels of TSAD model configurations, including the temporal connectivity between/within recurrent layers. This encoding enables a search method to find high-performing models based on the encoded features more effectively.
- We propose a novel prediction-based NAS for TSAD, *PASTA*, based on the newly introduced search space and multi-level configuration encoding. To verify the effectiveness of *PASTA*, we conduct thorough experiments on both synthetic and real-world benchmarks compared with diverse handcrafted TSAD, AutoML, and random search. On average, the models discovered by *PASTA* increase the enhanced time-series aware F_1 [30] scores by at least 13.6%.

The paper is organized as follows. Section II reviews related work. Section III explains problem statements and the proposed *PASTA*. Then, Section IV presents the experimental setup and results. Finally, Section V concludes this study.

II. Related Work

A. Time-Series Anomaly Detection (TSAD)

Although we can perform TSAD in (semi-)supervised settings, unsupervised methods are the most prevalent due to the insufficiency of labeled data. Traditionally, we can classify TSAD approaches into statistical and machine learning-based methods [3], [31], [32]. Recently, several DL-based studies [4], [33], [34] have shown to be superior to the traditional methods. The most well-known deep TSAD are prediction-based [6], [35], [36] and reconstruction-based [7], [8], [11], [12], [14], [15], [27], [35], [37], [38], [39], [40], [41], [42], [43], [44], [45] methods. The former uses prediction (or forecasting) errors as anomaly scores, while the latter uses reconstruction errors. Previous research has shown that reconstruction-based methods significantly outperform forecasting-based ones. Among different network types, Transformer-based models [41], [42], [43], [46] have achieved state-of-the-art results in the past few years. However, recent findings [34], [47] reveal that a simple reconstruction-based LSTM model can outperform state-of-the-art Transformer-based models under an appropriate experimental setup.

There have been several attempts to enhance the effectiveness of reconstruction-based methods using RNNs. Kieu et al. [11] introduce TSAD models based on ensembles of recurrent AEs with sparsely-connected RNNs to create multiple AEs having different temporal connection structures. This framework reduces the overfitting of individual AEs, while promoting diversity and robustness. The recurrent reconstructive network [27] uses skip transitions to enhance the connectivity between recurrent units by reducing the path length for a more efficient information flow, resulting

in faster convergence, mitigating gradient-related issues, and more accurate detection.

From multi-resolution learning perspectives, THOC [13] uses a dilated RNN with skip connections to capture temporal dynamics at multiple scales. RAMED [14] further incorporates ensembles of multiple decoders with different decoding lengths to capture anomalies that may be evident at different granularities. On top of ensembling, RAMED [14] proposes coarse-to-fine fusion mechanism to integrate lower-resolution information into higher-resolution decoders to improve the long-range decoding capabilities, allowing the model to capture both local and global temporal patterns. Lately, Qingning et al. [15] employ an attention-based recurrent AE model that captures features at various scales through a hierarchically connected recurrent encoder. These multi-scale approaches enable the models to understand both fine-grained and coarse-grained temporal patterns, improving their ability to detect anomalies that manifest differently across various temporal granularities compared to traditional RNNs that focus only on single-scale features and short-term dependencies likely insufficient to detect anomalies occurring at multiple scales.

In line with recent findings, this paper focuses on finding the best *ensembles of multi-resolution RNN architectures* with the unsupervised reconstruction-based method, especially for multivariate TSAD, which has consistently shown state-of-the-art performance.

B. Neural Architecture Search (NAS)

NAS was initiated as neuroevolution [48], [49] in the 1990s and has become popular in the last five years thanks to hardware advancements [50], [51]. A NAS framework requires *three* core components. First, we need a *search space* to define which architectures can be represented in principle. It can be either convolutional-based [50], [52], [53], [54], [55], [56], [57], [58], [59], [60], [61] or recurrent-based search space [50], [54], [55], [62], [63]. Recently, Transformer-based search spaces [64], [65], [66], [67] have started drawing attention from the computer vision and natural language processing (NLP) communities. However, as discussed above, RNN-based models can outperform Transformer-based models for TSAD. Second, a *search strategy* is developed to explore the search space. Commonly-used search strategies include reinforcement learning [50], [51], [68], evolutionary algorithms [69], [70], gradient-based methods [54], [71], [72], Bayesian optimization [73], [74], [75], and prediction-based methods [76], [77], [78], [79], [80]. Among these strategies, prediction-based NAS has proven to be the most efficient [77], [81]. Third, to evaluate candidate architectures on the downstream task with unseen data, we need a *performance estimator*. Early NAS [69], [82] evaluated candidate architectures by training from scratch with enormous computational costs, restricting the wide adoption in real-world environments. Therefore, many attempts tried to solve this issue by early stopping, low-fidelity training with proxy datasets or architectures [83], weight sharing [54], [71], [84], and, more recently, surrogate models for performance prediction, resulting in increased sample efficiency and decreased computation burden.

Although NAS has demonstrated its successes in diverse tasks [26], e.g., adversarial learning [85], [86], NLP [59], [64], [87], and spatial-temporal prediction [60], [61], NAS for time series is still underexplored despite the fact that there are various tasks in time-series data mining [88]. There are a few studies for classification [17], [18], [19] and forecasting [20], [62], [89] tasks. Even though these studies also propose new search spaces, they adopt existing motifs from previous studies in other domains and do not consider the temporal connectivity when designing search spaces for temporal modeling. In addition, since existing RNN-based studies for other tasks [26], [59], [63], [90], [91], [92] focus on finding neural networks using single-level macro or micro search spaces without considering temporal connection between RNN cells required for TSAD, we need to design the new search space for effective TSAD. Meanwhile, automated model selection methods [93], [94] focus only on building a pipeline from a collection of existing models, not searching for new architectures. A comprehensive review can be found in the survey papers [16], [95], [96], [97].

C. Neural Architecture Encoding

Early NAS [50], [55], [69], [98] used discrete encodings (e.g., text sequence or adjacency matrix) to represent design choices in search space. Unfortunately, these simple encodings faced the scalability problem in large search spaces [97]. Due to the impact of architecture representation on the search and final task-specific performance [25], recent studies seek for more efficient ways of architecture encoding to maximize the final performance and reduce the search cost, including continuous relaxation-based [54], learning-based structure-aware [28], [76], [99], [100] or computation-aware [29], [78], [101], and path-based [75], [79], [102] encodings. As reported by Yan et al. [28], the learning-based encodings have proven to be the optimal solution, especially for those trained without supervision signals (i.e., accuracy) from a search process, because the supervision signals could bias the architecture representation learning and search direction.

III. Proposed Framework: PASTA

This section formulates the problem of NAS for TSAD and describes the details of *PASTA*, including the search space, multi-level configuration encoding, and predictor-guided search. An overview of *PASTA* is illustrated in Fig. 2.

A. Problem Formulation

We contend that NAS for TSAD has two key characteristics. First, the architecture in the RNN-based AE—notably, the *temporal connectivity* between recurrent cells—needs to be adaptive for the given dataset to achieve competitive performance. Second, TSAD also requires the designs of anomaly scoring and corresponding objective functions. Formally, we outline the problem of NAS for TSAD as follows. Suppose that a model for TSAD consists of three components: the anomaly detection-specific (hereafter, task-specific) settings S, the RNN-based AE architecture A, and the temporal connectivity C. We denote a model as a triple (S, A, C).

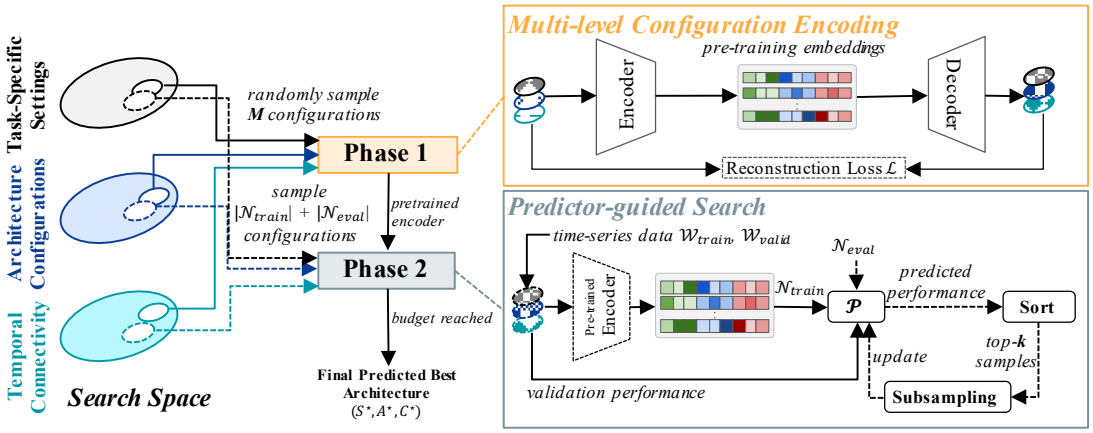


Fig. 2. Overall procedure of *PASTA*. **Phase 1** first randomly samples M configurations from the search space for pre-training the multi-level configuration encoder. **Phase 2** then samples $|\mathcal{N}_{train}|+|\mathcal{N}_{eval}|$ configurations and uses the pre-trained encoder to get their embeddings for the subsequent search process.

Given a time series of length T having d-dimensional vectors $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$, $\mathbf{x}_t \in \mathbb{R}^d$, the input of ($S$, A, C) is a sequence of time windows $\mathcal{W} = \{W_1, \ldots, W_{T-K+1}\}$, $W_t = \{\mathbf{x}_t, \ldots, \mathbf{x}_{t+K-1}\}$ of length K with stride 1 normalized and split from $\mathbf{X}$, following Kim et al. [103]. The goal of (S, A, C) is to predict the anomaly label $\hat{y}_t \in \{0, 1\}, t > T$ for the test windows $\mathcal{W}_{\text{test}}$. The labels are obtained by comparing anomaly scores $S_{score}(W_t)$ with a predefined threshold δ; $\hat{y}_t = 1$ if $S_{score}(W_t) > \delta$, otherwise 0.

Then, let the triple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{C}$) denote the search space of TSAD models, where $\mathcal{S}$ denotes the task-specific subspace, $\mathcal{A}$ denotes the architecture subspace, and $\mathcal{C}$ denotes the temporal connectivity subspace. Given the training set $\mathcal{W}_{\text{train}}$ and validation set $\mathcal{W}_{\text{valid}}$, we aim to find the optimal model ($S^\star$, $A^\star$, $C^\star$) trained on $\mathcal{W}_{\text{train}}$ that maximizes the predicted validation performance on $\mathcal{W}_{\text{valid}}$ (e.g., F_1 score) by a predictor $\mathcal{P}$,

$$(S^\star, A^\star, C^\star) = \underset{S\in\mathcal{S}, A\in\mathcal{A}, C\in\mathcal{C}}{\arg\max}\ \mathcal{P}(S, A, C). \tag{1}$$

S is a task-specific setting for an architecture A with a temporal connectivity C.

