TinyAD: Memory-Efficient Anomaly Detection for Time-Series Data in Industrial IoT

Yuting Sun, Tong Chen, *Member, IEEE*, Quoc Viet Hung Nguyen, and Hongzhi Yin, *Senior Member, IEEE*

***Abstract*—Monitoring and detecting abnormal events in cyber-physical systems is crucial to industrial production. With the prevalent deployment of the industrial Internet of Things (IIoTs), an enormous amount of time-series data is collected to facilitate machine learning models for anomaly detection, and it is of the utmost importance to directly deploy the trained models on the IIoT devices. However, it is most challenging to deploy complex deep learning models such as convolutional neural networks (CNNs) on these memory-constrained IIoT devices embedded with microcontrollers (MCUs). To alleviate the memory constraints of MCUs, we propose a novel framework named Tiny Anomaly Detection (TinyAD) to efficiently facilitate onboard inference of CNNs for real-time anomaly detection. First, we conduct a comprehensive analysis of depthwise separable CNNs and regular CNNs for anomaly detection and find that the depthwise separable convolution operation can reduce the model size by 50%–90% compared with the traditional CNNs. Then, to reduce the peak memory consumption of CNNs, we explore two complementary strategies, 1) in-place; and 2) patch-by-patch memory rescheduling, and integrate them into a unified framework. The in-place method decreases the peak memory of the depthwise convolution by sparing a temporary buffer to transfer the activation results, while the patch-by-patch method further reduces the peak memory of layer-wise execution by slicing the input data into corresponding receptive fields and executing in order. Furthermore, by adjusting the dimension of convolution filters, these strategies apply to both univariate time series and multidomain time series features. Extensive experiments on real-world industrial datasets show that our framework can reduce peak memory consumption by 2–5× with negligible computation overhead.**

***Index Terms*—Anomaly detection, convolutional neural networks (CNNs), inference optimization, Internet of Things (IoT), on-device deep learning, microcontrollers (MCUs), tiny machine learning.**

Manuscript received 2 September 2022; revised 12 December 2022 and 3 February 2023; accepted 5 March 2023. Date of publication 5 April 2023; date of current version 11 December 2023. This work was supported in part by the Australian Research Council under the streams of Future Fellowship under Grant FT210100624, in part by the Discovery Early Career Researcher Award under Grant DE200101465 and Grant DE230101033, in part by the Discovery Project under Grant DP190101985, and in part by UQ New Staff Research Start-up under Grant NS-2103. Paper no. TII-22-3766. *(Corresponding author: Hongzhi Yin.)*

Yuting Sun, Tong Chen, and Hongzhi Yin are with the School of Information Technology and Electrical Engineering, University of Queensland, Saint Lucia 4072, Australia (e-mail: yuting.sun@uqconnect.edu.au; tong.chen@uq.edu.au; h.yin1@uq.edu.au).

Quoc Viet Hung Nguyen is with the Institute for Integrated and Intelligent Systems, Griffith University, Southport 4222, Australia (e-mail: henry.nguyen@griffith.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2023.3254668.

Digital Object Identifier 10.1109/TII.2023.3254668

I. Introduction

THE industrial Internet of Things (IIoTs) is a paradigm integrated with sensor devices and communication technologies to enhance productivity and automation in industrial systems (e.g., smart manufacturing and agriculture). However, the IIoT infrastructures supported by cyber-physical systems, such as smart water management systems and power grids, are susceptible to cyberattacks and sensor failure. Such vulnerability of IIoT systems suggests the importance of timely detection of abnormal events. By deploying anomaly detection techniques, these systems can be alerted to early changes in normal status and potential loss of data reliability.

In the conventional IIoT network, the sensor data are normally transmitted through the IIoT network and fed into an expert cloud platform to leverage data-driven models for anomaly detection. However, the remote central computing unit can result in high latency due to communication overhead and resource scheduling [1]. Hence, a low-latency solution is to offload prediction models from the cloud servers to IIoT devices. Although the advancement in chip technology facilitates the capability of pushing machine intelligence to IIoT devices, deploying deep learning models to microcontroller (MCU)-based devices is still in its nascent stage. Unlike mobile devices and edge devices with rich compute and storage resources (even GPUs), most MCU-based IIoT devices (such as various industrial sensors) have internal flash memory $\leq$1 MB for data storage and SRAM $\leq$64 kB for in-memory processing [2], [3]. This memory constraint of MCU hinders the on-device deployment of the deep learning-based anomaly detection methods, such as DeepAnT [4] and LSTM-AD [5]. Thus, we aim to develop a solution for the state-of-the-art (SOTA) deep learning-based anomaly detection models to reach minimal memory consumption (especially peak memory) while preserving the prediction performance.

In the temporal context, the most common approaches in the literature for anomaly detection are prediction-based models, which can determine whether a new data point is an anomaly upon arrival [6]. With the advent of deep learning, convolutional neural network (CNN) and long short-term memory

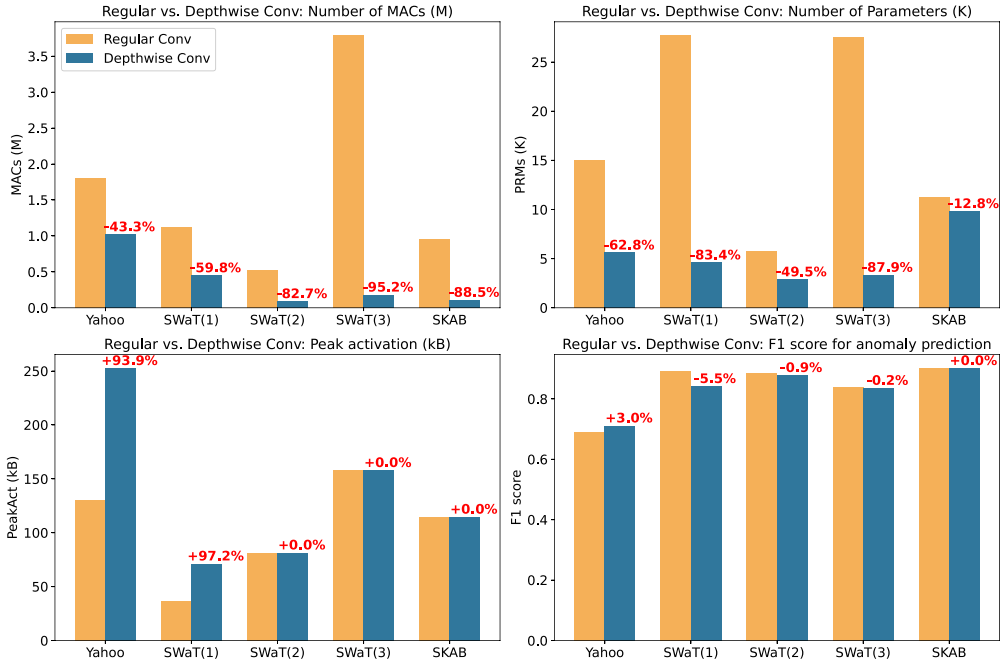


Fig. 1. Analysis on five industrial datasets: Comparison between regular convolution and depthwise separable convolution with respect to the number of MACs, number of parameters, peak activation size, and F1 score.

(LSTM)-based prediction models emerge as the most powerful solution to anomaly detection with the SOTA accuracy [7], [8]. Compared with LSTM, CNN shows its superiority in its model expressivity and training efficiency. Specifically, CNN-based temporal models can efficiently capture longer time scales by expanding larger receptive fields, but LSTM needs to propagate temporal information in a recurrent manner which leads to slower inference speed and larger memory cost. Moreover, CNN achieves remarkable performance in extracting time-independent and informative features from multidimensional data types by adjusting the dimension of convolution filters [9], [10]. In contrast, LSTM is limited to input dimensions and cannot sufficiently capture time-independent features from high-dimensional inputs. Also, the layer-by-layer structure and kernel-wise operation of CNN models pave the way for memory rescheduling, while the stacked recurrent layers in LSTM impede the rescheduling of the deterministic hidden states. Motivated by the prominent characteristics of CNN, we first investigate the performance of CNN-based models for time series anomaly detection. The widely used CNN models for anomaly detection consist of convolution, pooling layers, and fully connected layers [4], [8]. However, the regular convolution operation faces expensive computation and storage costs given its nature of cross-channel computation. Unlike regular convolution, the depthwise separable convolution performs spatial convolution independently for each input channel and creates linear output combinations by pointwise convolution. We conduct a comprehensive empirical comparison of CNN models with regular convolution and depthwise separable convolution, including prediction accuracy (measured by F1 score), computation complexity [measured by the number of multiply-accumulate operations (MACs)], model size (measured by the number of parameters), and peak memory size (measured by peak activation size). As shown in Fig. 1, the depthwise separable convolution can significantly reduce the number of MACs and model parameters by 50%–90%, compared with the regular convolution. Meanwhile, the F1 score on anomaly detection tasks indicates that the depthwise separable convolution achieves comparable prediction accuracy. To retain the same performance as the regular convolution, the depthwise convolution is likely to double the number of channels. However, depthwise convolution still involves much fewer parameters compared to regular convolution, which weakens the model's ability to learn strong representations from the data. Thus, we observe the slight performance drop (5.5%) of dataset SWaT(1) utilizing depthwise convolution. Although the depthwise separable convolution largely brings down the computation complexity and the model size with negligible accuracy loss, it still yields excessively high peak memory (i.e., peak activation size) for MCUs.