B. Tailored Search Space for TSAD

Due to the lack of intrinsic search space for TSAD, we newly design the search space for the RNN-based AEs, following the latest successful deep TSAD models [4], [11], [13], [14], which can learn complex temporal dynamics in multiple scales and construct an ensemble of multiple AEs to increase the robustness of TSAD. As summarized in Table I, it is a triple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{C}$) which represents the task-specific setting, architecture configuration, and temporal connectivity. Details and formal descriptions are as follows. Due to the space limit, we skip the details of architecture configurations ($\mathcal{A}$) subspace as they are already well-known.

1) Task-Specific Settings: We design the task-specific subspace $\mathcal{S}$ to accompany the global-level choices and constraints *for the entire TSAD model*, which affect the detection performance [4], [24]. The possible configurations are as follows.

Anomaly Scoring Function. The scoring function S_{score} of a model (S, A, C) determines how likely a value at a particular time step is anomalous. In reconstruction-based methods, S_{score} compares the original input and its reconstructed version. Formally, the anomaly scoring functions in our search space are defined by

$$\begin{aligned}
S_{score}^{\text{absolute error}} &= |W_t - \hat{W}_t| \\
S_{score}^{\text{squared error}} &= (W_t - \hat{W}_t)^2 \\
S_{score}^{\text{normal distribution}} &= (e_t - \mu_e)^T \Sigma_e^{-1} (e_t - \mu_e) \\
S_{score}^{\text{Mahalanobis distance}} &= \sqrt{(e_t - \mu_t)\Sigma^{-1}(e_t - \mu_t)^T} \\
S_{score}^{\text{max normalized error}} &= \max_i a_t^i, \ \text{where}\ a_t^i = \frac{e_t^i - \tilde{\mu}_i}{\tilde{\sigma}_i}.
\end{aligned} \tag{2}$$

W_t is the original input time-series windows. $\hat{W}_t$ is the reconstructed version. $e_t = W_t - \hat{W}_t$. $N(\mu_e, \Sigma_e)$ is the normal distribution estimated by e_t. e_t^i is the error at time t of a sensor/variable i. $\tilde{\mu}_i$ and $\tilde{\sigma}_i$ are the median and inter-quartile range across time steps of the e_t^i values, respectively.

Output Direction. In addition to the `forward` output direction, which is the same as the input, as shown in previous work [11], [14], [27], training with the `backward` (i.e., reverse) output direction is beneficial for TSAD. Thus, we also include this option in the search space.

Loss Function. The loss functions S_{loss} are defined by

$$\begin{aligned}
S_{loss}^{\text{MAE}} &= \frac{1}{T}\sum_{t=1}^{T} |W_t - \hat{W}_t| \\
S_{loss}^{\text{MSE}} &= \frac{1}{T}\sum_{t=1}^{T} (W_t - \hat{W}_t)^2 \\
S_{loss}^{\text{LogCosh}} &= \frac{1}{T}\sum_{t=1}^{T} log(cosh(\hat{W}_t - W_t)),
\end{aligned} \tag{3}$$

where W_t is the original input time-series windows, $\hat{W}_t$ is the reconstructed version, and T is the length of a time series.

Number of Autoencoders. Due to the robustness of ensemble learning [11], [14], we allow the search method to explore whether learning with multiple AE networks is really helpful.

Recurrent Cell Types. Since our focus is mainly on searching for better temporal connectivity and commonly-used recurrent layers are powerful enough to achieve state-of-the-art

TABLE I
SUMMARY OF PASTA'S SEARCH SPACE.

Subspace	Configuration Type	Option	Size
Task-specific Settings ($\mathcal{S}$)	Scoring Function	[absolute error, squared error, normal distribution, Mahalanobis distance, max normalized error]	5
	Output Direction	[forward, backward]	2
	Loss Function	[MSE, MAE, LogCosh]	3
	Noise Injection	[None, Gaussian Noise with stddev=0.2]	2
	# Autoencoders	[1, 2, 3, 4, 5]	5
	Latent Space Size	[8, 16, 32, 64, 128]	5
	Recurrent Cell Type	[RNN, LSTM, GRU]	3
Architecture Configurations ($\mathcal{A}$)	# Layers	[1, 2, 3, 4, 5]	5
	# Hidden Units	[16, 32, 64, 128, 256]	5
	Activation Function	[tanh, sigmoid, relu]	3
	Dropout	[None, Dropout with 20% probability]	2
Temporal Connectivity ($\mathcal{C}$)	Within Layer	[Default, Uniform Skip, Dense Random Skip, Sparse Random Skip]	4
	Between Layer	[Default, Full Connection, Feedback Transition, Skip Transition]	4

performance, we only include the vanilla RNN [104], LSTM [105], and GRU [106] in our search space, denoted as S_{cell}.

2) Architecture Configurations: The architecture configuration subspace $\mathcal{A}$ is a set of hyperparameters *for each layer* in an AE network. This subspace directly accounts for the representation power of the neural architecture. Here, we consider a number of hidden units, an activation function, and the use of dropout.

3) Temporal Connectivity: Given the identical task-specific and architecture configurations, we conduct preliminary experiments and find that only adjusting the connection between cells can significantly affect the TSAD performance. For example, the F_1 score increases from 0.697 to 0.809 by changing the default connection to dense random skip with feedback transition. The preliminary results of 16 types of temporal connectivity are presented in Fig. 3. Accordingly, we design this temporal connectivity subspace $\mathcal{C}$ to systematically find the optimal connections between recurrent cells for modeling the temporal dynamics of time series on multiple scales.

As exemplified in Fig. 4, we include two types—within-layer and between-layer—of connectivity *for each cell* in a layer. For the *within-layer* connection, we include three special connections successfully applied for TSAD [11], [13], [14]: uniform skip, dense random skip, and sparse random skip. The uniform skip connects time steps with a constant rate of 2^{l-1} in l-th layer (see Fig. 4b). For the random skips, the dense one randomly forms an extra connection on top of the default connection, while the sparse one fully forms the random connections for the entire layer.

Within-Layer Temporal Connectivity. This type of connectivity designates *how different cells in the same layer are connected*. Let $r_{t,l}$ be a recurrent cell r at time t in any layer l. The four choices for within-layer temporal connectivity are

$$\begin{aligned} r_{t,l}^{\text{default}} &= S_{\text{cell}}(\mathbf{h}_{t,l-1}, \mathbf{h}_{t-1,l}) \\ r_{t,l}^{\text{uniform skip}} &= S_{\text{cell}}(\mathbf{h}_{t,l-1}, \mathbf{h}_{t-2^{l-1},l}) \\ r_{t,l}^{\text{dense rand.}} &= \frac{S_{\text{cell}}(\mathbf{h}_{t,l-1}, \mathbf{h}_{t-1,l}) + S_{\text{cell}}(\mathbf{h}_{t,l-1}, \mathbf{h}_{t-L,l})}{2} \\ r_{t,l}^{\text{sparse rand.}} &= \frac{S_{\text{cell}}(\mathbf{h}_{t,l-1}, w_1\mathbf{h}_{t-L_1,l}) + S_{\text{cell}}(\mathbf{h}_{t,l-1}, w_2\mathbf{h}_{t-L_2,l})}{|w_1| + |w_2|}, \end{aligned} \tag{4}$$

where $\mathbf{h}_{t,l}$ is a hidden state of layer l at time t, $\mathbf{h}_{0,l} = \mathbf{0}$, and $\mathbf{h}_{t,0} = \mathbf{x}_t$. In the random skips (denoted as rand.) [11], [14], L is the skip length randomly sampled from $[1, 10]$, and w_1, w_2 are the weight coefficients randomly sampled from $\{(1,0),(0,1),(1,1)\}$ to ensure that the cell is connected to at least one previous cell.

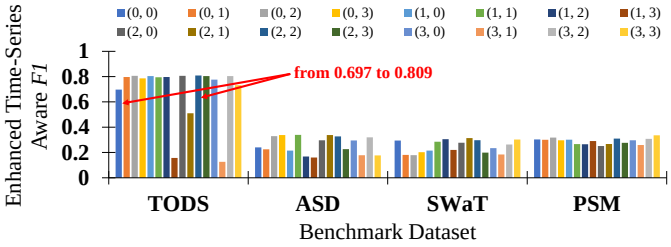


Fig. 3. F_1 scores between different types of temporal connectivity. Numbers in parenthesis are the indices indicating the type of connections in the search space. Specifically, {**0**: default connections, **1**: (full connection, uniform skip), **2**: (feedback transition, dense random skip), **3**: (skip transition, sparse random skip)}. The experimental setup is in the supplementary material.

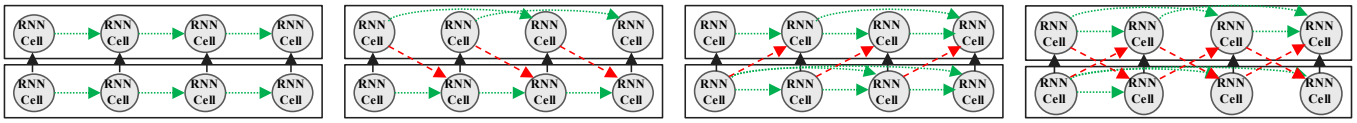
Similarly, regarding the *between-layer* connection, we include three special connections reported to be helpful in temporal modeling [27]: feedback transition, skip transition, and full connection, as shown in Fig. 4b, 4c, and 4d, respectively.

Between-Layer Temporal Connectivity. This type of connectivity represents *how cells in a layer connect to other cells in the upper or lower layers*. Let $r_{t,l}$ be a recurrent cell r in layer l at time step t. For between-layer temporal connectivity, the four choices are defined by

$$\begin{aligned} r_{t,l}^{\text{default}} &= S_{\text{cell}}(\mathbf{h}_{t,l-1}) \\ r_{t,l}^{\text{full connection}} &= S_{\text{cell}}(\mathbf{h}_{t-1,l-1}, \mathbf{h}_{t-1,l+1}) \\ r_{t,l}^{\text{feedback transition}} &= S_{\text{cell}}(\mathbf{h}_{t-1,l+1}) \\ r_{t,l}^{\text{skip transition}} &= S_{\text{cell}}(\mathbf{h}_{t-1,l-1}), \end{aligned} \tag{5}$$

where $\mathbf{h}_{t,l}$ is a hidden state of the layer l at time t, $\mathbf{h}_{0,l} = \mathbf{0}$, and $\mathbf{h}_{t,0} = \mathbf{x}_t$.

In particular, the dense and sparse random skip connections are formed independently for each layer, making the subspace size dependent on the input sequence length $K = |W_t|$. That is, as the input sequence length increases, the search space size increases exponentially, because the number of possible choices for each timestamp is multiplied to the search space size. Since K is usually around 100 in practice [8], [41] and



(a) Default. (b) Uni. + Feedback Transition. (c) Dense Rand. + Skip Transition. (d) Sparse Rand. + Full Connection.

Fig. 4. Examples of within (dotted-green) and between (dashed-red) layer temporal connectivity. Uni. means uniform skip. Rand. means random skip.

there can be up to two connections per cell within the same layer, the search space size becomes about up to $2^{99} \approx 6.34 \times 10^{29}$ only for a single layer.

Note that other settings, configurations, and customized connections can also be added to the above subspaces.

C. Phase 1: Multi-Level Configuration Encoding

As the encoding method of candidate architectures significantly affects the search speed and downstream task performance [25], [28], [29], we need to encode the configurations of each candidate architecture into the latent space before the search phase to enhance the search efficiency, given the large and complex search space. Formally, an encoder E is a function that maps a set of candidate models $(\mathcal{S}, \mathcal{A}, \mathcal{C})$ to n-dimensional Euclidean space, i.e., $E : (\mathcal{S}, \mathcal{A}, \mathcal{C}) \rightarrow \mathbb{R}^n$.

1) Encoder Networks: Given the heterogeneous properties of the configuration subspaces, we design three encoder networks, each of which learns a representation for a particular subspace. Then, we aggregate the three learned representations into a latent space $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_M\}, \mathbf{z}_i \in \mathbb{R}^n$. The proposed search method and performance predictor will use this continuous latent space instead of the raw discrete one.

Encoder for Task-Specific Settings. To map the raw representations of $\mathcal{S}$ to the latent space $\mathbf{Z}_S$, we use the fully connected (FC) layer. Given that the raw representations are one-hot vectors, FC is powerful enough to learn the features as used in standard AE because it does not require specific dependency modeling, e.g., spatial or temporal dependency. Hence, $\mathbf{Z}_S = FC_S(\mathcal{S})$.

Encoder for Architecture Configurations. To encode architecture configurations $\mathcal{A}$ effectively, we need an encoder that can capture the local relationships of different hyperparameters between layers. Therefore, we use two-dimensional convolutional ($Conv$) layers with max-pooling (M_{pool}) followed by an FC layer to learn the input of 3D tensors that represent the architecture hyperparameters in candidate networks. The latent representation for $\mathcal{A}$ is denoted as $\mathbf{Z}_A$. Thus, $\mathbf{Z}_A = FC_A(M_{\text{pool}}(Conv(\mathcal{A})))$.

Encoder for Temporal Connectivity. Unlike current encoding techniques that learn to represent the connection between layers or operations, we argue that they are unsuitable for our search space because using only layer-level representation cannot distinguish the architectures with distinct temporal connectivity even though their performances (i.e., F_1 score) are significantly different. Hence, embedding connectivity information in the temporal dimension into the latent space is inevitable for the subsequent search.

As depicted in Fig. 5, we show that for identical task-specific and architecture configurations with different temporal connectivity having significantly different performance (e.g., F_1 score) like the models in the preliminary experiments, we cannot use only operation-level or layer-level connections to distinguish them. Hence, we need to encode the temporal connectivity so that the search method can correctly find a well-performing model. We also verify this argument by running an ablation study under this scenario in §IV-E.

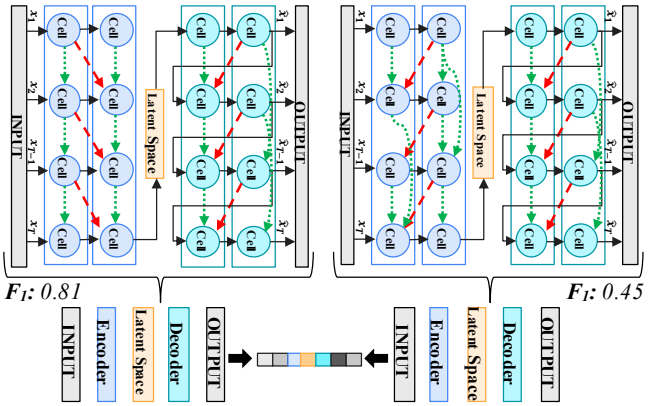


Fig. 5. Importance of temporal connectivity encoding. The same architectures with different temporal connectivity would give an identical embedding when the connections in the temporal dimension were disregarded.

Since the temporal connectivity forms the DAG, the natural choice for representing the connection in each encoder/decoder of an AE could be an adjacency matrix. It is possible for most existing NAS because the number of nodes for the entire network is merely about 10 to 20. However, the adjacency matrix approach faces the scalability problem as the growth rate is K^2. Moreover, computation-aware methods [29] have been shown to be superior to the adjacency matrix-based structure-aware methods. As a remedy, we propose to use a *connection tensor* instead of the hefty adjacency matrix to represent the temporal connectivity of candidate architectures, reducing the memory usage while having the property of a computation-aware method. The connection tensor consists of $K \times L$ c-tuple vectors, where L is the number of layers, c is the number of possible connections in a cell, and $L \times c \ll K$. The information in each vector is a list of negative numbers representing how far the current cell is connected to its previous cells, both within the same layer and between different layers. Thus, the smaller numbers indicate that the cell computes the longer-term dependency modeling, while the larger numbers indicate the shorter ones. This computation information cannot be represented by the adjacency matrix approach.

Fig. 6 illustrates an example of the proposed connection tensor with $L = 2$, $K = 4$, and $c = 4$, where L_i and C_t denote the i-th layer and the t-th recurrent cell, respectively. Here, despite the substantial space consumption, 64, the adjacency matrix only represents the topological connection of the entire structure without containing the type of connectivity in each

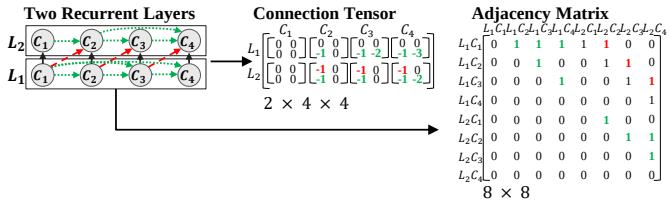


Fig. 6. An example of an adjacency matrix and a connection tensor for temporal connectivity representation.

value or local relationship between cells. In contrast, the connection tensor meaningfully describes each cell's local connectivity and underlying computation using position-based negative values with a smaller space consumption, 32. Specifically, the upper part (colored in red) indicates between-layer connectivity in the example, while the lower part (colored in green) indicates within-layer connectivity. Moreover, the scalability issue of the adjacency matrix will be more intensified in practice because K is usually large for the model to capture a long and diverse temporal dependency. When K is increased to 100, the adjacency matrix size becomes 200×200 for the encoder part and 200×200 for the decoder part, while the connection tensor uses only $100 \times 2 \times 4$ to represent each part, which is 50x smaller than the adjacency matrix representation.

Then, considering the connection tensor resembles the characteristics of multivariate time series having length K with $L \times c$ variables, we use the $Transformer$ network [107] to embed the temporal connectivity by capturing the dependency of significant computation information between cells and highlighting them via the attention mechanism. The global average pooling (G_{pool}) further summarizes the learned representations outputting $\mathbf{Z}_C$. Other networks, e.g., RNN or CNN, can also be used if they are capable of learning such dependency. Hence, $\mathbf{Z}_C = G_{\text{pool}}(Transformer(\mathcal{C}))$.

Ultimately, $\mathbf{Z}_S$, $\mathbf{Z}_A$, and $\mathbf{Z}_C$ are concatenated and input to a final FC layer to learn the final aggregated representation $\mathbf{Z}$. Formally, the final output of the encoder network E is $\mathbf{Z} = FC(Concat([\mathbf{Z}_S, \mathbf{Z}_A, \mathbf{Z}_C]))$.

2) Decoder Network: The decoder is a generative model aiming at reconstructing $\hat{\mathcal{S}}$, $\hat{\mathcal{A}}$, and $\hat{\mathcal{C}}$ from the latent variables $\mathbf{Z}$. Specifically, its constituent is a stack of FC layers. The final activation functions for $\hat{\mathcal{S}}$ and $\hat{\mathcal{A}}$ are the row-wise softmax σ to reconstruct the one-hot choices, while for $\hat{\mathcal{C}}$ is the $ReLU$ with the negative slope coefficient $= 1$ to reconstruct the negative values of the connection tensor. Hence, the decoder is formulated as $\hat{\mathcal{S}} = \sigma(FC_{\hat{S}}(\mathbf{Z}))$, $\hat{\mathcal{A}} = \sigma(FC_{\hat{A}}(\mathbf{Z}))$, and $\hat{\mathcal{C}} = ReLU(FC_{\hat{C}}(\mathbf{Z}))$.

3) Unsupervised Pre-Training: As in Fig. 2, we train the multi-level configuration encoding model on M configurations randomly sampled from the search space by minimizing the reconstruction loss $\mathcal{L} = \|\mathcal{S} - \hat{\mathcal{S}}\|_2 + \|\mathcal{A} - \hat{\mathcal{A}}\|_2 + \|\mathcal{C} - \hat{\mathcal{C}}\|_2$.

D. Phase 2: Performance Predictor-Guided Search

Given the vast and complex search space, we need a highly efficient search strategy. To achieve the highest efficiency possible, given a limited sample budget, this paper combines the idea of recent work [76], [77] to search on the proposed search space based on the learned latent space $\mathbf{Z}$ from the multi-level configuration encoding. With the performance predictor $\mathcal{P}$, we reformulate Eq. (1) as

$$(S^\star, A^\star, C^\star) = \underset{S \in \mathcal{S}, A \in \mathcal{A}, C \in \mathcal{C}}{\arg\max} \mathcal{P}(\underbrace{E(S, A, C)}_{\mathbf{z} \in \mathbf{Z}}). \quad (6)$$

It is worth noting that *PASTA* is a predictor-agnostic framework, so any prediction-based method can also be used. As illustrated in Fig. 2, after pre-training the multi-level configuration autoencoder network, we use the performance predictor $\mathcal{P}$ to guide the search procedure as follows.

1) Architecture-Performance Data Generation: To train the predictor $\mathcal{P}$, we first need a dataset of (architecture, performance) pairs. Unlike previous studies, we both uniformly and selectively sample (S, A, C) models from $(\mathcal{S}, \mathcal{A}, \mathcal{C})$. We *uniformly* select the models to preserve the overall structure of the search space. The *selectively* sampled subset is expected to contain the representative models for the full coverage of temporal connectivity subspace. Thus, the task-specific and architecture configurations of the representative models are fixed. For simplicity, we denote $\mathcal{N}_{\text{train}} = \{(S, A, C)_1, \ldots, (S, A, C)_{|\mathcal{N}_{\text{train}}|}\}$. Here, we first train each model in $\mathcal{N}_{\text{train}}$ on the given time-series windows $\mathcal{W}_{train}$. Then, to generate the dataset for $\mathcal{P}$, we validate the trained models on $\mathcal{W}_{valid}$. For this purpose, we need a few anomaly labels for $\mathcal{W}_{valid}$. However, unlabeled datasets can still be supported by model transfer, as discussed in §IV-D. Depending on the computation resource, the models in $\mathcal{N}_{\text{train}}$ can be trained using either a reduced [83] or full training scheme.