To reduce the peak memory of CNN models, MCUNet is proposed to reschedule the memory distribution among convolutional layers [11], [12], and it is designed for the image classification setting that requires many convolutional layers in the CNN models. When it comes to the signal data or time-series data, only a few (e.g., 1 or 2) convolutional layers will suffice, and hence, rescheduling the memory distribution does not work in this setting. Therefore, in this article, we propose a novel framework Tiny Anomaly Detection (TinyAD), to significantly decrease the peak memory of CNN models for real-time anomaly detection on MCU devices.

In this framework, we jointly optimize the peak memory consumption within and across the convolutional layers by innovatively exploring and integrating two memory rescheduling strategies: 1) in-place memory rescheduling within depthwise

convolutional layers; and 2) patch-by-patch execution order across convolutional layers. Given the layer-by-layer execution of CNN-based models, the peak memory consumption is normally dominated by the layer with the largest size of activations. To reduce the activation size of the depthwise convolutional layer, a temporary buffer is allocated in SRAM to transfer and update the activation results between channels, rather than holding all input and output activations in memory. However, the in-place memory rescheduling strategy cannot reschedule the computation memory across convolutional layers. To further reduce the peak memory (i.e., peak activation size), a patch-by-patch execution order across convolutional layers is proposed to slice the feature map into small spatial regions and execute a small patch at a time instead of computing the whole activation. Nevertheless, this sequential patch execution comes at the price of a slower inference speed. To accelerate on-device inference (i.e., prediction), we employ image-to-column (im2col) [13] to convert the kernel operation to matrix multiplication. We conduct extensive experiments on multiple real-world datasets, and the experimental results show that our proposed TinyAD significantly decreases the peak memory consumption by two to five times, regardless of various peak memory-dominant layers. Although the patch-by-patch execution confronts extra computation caused by overlapped receptive field, the computation overhead is trivial as CNN models just need a limited number of layers for time-series data. Also, as both strategies only manipulate the memory distribution during the inference process, our proposed TinyAD does not hurt the prediction accuracy of CNN models.

The major contributions of this article are as follows.

1) To the best of our knowledge, this is the first work to unlock the possibility of performing anomaly detection tasks using SOTA CNN models on MCU devices without model compression.
2) We perform a comprehensive analysis of depthwise separable CNNs and regular CNNs for anomaly detection and find that the depthwise separable convolution operation can significantly reduce the computation complexity (i.e., the number of MACs) and parameter size of CNN-based models by 50%–90%.
3) To largely reduce the peak memory cost of running CNN-based models, we propose a novel framework TinyAD, which integrates two complementary memory rescheduling strategies: In-place memory rescheduling within depthwise convolutional layers and patch-by-patch execution order across convolutional layers.
4) We conduct extensive experiments on multiple real-world datasets, and the experimental results show that our proposed TinyAD significantly decreases the peak memory of CNN-based models without compromising the prediction accuracy, paving the way for deploying CNN-based models on MCU devices.

II. Preliminaries

In this section, we begin with the essential introduction of the depthwise separable convolution and signal decomposition methods that are used in this work.

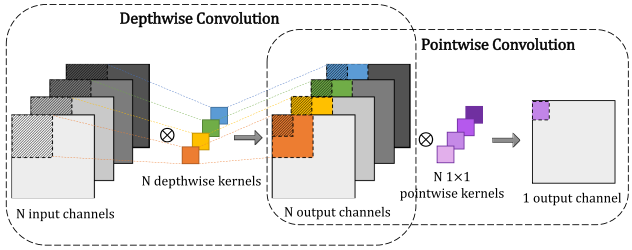


Fig. 2. Architecture of depthwise separable convolution.

A. Depthwise Separable Convolution

The depthwise separable convolution comprises a depthwise convolution and a pointwise convolution. The depthwise convolution performs channel-wise operations to model spatial relationships, while the pointwise convolution captures the cross-channel features by 1×1 kernels. The factorized form of depthwise separable convolution is shown in Fig. 2.

Formally, for a 2-D CNN (2D-CNN), we can represent a regular convolutional layer as a 4-D tensor $\mathbf{C} \in \mathbb{R}^{n_\mathrm{i} \times n_\mathrm{o} \times k_\mathrm{h} \times k_\mathrm{w}}$, where n_i and n_o denote the number of input and output channels, and the spatial height and width of kernels are represented as k_h and k_w, respectively [14]. Unlike regular 2-D convolution that captures intra- and interchannel correlations in one go, the depthwise separable convolution saves both computational and space complexity by dividing the traditional convolution into two steps. Specifically, the spatial feature-learning kernel of depthwise convolution is denoted as $\mathbf{D} \in \mathbb{R}^{n_\mathrm{i} \times K \times k_\mathrm{h} \times k_\mathrm{w}}$, where the positive integer K is the depthwise multiplier that determines the number of output channels after depthwise convolution as $K \times n_\mathrm{i}$. Fig. 2 demonstrates the standard depthwise convolution where $K = 1$ and $\mathbf{D} \in \mathbb{R}^{n_\mathrm{i} \times 1 \times k_\mathrm{h} \times k_\mathrm{w}}$, which indicates that the number of kernels is the same as the number of input channels. Also, the pointwise convolution kernel for channel combination is represented as $\mathbf{P} \in \mathbb{R}^{K n_\mathrm{i} \times n_\mathrm{o} \times 1 \times 1}$. For an input patch $\mathbf{x} \in \mathbb{R}^{n_\mathrm{i} \times k_\mathrm{h} \times k_\mathrm{w}}$, given the two 4-D tensors $\mathbf{D}$ and $\mathbf{P}$, the output vector $\mathbf{y}$ after the depthwise separable convolution is as follows:

$$\mathbf{y} = (\mathbf{P} \circ \mathbf{D}) * \mathbf{x} \tag{1}$$

where $\circ$ is the compound operation, $*$ is the convolution operation. Given a kernel for depthwise convolution as $\mathbf{D}_{i,k} = \mathbf{D}[i, k, :, :] \in \mathbb{R}^{k_\mathrm{h} \times k_\mathrm{w}}$, and a kernel for pointwise convolution as $\mathbf{P}_{j,o} = \mathbf{P}[j, o, :, :] \in \mathbb{R}^{1 \times 1}$. Each entry in $\mathbf{y}$ is obtained via $y_o = \sum_{k=1}^{K} \sum_{i=1}^{n_\mathrm{i}} \mathbf{P}_{n_\mathrm{i} \times (k-1)+i, o} * (\mathbf{D}_{i,k} * \mathbf{x}_i)$.

Considering a feature map of size $H \times W$, the computational complexity of regular convolution is $O(H \times W \times n_\mathrm{i} \times n_\mathrm{o} \times k_\mathrm{h} \times k_\mathrm{w})$, and the number of model parameters is given by $N = n_\mathrm{i} \times n_\mathrm{o} \times k_\mathrm{h} \times k_\mathrm{w}$. In contrast, depthwise separable convolution is more efficient in computation by achieving the computational complexity of $O(H \times W \times n_\mathrm{o} \times K \times (n_\mathrm{i} + k_\mathrm{h} \times k_\mathrm{w}))$. Furthermore, the number of model parameters is significantly reduced to $N = K \times n_\mathrm{i} \times (n_\mathrm{o} + k_\mathrm{h} \times k_\mathrm{w})$.