2) Search with Performance Predictor: After generating the training data, we use the pre-trained multi-level configuration encoder to obtain the latent space $\mathbf{z}$ of each model in $\mathcal{N}_{\text{train}}$. Thus, we redefine $\mathcal{N}_{\text{train}} = \{\mathbf{z}_1, \ldots, \mathbf{z}_{|\mathcal{N}_{\text{train}}|}\}$. Then, to predict the performance of any candidate configurations in the search space, we initially train $\mathcal{P}$ on $\mathcal{N}_{\text{train}}$ and iteratively update its parameters with subsampled sets of predicted top-k candidates until the budget is reached. We denote the subset of candidate configurations for evaluation during the search as $\mathcal{N}_{\text{eval}} = \{\tilde{\mathbf{z}}_1, \ldots, \tilde{\mathbf{z}}_{|\mathcal{N}_{\text{eval}}|}\}$. Since we cannot predict the entire search space, the $\mathcal{N}_{\text{eval}}$ is usually a large subset of the search space (e.g., $|\mathcal{N}_{\text{eval}}| = 10^4$). The model(s) in $\mathcal{N}_{\text{eval}}$ having the highest predicted performance by $\mathcal{P}$, i.e., $(S^\star, A^\star, C^\star)$, will be trained from scratch on $\mathcal{W}_{train}$. Finally, the trained model that has the best validation score on $\mathcal{W}_{valid}$ will be selected.

IV. Experiments

This section presents a series of empirical studies to evaluate the effectiveness of *PASTA* for TSAD. The source code is available at https://github.com/kaist-dmlab/PASTA.

A. Experimental Setup

1) Benchmark Datasets: To comprehensively evaluate the TSAD models, we use the following public synthetic and real-world *multivariate* time-series benchmarks. **TODS** [108] is a synthetic time-series generator based on behavior-driven taxonomy of anomalies categorized into point-global (**Global**), point-contextual (**Contextual**), pattern-shapelet (**Shapelet**), pattern-seasonal (**Seasonal**), and pattern-trend (**Trend**). Thus, we have 5 entities from this benchmark. **ASD** [8] is the

TABLE II
BENCHMARK DATASET STATISTICS.

Benchmarks	Applications	Window Size K	# Entities (datasets)	# Dims	Average per Entity # Train	# Test	% Anom.
TODS (Point)	Synthetic	100	2	5	20,000	5,000	4.92%
TODS (Pattern)	Synthetic	100	3	5	20,000	5,000	4.96%
ASD	Server Monitoring	100	12	19	8,528	4,320	4.16%
SWaT	Cyber-Physical System	30	1	51	495,000	449,930	14.59%
PSM	Server Monitoring	30	1	26	132,481	87,841	23.05%

recently introduced application server benchmark containing 12 entities with 19 metrics of the servers' status. **PSM** [12] is also a newly introduced pooled server metrics dataset containing 1 entity collected internally from multiple application server nodes at eBay. **SWaT** [109] is the most widely-used 1-entity dataset for TSAD collected from a real-world secure water treatment plant. The statistics of all benchmarks are presented in Table II. TODS is used to evaluate the detection performance on diverse difficulties and anomaly types, while ASD, PSM, and SWaT are used to evaluate the performance in real-world scenarios.

2) Evaluation Metrics: Many problems caused by evaluation metrics have been recently brought to attention [30], [47], [103], [110]. We thus adopt new evaluation metrics designed for TSAD to rigorously assess the performance. To tackle the overrating issue, we use the enhanced time-series aware precision-recall metrics [30] for calculating the best F_1 score. For a fair comparison, we use the default hyperparameters of the evaluation metric to evaluate all models. As it is more important to have an excellent F_1 score at a certain threshold δ in practice than to have a generally good performance on most thresholds [40], we report the highest F_1 score of a model computed by enumerating 1K thresholds uniformly distributed from the minimum to the maximum anomaly score over all time steps in the test set [10], [14]. This measurement helps eliminate the effect of threshold selection.

To increase the robustness in evaluating the model quality under the threshold-independent setting, we adopt another set of new metrics named Range Area Under the Curve (R_AUC) and Volume Under the Surface (VUS) [110], considering both the receiver operating characteristic (ROC) and precision-recall (PR) curves. The VUS extends the mathematical model of the R_AUC by allowing the buffer length to be varied. Hence, R_AUC_ROC and R_AUC_PR are defined for the former, and VUC_ROC and VUC_PR are defined for the latter.

3) Comparison Baselines: We compare the architectures discovered by *PASTA* with traditional methods, state-of-the-art handcrafted DL models, and search-based approaches. For traditional methods, we include Isolation Forest (IF) [111], Local Outlier Factor (LOF) [112], One-Class Support Vector Machine (OC-SVM) [113], Matrix Profile (MP) [114], and MERLIN [115]. For prediction-based DL models, Telemanom [6] and GDN [36] are included, while DAGMM [116], OmniAnomaly [7], RAE-SF [11], USAD [39], RANSynCoders [12], InterFusion [8], TranAD [42], and TimesNet [45] are included for reconstruction-based DL models. For search-based approaches, we include random search (RS) with *PASTA*'s search space and TODS-AutoML [93]. As mentioned in §II, we focus on finding the *reconstruction-based* neural architectures. We also include random anomaly score (RA) [103] to ensure that the results are meaningful. See supplementary material for baseline descriptions.

B. Implementation Details

1) Multi-Level Configuration Autoencoder:

- **Model Hyperparameters**. We set the hyperparameters of the multi-level configuration network as follows. For the ***encoder***, the number of hidden units for all FC layers is 16. The $Conv$ filter size is 16 with 2×2 kernel and 3×3 max pooling. We use the ReLU activation function for both FC and $Conv$. The $Transformer$ block has 8 attention heads with a hidden state size of 20 and hidden units for FC layers set to be the same size as the latent space $\mathbf{z}$. Here, the default latent space size is 32, i.e., $\mathbf{z} \in \mathbb{R}^{32}$. The dropout rate for $Transformer$ is 0.1. For the ***decoder***, the number of hidden units in all FC layers is also 16, except for the output layer.
- **Training Hyperparameters**. To train the multi-level configuration network, we first randomly sample 600K configurations from the search space, i.e., $M = 600$K. We use 90% of them for training and the remaining 10% are for testing, following Yan et al. [28]. The number of training epochs is 100. Also, we adopt early stopping with patience 10. The batch size is 32. The loss $\mathcal{L}$ is optimized using Adam optimizer with initial learning rate of 0.001.

2) Performance Predictor:

- **Architecture-Performance Generation**. As mentioned in §III-D2, we use both reduced-training and full-training schemes to construct (architecture, performance) pairs $\mathcal{N}_{\text{train}}$ with less computation time and resources. The first 30% of the testing data is used as validation set $\mathcal{W}_{\text{valid}}$. The performance scores of reduced-trained models are further recalibrated to mitigate potential noises by averaging the fully-trained performance scores of top-3 similar models based on their similarity scores computed by the cosine similarity function given their embeddings. In particular, we have fully-trained 256 models for each benchmark covering the possible options in the temporal connectivity subspace. For each benchmark, we get the following numbers of models under the fixed settings. **TODS**: 618, **ASD**: 874, **SWaT**: 662, and **PSM**: 393.
- **Prediction Model**. As an ensemble of predictors has shown to be efficient for performance prediction [117], we adopt NGBoost [118] as the default performance predictor $\mathcal{P}$, given its short training time with competitive accuracy. For all experiments, the *default* model and training hyperparameters of NGBoost are used.
- **Search Hyperparameters**. The search budget, i.e., number of total queries, is 200. We use *half* of search budget to initialize the performance predictor. Then, similarly to WeakNAS [77], we iteratively retrain the predictor using extra 20 models randomly sampled from top-100 predicted performance by $\mathcal{P}$ on the samples in $\mathcal{N}_{\text{eval}}$ until the limit is reached, where $|\mathcal{N}_{\text{eval}}| = 20$K. Last, models having top-5 predicted validation performance by the final predictor are selected to be trained and validated from scratch for final validation, following Wen et al. [76]. The model that achieves best validation performance is

TABLE III
PERFORMANCE COMPARISON BETWEEN TSAD METHODS IN TERMS OF THE BEST ENHANCED TIME-SERIES AWARE F_1 SCORE [30] WITH THE HIGHEST SCORES HIGHLIGHTED IN BOLD.

Methods	TODS (Point)		TODS (Pattern)			ASD	SWaT	PSM	Avg. ↑	Rank ↓
	Global	Contextual	Shapelet	Seasonal	Trend					
RA	0.087 (±0.010)	0.087 (±0.008)	0.093 (±0.009)	0.104 (±0.012)	0.105 (±0.021)	0.097 (±0.020)	0.257 (±0.007)	0.341 (±0.004)	0.146 (±0.011)	17
IF	0.693 (±0.078)	0.500 (±0.005)	0.455 (±0.070)	0.665 (±0.049)	0.154 (±0.005)	0.287 (±0.018)	0.221 (±0.031)	0.415 (±0.025)	0.424 (±0.035)	7
LOF	**1.000** (±0.000)	0.760 (±0.000)	0.795 (±0.000)	0.935 (±0.000)	0.237 (±0.000)	0.309 (±0.000)	0.180 (±0.000)	0.315 (±0.000)	0.567 (±0.000)	4
OC-SVM	0.994 (±0.000)	0.472 (±0.000)	0.609 (±0.000)	0.770 (±0.000)	0.253 (±0.000)	0.344 (±0.000)	0.108 (±0.000)	0.281 (±0.000)	0.479 (±0.000)	6
MP	0.039 (±0.000)	0.054 (±0.000)	0.097 (±0.000)	0.089 (±0.000)	0.229 (±0.000)	0.168 (±0.000)	0.235 (±0.000)	0.315 (±0.000)	0.153 (±0.000)	16
MERLIN	0.372 (±0.000)	0.227 (±0.000)	0.124 (±0.000)	0.197 (±0.000)	0.126 (±0.000)	0.163 (±0.000)	0.155 (±0.000)	0.242 (±0.000)	0.201 (±0.000)	15
Telemanom	0.022 (±0.003)	0.038 (±0.009)	0.127 (±0.002)	0.069 (±0.002)	0.113 (±0.008)	0.101 (±0.020)	0.146 (±0.039)	0.222 (±0.025)	0.105 (±0.013)	19
GDN	0.053 (±0.007)	0.076 (±0.008)	0.566 (±0.114)	0.766 (±0.109)	0.209 (±0.009)	0.253 (±0.029)	0.208 (±0.089)	0.232 (±0.019)	0.295 (±0.048)	9
DAGMM	0.348 (±0.243)	0.112 (±0.021)	0.089 (±0.012)	0.211 (±0.119)	0.122 (±0.027)	0.257 (±0.098)	0.180 (±0.049)	0.372 (±0.045)	0.211 (±0.077)	13
OmniAnomaly	0.316 (±0.037)	0.060 (±0.012)	0.142 (±0.024)	0.141 (±0.011)	0.203 (±0.028)	0.266 (±0.034)	0.446 (±0.156)	0.441 (±0.005)	0.252 (±0.038)	12
RAE-SF	0.556 (±0.000)	0.172 (±0.000)	0.084 (±0.000)	0.102 (±0.000)	0.201 (±0.000)	0.212 (±0.001)	0.126 (±0.005)	0.207 (±0.000)	0.208 (±0.001)	14
USAD	0.015 (±0.026)	0.055 (±0.033)	0.093 (±0.014)	0.128 (±0.070)	0.160 (±0.004)	0.118 (±0.082)	0.244 (±0.042)	0.264 (±0.001)	0.135 (±0.034)	18
RANSynCoders	0.549 (±0.152)	0.111 (±0.008)	0.187 (±0.022)	0.093 (±0.005)	0.099 (±0.011)	0.201 (±0.026)	0.402 (±0.053)	0.383 (±0.009)	0.253 (±0.036)	11
InterFusion	0.985 (±0.000)	0.730 (±0.001)	0.786 (±0.000)	0.930 (±0.000)	0.253 (±0.000)	0.329 (±0.006)	0.175 (±0.020)	0.591 (±0.001)	0.597 (±0.004)	2
TranAD	0.060 (±0.008)	0.066 (±0.006)	0.740 (±0.003)	0.839 (±0.001)	0.249 (±0.017)	0.432 (±0.024)	0.199 (±0.051)	0.330 (±0.024)	0.364 (±0.017)	8
TimesNet	0.958 (±0.004)	0.789 (±0.005)	0.812 (±0.006)	0.931 (±0.002)	**0.417** (±0.005)	0.277 (±0.030)	0.132 (±0.016)	0.435 (±0.007)	0.594 (±0.009)	3
RS	0.736 (±0.020)	0.576 (±0.007)	0.758 (±0.013)	0.664 (±0.011)	0.092 (±0.003)	0.455 (±0.076)	0.447 (±0.063)	0.401 (±0.024)	0.516 (±0.027)	5
TODS-AutoML	0.928 (±0.009)	0.328 (±0.000)	0.000 (±0.000)	0.463 (±0.005)	0.000 (±0.000)	0.008 (±0.000)	0.121 (±0.000)	0.335 (±0.000)	0.273 (±0.002)	10
PASTA	**1.000** (±0.002)	**0.790** (±0.003)	**0.815** (±0.003)	**0.936** (±0.001)	0.275 (±0.020)	**0.471** (±0.029)	**0.522** (±0.008)	**0.611** (±0.003)	**0.678** (±0.009)	1