B. Signal Decomposition Methods

To acquire supplementary information from time-series data, feature extraction is one of the key procedures that can directly

TABLE I
HANDCRAFTED TIME DOMAIN FEATURE SETS

Quantity	Equations
Min	$\min = \min(\mathbf{z})$
Mean	$\mu = \frac{1}{n}\sum_{i=1}^{n} z_i$
Root mean square	$\text{rms} = \sqrt{\frac{1}{n}\sum_i z_i^2}$
Variance	$\text{var} = \frac{1}{n-1}\sum_{i=1}^{n}(z_i-\mu)^2$
Standard deviation	$\text{std}(\sigma) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(z_i-\mu)^2}$
Peak	$\text{p} = \max(\lvert\mathbf{z}\rvert)$
Peak-to-peak	$\text{p2p} = \max(\mathbf{z}) - \min(\mathbf{z})$
Crest factor	$\text{cf} = \frac{\text{p}}{\text{rms}}$
Skewness	$\text{skew} = \frac{\frac{1}{n}\sum_{i=1}^{n}(z_i-\mu)^3}{\sigma^3}$
Kurtosis	$\text{kurt} = \frac{\frac{1}{n}\sum_{i=1}^{n}(z_i-\mu)^4}{\sigma^4}$
Form factor	$\text{ff} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}}{\mu}$
Pulse indicator	$\text{pi} = \frac{\text{p}}{\mu}$

TABLE II
HANDCRAFTED FREQUENCY DOMAIN FEATURE SETS

Quantity	Equations
Spectral power	$\text{sp} = \sum_{i=1}^{k}(f_i)^3 S(f_i)$
Mean power frequency	$\text{mpf} = \frac{1}{k}\sum_{i=1}^{k}\frac{f_i S(f_i)}{\sum_{i=1}^{k} S(f_i)}$
Spectral skewness	$\text{sskew} = \sum_{i=1}^{k}\left(\frac{f_i-\bar{f}}{\sigma}\right)^3 S(f_i)$
Spectral kurtosis	$\text{skurt} = \sum_{i=1}^{k}\left(\frac{f_i-\bar{f}}{\sigma}\right)^4 S(f_i)$

affect anomaly detection accuracy. Therefore, we introduce the methods used for extracting time, frequency, and time-frequency domain features from raw time series input [15].

1) Time Features: The raw time-series data are intrinsically represented in the time domain. As temporal feature extraction methods aim to capture the morphology of time series, statistical methods are directly applied to capture the traits in the time domain. Considering a signal $\mathbf{z} = \{z_1, z_2, z_3, \ldots, z_n\}$, we explored 12 time domain features as described in Table I.

2) Frequency Features: Fourier transform is a widely used method to convert the time-domain representation of time series to the power spectral density, which describes the relative magnitudes of a time signal against its frequency components. Given a frequency f_i, the signal power is represented as $S(f_i)$. We considered four frequency domain features which are described in Table II.

3) Time-Frequency Features: To analyze how the frequency components of a time series change over time, we conduct discrete wavelet transform (DWT) to decompose the time series into a mutually orthogonal set of wavelets [15]. By scaling and translating the mother wavelet $\psi(t)$ (e.g., Daubechies), the DWT

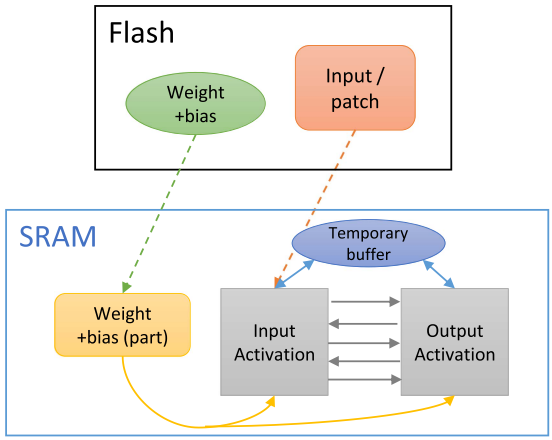


Fig. 3. Memory allocation of CNN-based anomaly detection models on MCU.

is expressed as follows:

$$\psi_{j,k}(t) = 2^{-j/2}\psi\left(2^{-j}t - k\right) \tag{2}$$

where $k \in [1, 2^{-j}N]$ is a location index and N denotes the number of observations. $j \in [0, J]$, and J is the number of scales. To quantify the time-frequency features, we obtain the coefficients of DWT at different levels as $\mathbf{w}_\phi(i)$, and the wavelet energy is given by $\sum_{i=1}^{N}\mathbf{w}_\phi^2(i)/N$. In this work, we explore the wavelet energy ranging from level 1 to level 3 generated by mother wavelet Daubechies 1 (db1) and Daubechies 2 (db2).