selected. Notably, the search hyperparameters can be adjusted based on the availability of computation resources.

3) Anomaly Detection Models:

- **Model Hyperparameters**. All model hyperparameters of the state-of-the-art baselines are set as suggested in their source code or original papers. For **IF**, **LOF**, and **OC-SVM**, we search the model hyperparameters as suggested by Shen et al. [13]. For **MP**, we tune the subsequence length based on the default K value within $\{K/2,\ K/1.5,\ K,\ K \times 1.5,\ K \times 2\}$. For **MERLIN**, we use both recommeded hyperparameters [42] and search for minimum (minL) and maximum (maxL) values that correspond to the discord lengths when train on the new benchmarks. Here, minL $\in \{1, 5, 10, 20, 50, 80, 100\}$ and maxL $\in \{10, 20, 40, 60, 100, 140, 200\}$. For **TODS-AutoML**, we select the best detection results from different time limits $\in \{600, 7200, 10800, 21600\}$ seconds. We use grid search for these hyperparameter search spaces.
- **Training Hyperparameters**. To use less computation cost, we train the TSAD models using the following two sets of training hyperparameters. The first set of training hyperparameters is for ***full training***. Here, the batch size is 32, the number of epochs is 100, and early stopping is applied with patience 5. Regarding the second set, it is used for ***reduced training***. The batch size is also 32, while the number of epochs is only 5 without early stopping. For both cases, we use the Adam optimizer with an initial learning rate of 0.001. Note that all the baselines and final validation step of the selected candidate in *PASTA* use *full training* settings, while the *reduced training* settings are used for architecture-performance sample generation and validation of candidate architectures during search.

TABLE IV
ADDITIONAL PERFORMANCE COMPARISON FOR PASTA IN VUS [110] EVALUATION METRICS WITH THE HIGHEST SCORES HIGHLIGHTED IN BOLD. SEE [110] FOR THE DETAILS OF THESE EVALUATION METRICS.

Evaluation Metrics	Methods	TODS (Point)		TODS (Pattern)			ASD	SWaT	PSM	Average
		Global	Contextual	Shapelet	Seasonal	Trend				
R_AUC_ROC	InterFusion	0.9985	0.8591	0.8473	0.9319	0.5388	0.6482	0.3808	0.7712	0.7470
	TimesNet	0.9877	0.8599	0.8511	0.9287	**0.8174**	0.6203	0.2059	0.5707	0.7302
	PASTA	**0.9999**	**0.8672**	**0.8593**	**0.9431**	0.5810	**0.7689**	**0.8551**	**0.7745**	**0.8311**
R_AUC_PR	InterFusion	0.9820	0.5468	0.4936	0.8369	0.0535	0.1276	0.1170	0.5464	0.4630
	TimesNet	0.9526	0.6525	0.6551	0.8520	**0.2266**	0.1270	0.0957	0.3225	0.4855
	PASTA	**0.9987**	**0.6575**	**0.6616**	**0.8732**	0.0581	**0.2731**	**0.7629**	**0.6011**	**0.6108**
VUS_ROC	InterFusion	0.9781	0.8530	0.8457	0.9322	0.5505	0.6495	0.3812	0.7728	0.7454
	TimesNet	0.9806	0.8549	0.8528	0.9332	**0.8291**	0.6222	0.2067	0.5726	0.7315
	PASTA	**0.9844**	**0.8601**	**0.8721**	**0.9426**	0.5904	**0.7709**	**0.8552**	**0.7790**	**0.8318**
VUS_PR	InterFusion	0.9221	0.5269	0.4909	0.8307	0.0543	0.1280	0.1171	0.5489	0.4524
	TimesNet	0.9120	0.6224	0.6509	0.8477	**0.2383**	0.1274	0.0959	0.3238	0.4773
	PASTA	**0.9376**	**0.6230**	**0.6684**	**0.8663**	0.0590	**0.2735**	**0.7630**	**0.6028**	**0.5992**

TABLE V
COMPARISON OF ENCODING METHODS ON PERFORMANCE PREDICTOR.

Encoding Method	RMSE↓	Spearman's ρ ↑	Kendall's τ ↑
adjacency matrix	0.1000	0.6593	0.5261
arch2vec	0.0946	0.7157	0.5489
PASTA	**0.0764**	**0.7762**	**0.5846**

TABLE VI
COMPARISON OF ENCODING METHODS UNDER THE **IDENTICAL** CASES.

Encoding Method	RMSE↓	Spearman's ρ ↑	Kendall's τ ↑
adjacency matrix	0.0598	NaN	NaN
arch2vec	0.0598	NaN	NaN
PASTA	**0.0513**	**0.3770**	**0.2709**

C. Overall Performance Comparison

1) Time-Series Anomaly Detection: We report the main experimental results for TSAD averaged from three runs on different random seeds[3] across different datasets in the same benchmark. The averaged performance scores and standard deviations are reported in Table III. On average, the results clearly indicate that models discovered by *PASTA* significantly outperform both traditional approaches and state-of-the-art handcrafted detectors; specifically, it yields a detection accuracy of 13.6–545.7% higher than the other methods. InterFusion and TimesNet are ranked second and third, respectively.

As in recent studies [43], [119], we also compare *PASTA* to the well-performing models (InterFusion and TimesNet) using the additional metrics [110]. As presented in Table IV, the improvement in the range-AUC metrics (R_AUC_ROC or R_AUC_PR) is up to 31.9%, while the enhancement in the VUS-based metrics (VUS_ROC or VUS_PR) is up to 32.4%.

Overall, these results indeed demonstrate that *PASTA* can adaptively find a proper architecture given the different characteristics of time-series benchmarks. We conjecture that the improvement of the detection performance is from the integration of temporal connectivity in the proposed search space because it enables the search process to learn how to select the proper connections for temporal dependency modeling. Interestingly, even models found by random search (RS) also perform better than several state-of-the-art handcrafted models.

2) Performance Prediction: This experiment examines the usefulness of the multi-level configuration encoding on the performance predictor by measuring *how well the (learned) representations can predict the performance of the given candidate models* because good representations can facilitate the learning of the predictor, thereby enhancing its performance. We use 80% of the generated (architecture, performance) pairs dataset for training, and the results averaged across different benchmarks are reported based on the remaining test data. Following White et al. [81], we use RMSE, Spearman's ρ, and Kendall's τ rank correlation to evaluate the performance predictor in overall prediction accuracy and ranking capability.

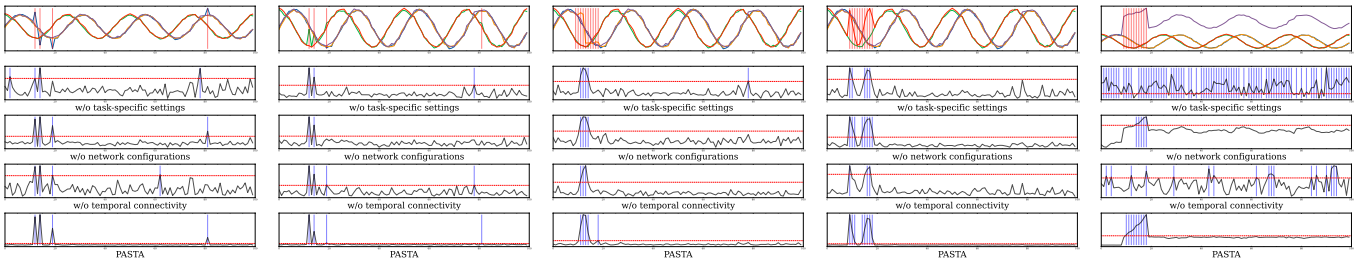
Here, we compare the multi-level configuration encoding, *PASTA*, with widely-used layer-level adjacency matrix [25] and state-of-the-art unsupervised architecture encoding arch2vec [28]. Given the identical predictor NGBoost [118] (i.e., $\mathcal{P}$ in Eq. (6)) and experimental settings, Table V shows that our multi-level configuration encoding of *PASTA* both reduces the performance prediction error and increases the ranking capability. Expectedly, learned latent representations are better than raw adjacency matrix representation.

Besides, we also include the scenario discussed earlier in §III-C to confirm the *necessity of temporal connectivity encoding*. The result is reported in Table VI. In both cases, *PASTA*'s multi-level configuration encoding demonstrates its usefulness in enhancing the detection accuracy prediction, having the lowest RMSE and the highest ranking correlation scores. Notably, layer-level adjacency matrix and arch2vec cannot differentiate models with different temporal connectivity; thus, the exact same embeddings are produced, resulting in the *NaN* values when the correlation scores were computed.

[3]The random seed for each run is 2^r, where r is the running index starting from 0. The seed for the search phase is 7.

TABLE VII
ABLATION STUDY ON MULTI-LEVEL CONFIGURATION ENCODING COMPONENTS FOR PASTA WITH THE BEST SCORES HIGHLIGHTED IN BOLD.