III. METHODOLOGY

In this article, we propose TinyAD, a framework that enables depthwise separable CNN-based anomaly detection model on MCU embedded edge devices. CNN is a typical sequential network architecture as the activation results are propagated layer by layer. Also, each receptive field can be identified and disjoined from the whole feature maps as a patch thanks to the independence of kernel operations. These specialties of CNN allow the flexibility of memory rescheduling during the forward pass. When deploying machine learning techniques on MCU devices, all model parameters and input data are stored in the Flash memory. At the same time, SRAM takes over all computations (e.g., convolution), holds and relieves the parameters and activation results alternatively during the inference process.

The Flash and SRAM memory allocation for the CNN-based model is described in Fig. 3. In practice, the parameters (weights and biases) of each layer can be structured as a list of objects in a saved file (e.g.,. json file), allowing us to load and parse one layer at a time, rather than load all model parameters in SRAM all at once. The memory space which holds parameters of the current layer will be released after the layer-specific computation finishes, and refilled by newly loaded parameters of the next layer. In TinyAD, the input patch is first loaded to the space in SRAM, and then the output from the lower layer will then serve as the input for the subsequent upper layer. Thus, we only need to allocate two variables/slots to store the intermediate results exchangeably. The temporary buffer is designed

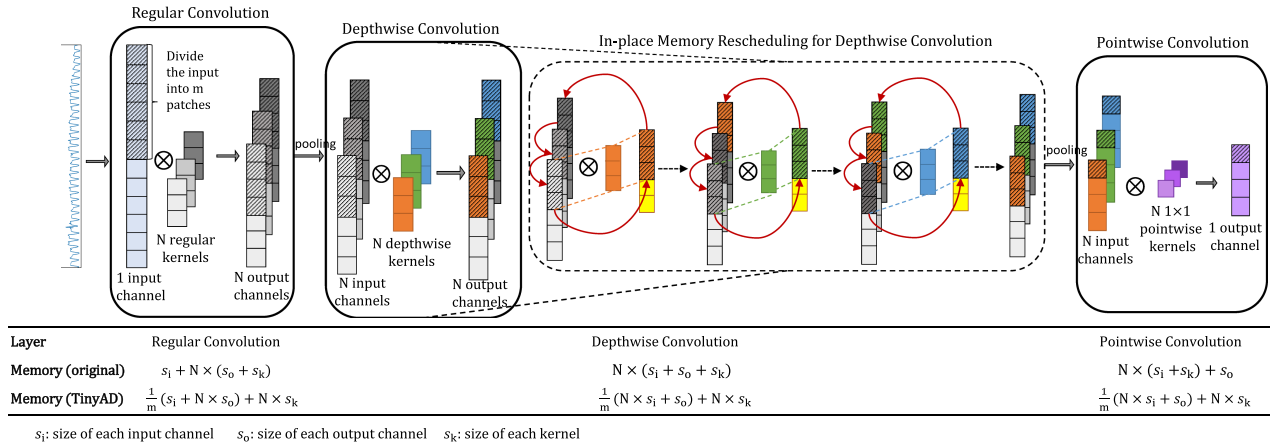


Fig. 4. Architecture of TinyAD: patch-by-patch and in-place memory rescheduling methods for time series anomaly detection. (1D-CNN).