Encoder Variations	Time-Series Anomaly Detection (F_1)									Performance Prediction		
	TODS (Point)		TODS (Pattern)			ASD	SWaT	PSM	Avg. ↑	RMSE ↓	Spearman's ρ ↑	Kendall's τ ↑
	Global	Contextual	Shapelet	Seasonal	Trend							
w/o Task-Specific Settings	0.658	0.550	0.745	0.608	0.097	0.324	0.429	0.557	0.496	0.090	0.595	0.422
w/o Network Configurations	0.994	0.692	0.739	0.914	0.236	0.374	0.413	0.390	0.594	0.078	0.770	0.578
w/o Temporal Connectivity	0.823	0.571	0.767	0.778	0.105	0.321	0.361	0.444	0.521	0.084	0.763	0.567
PASTA	**1.000**	**0.790**	**0.815**	**0.936**	**0.275**	**0.471**	**0.522**	**0.611**	**0.678**	**0.076**	**0.776**	**0.585**



(a) Point Global. (b) Point Contextual. (c) Pattern Shapelet. (d) Pattern Seasonal. (e) Pattern Trend.

Fig. 7. Original time-series subsequences (first row) with ground-truth anomaly regions highlighted in red and anomaly scores produced by models discovered via each encoding method for different anomaly types from the TODS benchmark. Predicted anomalies of each model are highlighted in blue, corresponding to its threshold marked by the red dashed line.

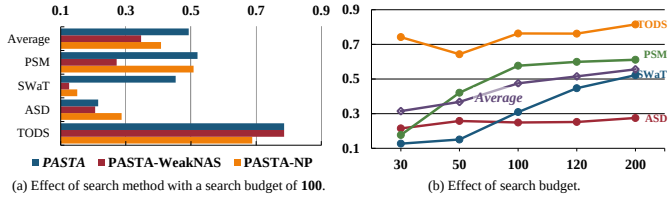
D. Transferability of Found Architectures

As the lack of labels is a key challenge in TSAD, we thus examine the transferability of the found models to test whether the proposed *PASTA* is practical. Specifically, we conduct an in-domain transfer by using *50% of labeled datasets* of the ASD benchmark (i.e., 6 out of 12 datasets) to generate the (architecture, performance) pairs and search for a TSAD model. Then, we use the found model for the *entire* benchmark and observe that the detection performance F_1 only drops by about 10% from 0.471 to 0.411, which is still comparable to and outperforms several baselines. This result shows a certain level of *PASTA*'s transferability for similar application domains, thereby mitigating the lack of labels to some extent.

E. Ablation Studies

To investigate and understand how each component in the proposed *multi-level configuration encoding* affects the downstream detection and performance prediction accuracy, we conduct the ablation experiments with three variants: w/o task-specific settings, w/o network configurations, and w/o temporal connectivity. Similar to the main experiments on performance prediction, we use 80% of the generated (architecture, performance) pairs dataset for training, and the results averaged across different benchmarks are reported based on the remaining test data.

Table VII shows the results of anomaly detection and performance prediction accuracy. On average, we observe that task-specific settings and temporal connectivity information are the most crucial to search the well-performing TSAD models. By varying *PASTA* encoder's component, we notice that the task-specific settings and temporal connectivity information are also the most crucial for the predictor to predict the detection performance of given TSAD models accurately, which is well-aligned with previous findings of deep TSAD designed by human experts. This result suggests that a suitable anomaly scoring function, an effective layer type, and connections



(a) Effect of search method with a search budget of **100**. (b) Effect of search budget.

Fig. 8. Comparison of F_1 score [30] given different search methods and budgets on the *first* dataset (due to the computational cost) of each benchmark.

in the temporal dimension of RNN-based AEs are strongly associated with anomaly detection performance.

In addition, Fig. 7 visualizes the anomaly scores produced by models found using different encoding variants for qualitative examination. As shown in the figure, the model found by *PASTA* gives the most distinguishable anomaly scores across all anomaly types. In contrast, the absence of either temporal connectivity or task-specific settings significantly affects the final downstream detection quality due to the high fluctuation of anomaly scores, leading to more false positives. These qualitative and quantitative results substantiate that *PASTA*'s multi-level configuration encoding is helpful in guiding the search process and effective for detecting various anomaly types, especially with the proposed search space.

F. Effects of Hyperparameters

To further examine how different hyperparameters affect the downstream detection and performance prediction accuracy, we conduct the following hyperparameter studies. Unlike the main ablation study on multi-level configuration encoding, we do not conduct the experiments for all datasets through the entire NAS process not only due to the unduly intensive computation cost, but it is also clear that the subsequent search process relies on the performance predictor, meaning that a well-performing predictor will lead to better downstream detection results (*small-scaled results are also reported below*).

TABLE VIII
PREDICTION ACCURACY BY DIFFERENT ARCHITECTURE OF E.

Encoder Architecture	RMSE↓	Spearman's ρ ↑	Kendall's τ ↑
Only FC	0.0839	0.6440	0.4658
FC + $Conv$	0.0771	0.7528	0.5600
FC + RNN	0.0870	0.6673	0.4790
FC + $LSTM$	0.0822	0.7135	0.5213
FC + GRU	0.0842	0.7083	0.5180
FC + $Transformer$	0.0813	0.7351	0.5412
PASTA-VAE	0.0900	0.6092	0.4315
PASTA	**0.0764**	**0.7762**	**0.5846**

TABLE IX
PREDICTOR ACCURACY BY VARYING THE LATENT SPACE SIZE OF E.

Latent Space Size	RMSE↓	Spearman's ρ ↑	Kendall's τ ↑
8	0.0896	0.6173	0.4427
16	0.0867	0.6704	0.4888
32 *(default)*	0.0764	**0.7765**	**0.5851**
64	**0.0755**	0.7346	0.5396
128	0.0890	0.6880	0.4949

1) Effect of Search Strategy: Since the search strategy can also affect downstream performance, we compare *PASTA*'s variants in terms of the search strategy. As in Fig. 8a, on average, it is evident that combining the advantages of Neural Predictor (NP) [76] and WeakNAS [77] achieves better results given the same search budget of 100 samples. Here, we speculate that NP provides high-quality results of performance prediction; at the same time, WeakNAS provides a diversity of candidate architectures.

2) Effect of Sample Budget: This experiment investigates how the search budget or the number of total queries affects the final detection performance. Although Fig. 8b shows that a higher search budget gives better average detection performance, a lower search budget can still produce good detection results for some datasets. This probably indicates the sample efficiency of *PASTA*.

3) Effect of Encoder Architecture: In this experiment, we show that the *heterogeneity of each subspace needs a proper encoder network*. From Table VIII, it is apparent that combining diverse types of layers is beneficial for modeling heterogeneous search space because using only fully-connected (FC) layers is not sufficient to capture the different properties of each subspace. Similarly, a simple combination with only recurrent (RNN, $LSTM$, or GRU), convolutional ($Conv$), or $Transformer$ block is not enough or even deteriorates the prediction accuracy. Moreover, we also test *PASTA* with variational loss (VAE) as used in arch2vec. Still, *PASTA* with simple reconstruction loss performs best.

4) Effect of Latent Space Size: In this experiment, we vary the size of latent space $\mathbf{Z}$ to examine how it affects the predictor accuracy. While there are slight differences between latent space sizes, as presented in Table IX, we can observe that 32 is the optimal size for our search space using multi-level configuration encoding. Thus, we use it as the default value to pre-train the multi-level configuration encoding for selected samples during the search phase.

5) Effect of Predictor Choice: According to a recent study [117], ensemble models are good at predicting the accuracy of architectures in search spaces. However, there are various choices in the research community. Here, we vary the different ensemble models to find the best one in terms of prediction accuracy and speed. Evidently, natural gradient boosting (NGB) is the most appropriate model, while extreme gradient boosting (XGB) and light gradient boosting (LGB) are too slow. The faster models are ensembles of multi-layer perceptron (MLP) based on the Bagging or AdaBoost method, yet their accuracy is not strong enough to guide the search. The results are shown in Table X.

TABLE X
PREDICTION ACCURACY BY DIFFERENT PREDICTORS.

Predictor	RMSE↓	Spearman's ρ ↑	Kendall's τ ↑	Training Time↓
Bagging-MLP	0.0916	0.6575	0.4705	**4.3489s**
AdaBoost-MLP	0.0908	0.6806	0.4918	8.2175s
LGB	0.0767	0.7747	0.5870	103.4623s
XGB	0.0774	**0.7788**	**0.5881**	40.7833s
NGB *(default)*	**0.0764**	0.7765	0.5851	9.6204s

TABLE XI
COMPUTATION COSTS USED FOR EACH BENCHMARK.

Benchmark	GPU Hours					
	Train Predictor (NGBoost)	Pretrain Encoder (One Time / K)	Search Time	Valid. Time	Train Time	Total
TODS	0.0016	14.52	0.16	24.81	18.05	57.55
ASD	0.0020	(K = 100)	0.15	10.90	48.14	73.71
PSM	0.0013	7.97	0.08	7.90	1.33	17.28
SWaT	0.0016	(K = 30)	0.07	4.76	12.64	25.44
Average	0.0016	11.25	0.11	12.09	20.04	43.50

TABLE XII
COMPARISON OF AVERAGE TRAINING TIME IN GPU SECONDS PER EPOCH BY VARYING ENCODING METHODS WITH THE LATENT SPACE SIZE OF 32.

Benchmark / Encoding Method	TODS / ASD (K = 100)	SWaT / PSM (K = 30)	Average
arch2vec	149.60	117.00	133.30
w/o Task-Specific Settings	294.00	518.60	406.30
w/o Network Configurations	272.20	479.20	375.70
w/o Temporal Connectivity	59.60	121.60	90.60
PASTA	294.40	518.80	406.60

G. Time Complexity

This subsection reports the computation costs used for the entire NAS process. Although we cannot quantify the exact amount of time used for architecture-performance data generation, approximately, we use a month to generate the number of models mentioned earlier. Note that the architecture-performance data generation and multi-level configuration encoder pre-training are the *one-time* cost.

Concerning the actual computation during the search process, Table XI shows the distribution of time in GPU hours used from the search phase to the training time of the final candidate model. The validation time is averaged from top-k models (i.e., 5 in our setting), while the training time is averaged from three runs. It is worth noting that the computation cost mostly depends on the dataset size and the complexity of the models selected during the search phase.

Table XII shows the training time between different encoder inputs, and Table XIII compares the training time depending on encoder architectures. Finally, Table XIV presents the comparison of training time between state-of-the-art handcrafted baseline models and *PASTA*. According to these computation costs, it is noticeable that *PASTA* relatively has high computational cost though with higher detection performance. Thus,

TABLE XIII
COMPARISON OF AVERAGE TRAINING TIME IN GPU SECONDS PER EPOCH BY VARYING ENCODER NETWORKS WITH THE LATENT SPACE SIZE OF 32.

Benchmark / Encoder Network	TODS / ASD (K = 100)	SWaT / PSM (K = 30)	Average
Only FC	83.60	136.20	109.90
FC + $Conv$	119.40	203.80	161.60
FC + RNN	2,175.60	1,618.20	1,896.90
FC + $LSTM$	673.80	696.20	685.00
FC + GRU	654.80	720.40	687.60
FC + $Transformer$	302.80	532.60	417.70
PASTA-VAE	295.20	517.40	406.30
PASTA	294.40	518.80	406.60

TABLE XIV
COMPARISON OF TRAINING TIME AVERAGED FROM THREE RUNS BETWEEN THE BASELINE MODELS AND MODELS FOUND BY PASTA IN GPU HOURS.