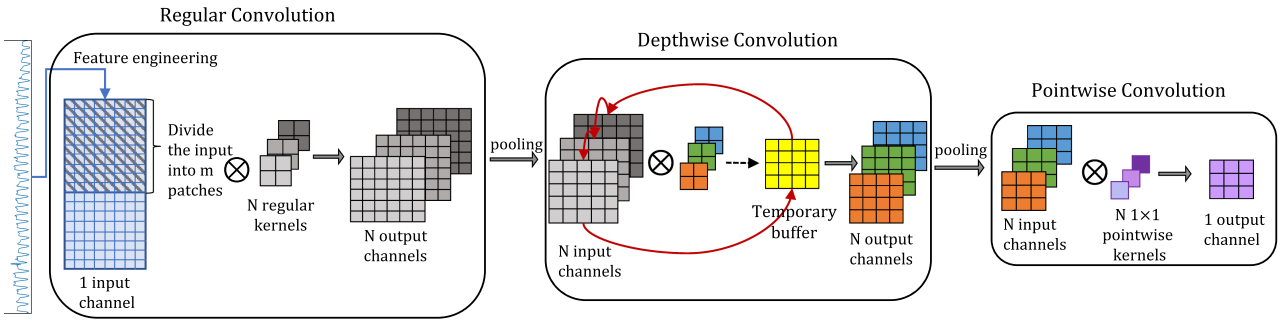


Fig. 5. TinyAD for 2D-CNN.

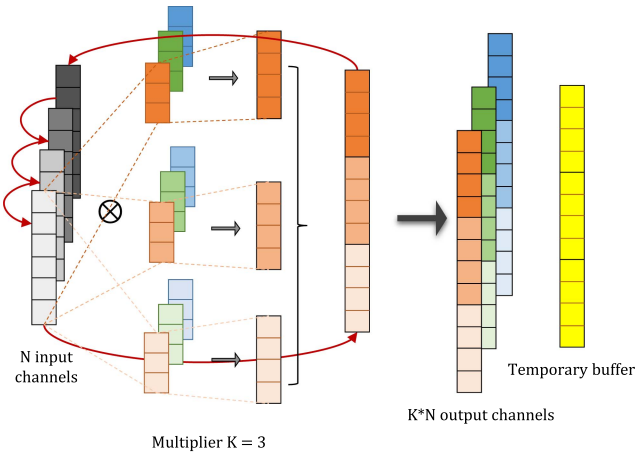
specifically for the depthwise convolution layer to temporarily hold output activations derived from each input channel, which is discussed in Section III-A. In this memory allocation scheme, the computation results of each layer are not transferred back to the flash until the inference process ends, which prevents the computation latency caused by the I/O stream.

Fig. 4 illustrates a 1D-CNN example with raw time series as input. Without memory rescheduling techniques, the memory consumption of each convolutional layer is determined by the sum of input and output activation sizes across all channels. As a result, the peak memory is determined by the dominating layer with the largest activation size in total. However, the memory consumption of each layer can be largely reduced after we deploy the approaches of memory rescheduling, among which the memory consumption of a convolution layer is reduced by at most 2 m times with the benefit of both patch-by-patch and in-place memory rescheduling methods, where m is the number of patches we divide the input time series into. For 2-D time series input with multidomain features, TinyAD is also applicable as we describe in Fig. 5.

In this section, we first discuss the in-place memory rescheduling method specifically for depthwise convolution layers, and the patch-by-patch method to control the execution order of feature maps. Then, we demonstrate the benefits of combining these two methods, which further alleviates the memory limit of anomaly detection models regardless of the memory-dominating layer.

A. Depthwise Convolution With in-Place Memory Rescheduling

As discussed in Section II-A, depthwise separable convolution replaces the feature learning process performed by regular convolutions with two phases: 1) An intra-channel spatial feature learning phase; and 2) a cross-channel feature combination phase. Considering the convolution operations are conducted on each channel independently, we can reschedule the memory footprint by assigning a temporary buffer to hold and transfer the activation results between channels. As described in Fig. 4, during the inference of depthwise convolution, rather than hold the whole input and output activation results, we overwrite the input activation of a channel with the output activation of another channel in the queue by utilizing the buffer as a "transit station," and shift the input activation across the channels without information loss. As shown in Fig. 6, given a depthwise convolution with multiplier K, the activation size of each input and output channel denoted as s_{i} and s_{o} respectively, and size of each kernel denoted as s_{k}, the size of temporary buffer is determined by $\max(s_{\mathrm{i}}, K \times s_{\mathrm{o}})$. This design ensures sufficient memory allocation when overwriting the input activation of a channel with the output activation of another channel in the queue. Thus, we can reduce the memory consumption of depthwise convolution from $N \times s_{\mathrm{i}} + NK \times s_{\mathrm{o}} + NK \times s_{\mathrm{k}}$ to $(N+1) \times \max(s_{\mathrm{i}}, K \times s_{\mathrm{o}}) + KN \times s_{\mathrm{k}}$. When $K = 1$, the



Layer	Depthwise Convolution (multiplier K = 3)
Memory (In-place)	$(\mathrm{N}+1) \times \max[s_{\mathrm{i}}, K \times s_{\mathrm{o}}] + KN \times s_{\mathrm{k}}$

s_{i}: size of each input channel s_{o}: size of each output channel s_{k}: size of each kernel

Fig. 6. Example of in-place memory rescheduling when depthwise multiplier $K = 3$.

method achieves its best effectiveness by reducing the memory consumption of depthwise convolution by around two times.