Model	TODS	ASD	SWaT	PSM	Average
Telemanom	6.18	3.43	1.10	1.27	2.99
GDN	18.84	28.74	6.21	30.28	21.02
DAGMM	0.51	0.09	0.06	0.24	0.23
OmniAnomaly	62.33	53.95	33.06	113.66	65.75
RAE-SF	32.17	29.04	5.87	8.66	18.93
USAD	1.38	1.41	0.66	1.39	1.21
RANSynCoders	5.92	5.08	5.45	10.87	6.83
InterFusion	9.54	0.93	0.52	0.68	2.92
TranAD	2.11	2.14	0.90	2.17	1.83
TimesNet	3.95	1.12	1.94	1.38	2.09
RS	8.91	40.55	0.48	0.44	12.59
PASTA	18.05	48.14	12.64	1.33	20.04

future work that jointly optimizes (like Pareto front) both performance and computational complexity during the search phase will be a very promising research direction.

V. CONCLUSION

This paper proposes *PASTA*, a prediction-based neural architecture search for multivariate time-series anomaly detection. *PASTA* consists of a well-tailored search space, multi-level configuration encoding with a novel temporal connectivity encoding between recurrent cells, and an efficient predictor-guided search. With rigorous experiments, we verify that the models discovered by *PASTA* show their superiority for detecting anomalies in multivariate time series by improving the enhanced time-series aware F_1 score by at least 13.6% on average with the help of multi-level configuration encoding. We also expect that our work will facilitate researchers and practitioners in discovering new deep TSAD models for the emerging time-series data using fewer computation resources and human efforts.

REFERENCES

[1] A. J. Fox, "Outliers in time series," *JRSSB*, vol. 34, no. 3, 1972.

[2] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE TKDE*, vol. 26, no. 9, pp. 2250–2267, 2013.

[3] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM CSUR*, vol. 54, no. 3, pp. 1–33, 2021.

[4] K. Choi, J. Yi, C. Park, and S. Yoon, "Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines," *IEEE Access*, vol. 9, pp. 120 043–120 065, 2021.

[5] Y. Luo, Y. Xiao, L. Cheng, G. Peng, and D. Yao, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," *ACM CSUR*, vol. 54, no. 5, pp. 1–36, 2021.

[6] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *KDD*, 2018, pp. 387–395.

[7] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *KDD*, 2019, pp. 2828–2837.

[8] Z. Li, Y. Zhao, J. Han, Y. Su, R. Jiao, X. Wen, and D. Pei, "Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding," in *KDD*, 2021, pp. 3220–3230.

[9] W. Chen, L. Tian, B. Chen, L. Dai, Z. Duan, and M. Zhou, "Deep variational graph convolutional recurrent network for multivariate time series anomaly detection," in *ICML*, 2022, pp. 3621–3633.

[10] Y. Nam, P. Trirat, T. Kim, Y. Lee, and J.-G. Lee, "Context-aware deep time-series decomposition for anomaly detection in businesses," in *ECML PKDD*, 2023, pp. 330–345.

[11] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Outlier detection for time series with recurrent autoencoder ensembles," in *IJCAI*, 2019, pp. 2725–2732.

[12] A. Abdulaal, Z. Liu, and T. Lancewicki, "Practical approach to asynchronous multivariate time series anomaly detection and localization," in *KDD*, 2021, pp. 2485–2494.

[13] L. Shen, Z. Li, and J. Kwok, "Timeseries anomaly detection using temporal hierarchical one-class network," in *NeurIPS*, 2020, pp. 13 016–13 026.

[14] L. Shen, Z. Yu, Q. Ma, and J. T. Kwok, "Time series anomaly detection with multiresolution ensemble decoding," in *AAAI*, 2021, pp. 9567–9575.

[15] L. Qingning, L. Wenzhong, Z. Chuanze, C. Yizhou, W. Yinke, Z. Zhijie, S. Linshan, and L. Sanglu, "Multi-scale anomaly detection for time series with attention-based recurrent autoencoders," in *ACML*, 2023, pp. 674–689.

[16] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *JMLR*, vol. 20, no. 1, pp. 1997–2017, 2019.

[17] H. Rakhshani, H. I. Fawaz, L. Idoumghar, G. Forestier, J. Lepagnot, J. Weber, M. Brévilliers, and P.-A. Muller, "Neural architecture search for time series classification," in *IJCNN*, 2020, pp. 1–8.

[18] Z. Xiao, X. Xu, H. Xing, R. Qu, F. Song, and B. Zhao, "RNTS: Robust neural temporal search for time series classification," in *IJCNN*, 2021, pp. 1–8.

[19] Y. Ren, L. Li, X. Yang, and J. Zhou, "AutoTransformer: Automatic transformer architecture design for time series classification," in *PAKDD*, 2022, pp. 143–155.

[20] X. Wu, D. Zhang, C. Guo, C. He, B. Yang, and C. S. Jensen, "AutoCTS: Automated correlated time series forecasting," *VLDB Endowment*, vol. 15, no. 4, pp. 971–983, 2021.

[21] D. Deng, F. Karl, F. Hutter, B. Bischl, and M. Lindauer, "Efficient automated deep learning for time series forecasting," in *ECML PKDD*, 2022, pp. 664–680.

[22] Z. Lai, D. Zhang, H. Li, C. S. Jensen, H. Lu, and Y. Zhao, "LightCTS: A lightweight framework for correlated time series forecasting," in *SIGMOD*, 2023.

[23] Y. Mehta, C. White, A. Zela, A. Krishnakumar, G. Zabergja, S. Moradian, M. Safari, K. Yu, and F. Hutter, "NAS-Bench-Suite: NAS evaluation is (now) surprisingly easy," in *ICLR*, 2022.

[24] A. Garg, W. Zhang, J. Samaran, R. Savitha, and C.-S. Foo, "An evaluation of anomaly detection and diagnosis in multivariate time series," *IEEE TNNLS*, vol. 33, no. 6, pp. 2508–2517, 2021.

[25] C. White, W. Neiswanger, S. Nolen, and Y. Savani, "A study on encodings for neural architecture search," in *NeurIPS*, 2020.

[26] C. White, M. Safari, R. Sukthanker, B. Ru, T. Elsken, A. Zela, D. Dey, and F. Hutter, "Neural architecture search: Insights from 1000 papers," *arXiv:2301.08727*, 2023.

[27] Y.-H. Yoo, U.-H. Kim, and J.-H. Kim, "Recurrent reconstructive network for sequential anomaly detection," *IEEE CYB*, vol. 51, no. 3, pp. 1704–1715, 2021.

[28] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang, "Does unsupervised architecture representation learning help neural architecture search?" in *NeurIPS*, 2020.

[29] S. Yan, K. Song, F. Liu, and M. Zhang, "CATE: computation-aware neural architecture encoding with transformers," in *ICML*, 2021, pp. 11 670–11 681.

[30] W.-S. Hwang, J.-H. Yun, J. Kim, and B. G. Min, "Do you know existing accuracy metrics overrate time-series anomaly detections?" in *ACM SIGAPP SAC*, 2022, pp. 403–412.

[31] M. Braei and S. Wagner, "Anomaly detection in univariate time-series: A survey on the state-of-the-art," *arXiv preprint arXiv:2004.00433*, 2020.

[32] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: a comprehensive evaluation," in *Proceedings of the VLDB Endowment*, vol. 15, no. 9, 2022, pp. 1779–1797.

[33] Z. Z. Darban, G. I. Webb, S. Pan, C. C. Aggarwal, and M. Salehi, "Deep learning for time series anomaly detection: A survey," *arXiv:2211.05244*, 2022.

[34] H. Si, C. Pei, H. Cui, J. Yang, Y. Sun, S. Zhang, J. Li, H. Zhang, J. Han, D. Pei *et al.*, "Timeseriesbench: An industrial-grade benchmark for time series anomaly detection models," *arXiv preprint arXiv:2402.10802*, 2024.

[35] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, and Q. Zhang, "Multivariate time-series anomaly detection via graph attention network," in *IEEE ICDM*, 2020, pp. 841–850.

[36] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *AAAI*, 2021, pp. 4027–4035.

[37] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "LSTM-based encoder-decoder for multi-sensor anomaly detection," *arXiv:1607.00148*, 2016.

[38] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *WWW*, 2018, pp. 187–196.

[39] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "USAD: Unsupervised anomaly detection on multivariate time series," in *KDD*, 2020, pp. 3395–3404.

[40] C. Feng and P. Tian, "Time series anomaly detection for cyber-physical systems via neural system identification and bayesian filtering," *arXiv:2106.07992*, 2021.

[41] J. Xu, H. Wu, J. Wang, and M. Long, "Anomaly Transformer: Time series anomaly detection with association discrepancy," in *ICLR*, 2022.

[42] S. Tuli, G. Casale, and N. R. Jennings, "TranAD: Deep transformer networks for anomaly detection in multivariate time series data," *PVLDB Endow.*, vol. 15, pp. 1201–1214, 2022.

[43] Y. Yang, C. Zhang, T. Zhou, Q. Wen, and L. Sun, "Dcdetector: Dual attention contrastive representation learning for time series anomaly detection," in *KDD*, 2023, p. 3033–3045.

[44] Y. Li, W. Chen, B. Chen, D. Wang, L. Tian, and M. Zhou, "Prototype-oriented unsupervised anomaly detection for multivariate time series," in *ICML*, 2023, pp. 19 407–19 424.

[45] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long, "TimesNet: Temporal 2d-variation modeling for general time series analysis," in *ICLR*, 2023, pp. 1–23.

[46] I. U. Haq and B. S. Lee, "TransNAS-TSAD: harnessing transformers for multi-objective neural architecture search in time series anomaly detection," *arXiv:2311.18061*, 2023.

[47] R. Ghorbani, M. J. Reinders, and D. M. Tax, "PATE: Proximity-aware time series anomaly evaluation," in *KDD*, 2024.

[48] E. Galván and P. Mooney, "Neuroevolution in deep neural networks: Current trends and future challenges," *IEEE TAI*, vol. 2, no. 6, pp. 476–493, 2021.

[49] M. Tenorio and W.-T. Lee, "Self organizing neural networks for the identification problem," in *NIPS*, 1988, pp. 57–64.

[50] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[51] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *ICLR*, 2017.

[52] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *ICML*, 2019, pp. 7105–7114.

[53] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *ICLR*, 2020.

[54] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *ICLR*, 2018.

[55] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *ICML*, 2018, pp. 4095–4104.

[56] D. Dimanov, E. Balaguer-Ballester, C. Singleton, and S. Rostami, "MONCAE: Multi-objective neuroevolution of convolutional autoencoders," *arXiv:2106.11914*, 2021.

[57] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *IEEE/CVF CVPR*, 2019, pp. 10 734–10 742.

[58] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for MobileNetV3," in *IEEE/CVF ICCV*, 2019, pp. 1314–1324.

[59] Y. Wang, Y. Yang, Y. Chen, J. Bai, C. Zhang, G. Su, X. Kou, Y. Tong, M. Yang, and L. Zhou, "TextNAS: A neural architecture search space tailored for text representation," in *AAAI*, 2020, pp. 9242–9249.

[60] T. Li, J. Zhang, K. Bao, Y. Liang, Y. Li, and Y. Zheng, "AutoST: Efficient neural architecture search for spatio-temporal prediction," in *KDD*, 2020, pp. 794–802.

[61] Z. Pan, S. Ke, X. Yang, Y. Liang, Y. Yu, J. Zhang, and Y. Zheng, "AutoSTG: Neural architecture search for predictions of spatio-temporal graph," in *TheWebConf*, 2021, pp. 1846–1855.

[62] R. Jie and J. Gao, "Differentiable neural architecture search for high-dimensional time series forecasting," *IEEE Access*, vol. 9, pp. 20 922–20 932, 2021.

[63] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, A. Filippov, and E. Burnaev, "Nas-bench-nlp: neural architecture search benchmark for natural language processing," *IEEE Access*, vol. 10, pp. 45 736–45 747, 2022.

[64] Y. Fan, F. Tian, Y. Xia, T. Qin, X.-Y. Li, and T.-Y. Liu, "Searching better architectures for neural machine translation," *IEEE/ACM TASLP*, vol. 28, pp. 1574–1585, 2020.

[65] J. Xu, X. Tan, R. Luo, K. Song, J. Li, T. Qin, and T.-Y. Liu, "NAS-BERT: Task-agnostic and adaptive-size bert compression with neural architecture search," in *KDD*, 2021, pp. 1933–1943.

[66] M. Ding, X. Lian, L. Yang, P. Wang, X. Jin, Z. Lu, and P. Luo, "HR-NAS: Searching efficient high-resolution neural architectures with lightweight transformers," in *IEEE/CVF CVPR*, 2021, pp. 2982–2992.

[67] D. So, W. Mańke, H. Liu, Z. Dai, N. Shazeer, and Q. V. Le, "Primer: Searching for efficient transformers for language modeling," in *NeurIPS*, 2021, pp. 6010–6022.

[68] Y. Li, Z. Chen, D. Zha, K. Zhou, H. Jin, H. Chen, and X. Hu, "Automated anomaly detection via curiosity-guided search and self-imitation learning," *IEEE TNNLS*, 2021.

[69] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019, pp. 4780–4789.

[70] X. Chen, Y. Sun, M. Zhang, and D. Peng, "Evolving deep convolutional variational autoencoders for image classification," *IEEE TEVC*, vol. 25, no. 5, pp. 815–829, 2020.

[71] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *NIPS*, 2018.

[72] H. Peng, H. Du, H. Yu, Q. Li, J. Liao, and J. Fu, "Cream of the crop: Distilling prioritized paths for one-shot neural architecture search," in *NeurIPS*, 2020.

[73] H. Zhou, M. Yang, J. Wang, and W. Pan, "BayesNAS: A bayesian approach for neural architecture search," in *ICML*, 2019, pp. 7603–7613.

[74] H. Shi, R. Pi, H. Xu, Z. Li, J. Kwok, and T. Zhang, "Bridging the gap between sample-based and one-shot neural architecture search with BONAS," in *NeurIPS*, 2020, pp. 1808–1819.

[75] C. White, W. Neiswanger, and Y. Savani, "BANANAS: Bayesian optimization with neural architectures for neural architecture search," in *AAAI*, 2021, pp. 10 293–10 301.

[76] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans, "Neural predictor for neural architecture search," in *ECCV*, 2020, pp. 660–676.

[77] J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, and L. Yuan, "Stronger NAS with weaker predictors," in *NeurIPS*, 2021, pp. 28 904–28 918.

[78] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang, "A generic graph-based neural architecture encoding scheme for predictor-based NAS," in *ECCV*, 2020, pp. 189–204.

[79] C. Wei, C. Niu, Y. Tang, and J. Liang, "NPENAS: Neural predictor guided evolution for neural architecture search," *arXiv:2003.12857*, 2020.

[80] L. Dudziak, T. C. P. Chau, M. S. Abdelfattah, R. Lee, H. Kim, and N. D. Lane, "BRP-NAS: prediction-based NAS using gcns," in *NeurIPS*, 2020.

[81] C. White, A. Zela, R. Ru, Y. Liu, and F. Hutter, "How powerful are performance predictors in neural architecture search?" in *NeurIPS*, 2021, pp. 28 454–28 469.

[82] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *IEEE/CVF CVPR*, 2018, pp. 8697–8710.
[83] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "EcoNAS: Finding proxies for economical neural architecture search," in *IEEE/CVF CVPR*, 2020, pp. 11 393–11 401.
[84] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *UAI*, 2020, pp. 367–377.
[85] X. Gong, S. Chang, Y. Jiang, and Z. Wang, "AutoGAN: Neural architecture search for generative adversarial networks," in *IEEE/CVF ICCV*, 2019, pp. 3224–3234.
[86] C. Gao, Y. Chen, S. Liu, Z. Tan, and S. Yan, "AdversarialNAS: Adversarial neural architecture search for gans," in *IEEE/CVF CVPR*, 2020, pp. 5680–5689.
[87] T. Mo and B. Liu, "Encoder-decoder neural architecture optimization for keyword spotting," *arXiv:2106.02738*, 2021.
[88] P. Esling and C. Agon, "Time-series data mining," *ACM CSUR*, vol. 45, no. 1, pp. 1–34, 2012.
[89] S. Y. Shah, D. Patel, L. Vu, X.-H. Dang, B. Chen, P. Kirchner, H. Samulowitz, D. Wood, G. Bramble, W. M. Gifford *et al.*, "AutoAI-TS: Autoai for time series forecasting," in *ACM SIGMOD*, 2021, pp. 2584–2596.
[90] A. D. Kini, S. S. Yadav, A. S. Thakur, A. B. Awari, Z. Lyu, and T. Desell, "Co-evolving recurrent neural networks and their hyperparameters with simplex hyperparameter optimization," in *GECCO*, 2023, p. 1639–1647.
[91] P. Arora, S. M. J. Jalali, S. Ahmadian, B. K. Panigrahi, P. N. Suganthan, and A. Khosravi, "Probabilistic wind power forecasting using optimized deep auto-regressive recurrent neural networks," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 3, pp. 2814–2825, 2023.
[92] B. B. Moser, F. Raue, J. Hees, and A. Dengel, "Dartsrenet: Exploring new rnn cells in renet architectures," in *Artificial Neural Networks and Machine Learning–ICANN 2020: 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15–18, 2020, Proceedings, Part I 29*. Springer, 2020, pp. 850–861.
[93] K.-H. Lai, D. Zha, G. Wang, J. Xu, Y. Zhao, D. Kumar, Y. Chen, P. Zumkhawaka, M. Wan, D. Martinez, and X. Hu, "TODS: An automated time series outlier detection system," in *AAAI*, 2021, pp. 16 060–16 062.
[94] M. Goswami, C. I. Challu, L. Callot, L. Minorics, and A. Kan, "Unsupervised model selection for time series anomaly detection," in *ICLR*, 2023.
[95] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE TNNLS*, 2021.
[96] E.-G. Talbi, "Automated design of deep neural networks: A survey and unified taxonomy," *ACM CSUR*, vol. 54, no. 2, pp. 1–37, 2021.
[97] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM CSUR*, vol. 54, no. 4, pp. 1–34, 2021.
[98] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca, "Neural architecture search using deep neural networks and monte carlo tree search," in *AAAI*, 2020, pp. 9983–9991.
[99] H. Cheng, T. Zhang, Y. Zhang, S. Li, F. Liang, F. Yan, M. Li, V. Chandra, H. Li, and Y. Chen, "NASGEM: neural architecture search via graph embedding method," in *AAAI*, 2021, pp. 7090–7098.
[100] K. Jing, J. Xu, and P. Li, "Graph masked autoencoder enhanced predictor for neural architecture search," in *IJCAI*, 2022, pp. 3114–3120.
[101] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-VAE: A variational autoencoder for directed acyclic graphs," in *NeurIPS*, 2019, pp. 1586–1598.
[102] C. Wei, Y. Tang, C. Niu, H. Hu, Y. Wang, and J. Liang, "Self-supervised representation learning for evolutionary neural architecture search," *IEEE CIM*, vol. 16, no. 3, pp. 33–49, 2021.
[103] S. Kim, K. Choi, H.-S. Choi, B. Lee, and S. Yoon, "Towards a rigorous evaluation of time-series anomaly detection," in *AAAI*, 2022, pp. 7194–7201.
[104] B. Horne and C. Giles, "An experimental comparison of recurrent neural networks," in *NIPS*, 1994, pp. 697–704.
[105] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[106] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv:1406.1078*, 2014.
[107] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.
[108] K.-H. Lai, D. Zha, J. Xu, Y. Zhao, G. Wang, and X. Hu, "Revisiting time series outlier detection: Definitions and benchmarks," in *NeurIPS*, 2021.
[109] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in *CRITIS*, 2016, pp. 88–99.
[110] J. Paparrizos, P. Boniol, T. Palpanas, R. S. Tsay, A. Elmore, and M. J. Franklin, "Volume under the surface: A new accuracy evaluation measure for time-series anomaly detection," *VLDB Endowment*, vol. 15, no. 11, pp. 2774–2787, 2022.
[111] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *IEEE ICDM*, 2008, pp. 413–422.
[112] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying density-based local outliers," in *ACM SIGMOD*, 2000, pp. 93–104.
[113] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," in *Advances in neural information processing systems*, 1999, pp. 582–588.
[114] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh, "Matrix Profile I: All pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets," in *IEEE ICDM*, 2016, pp. 1317–1322.
[115] T. Nakamura, M. Imamura, R. Mercer, and E. Keogh, "MERLIN: Parameter-free discovery of arbitrary length anomalies in massive time series archives," in *IEEE ICDM*, 2020, pp. 1190–1195.
[116] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *ICLR*, 2018.
[117] S. Yan, C. White, Y. Savani, and F. Hutter, "NAS-Bench-x11 and the power of learning curves," in *NeurIPS*, 2021, pp. 22 534–22 549.
[118] T. Duan, A. Anand, D. Y. Ding, K. K. Thai, S. Basu, A. Ng, and A. Schuler, "NGBoost: Natural gradient boosting for probabilistic prediction," in *ICML*, 2020, pp. 2690–2700.
[119] Y. Nam, S. Yoon, Y. Shin, M. Bae, H. Song, J.-G. Lee, and B. S. Lee, "Breaking the time-frequency granularity discrepancy in time-series anomaly detection," in *WWW*, 2024, p. 4204–4215.

Patara Trirat received the BS degree in computer science from Kasetsart University in 2016. In 2020, he received the MS degree in knowledge service engineering from KAIST. He is a PhD student of school of computing at KAIST. His research interests include data mining and analysis, automated machine learning, and applied artificial intelligence.

Jae-Gil Lee received the BS, MS, and PhD degrees in computer science from KAIST. He is a professor with KAIST and is leading the Data Mining Lab. Previously, he was a postdoctoral researcher with the IBM Almaden Research Center and a postdoc research associate with the University of Illinois Urbana-Champaign. His research interests include data-centric AI, deep learning-based big data analysis, mobility and stream data mining, and large-scale distributed deep learning.