B. Patch-by-Patch Execution Rescheduling

Regular CNN without memory rescheduling performs convolution layer-by-layer, which requires a full load of input and output activation during the inference process. However, the patch-by-patch method cuts down memory usage by dividing the input feature map into small spatial regions, which are executed under the predesigned CNN architecture in order. Each output patch of the last conventional layer is held in SRAM until all the patches are passed through the convolutional layers. Given a predesigned CNN-based anomaly detection model, the receptive field (striped areas in Fig. 4) can be identified and matched layer-by-layer with respect to the final output beforehand. Although obtaining the nonoverlapping output patches comes at the price of overlapping receptive fields and repeated computation overhead, the time series anomaly detection models superbly avoid perceptible computation latency by its distinction of predicting anomalies with a limited number of convolutional layers. The adoption of patch-by-patch can effectively diminish the memory usage of each layer by m times, where m is the number of patches that the input time series has been divided.

C. Combining In-Place and Patch-by-Patch Schemes

As in-place and patch-by-patch methods reap the benefits of easing memory constraints, respectively, with intralayer and interlayer memory rescheduling, we combine these two strategies to maximize the potential of reducing the peak SRAM memory for CNN-based anomaly detection models. As described in Fig. 4, each patch is passed through the layers and initiates the in-place memory rescheduling when executing the depthwise convolution. Although patch-by-patch is applicable to various types of convolutional layers, the in-place memory rescheduling can further reduce the memory consumption of the depthwise convolutional layer, which makes the whole process of convolution more memory efficient. When depthwise convolution dominates the peak memory consumption in SRAM, TinyAD can reduce the memory usage from $N \times s_{\mathrm{i}} + NK \times s_{\mathrm{o}} + NK \times s_{\mathrm{k}}$ to $\frac{(N+1)\times\max(s_{\mathrm{i}}, K\times s_{\mathrm{o}})}{m} + KN \times s_{\mathrm{k}}$. Specifically, for a standard depthwise convolution layer where the depthwise multiplier $K = 1$, solely leveraging the patch-by-patch strategy can only reduce the memory by m times, and in-place memory rescheduling can only reduce the memory by at most two times. On the contrary, TinyAD can achieve its best performance by reducing the peak memory by 2 m times. This improvement further addresses the memory bottleneck in anomaly detection models on MCU.

IV. Experiment

A. Experiment Settings

1) Dataset: We conduct experiments on four industrial datasets and one commercial dataset. Each industrial dataset is a univariate sensor dataset, where the anomalies are abnormal sensor values caused by random or scheduled cyber attacks. The ratio of anomalies in SWaT and SKAB is 1% and 50%, respectively. We also tested our framework on a commercial dataset released by Yahoo Labs which contains around 0.2%–1% anomalies indicating outliers in computing systems. The decomposed signal features are extracted using the approaches introduced in Section II-B. A detailed description of each dataset is provided as follows.

- *Secure Water Treatment testbed (SWaT) [16]:* SWaT is a industrial water treatment plant, which records network data by second. We utilize records from three sensors named "FIT 401" (SWaT1), "LIT 101" (SWaT2), and "LIT 301" (SWaT3). These datasets are collected throughout five to six days with around 450 000 instances each.
- *Skoltech Anomaly Benchmark (SKAB) [17]:* SKAB is a miniature water circulation system, which contains 34 multivariate time series collected by sensors embedded on the testbed. We extract the univariate sensor signal "RateRMS" over 5 h, which has 18 161 instances to depict the circulation flow rate of the water.
- *Yahoo Webscope [18]:* In this work, we utilize the subdataset A1 benchmark as it consists of real-world time series, which depicts the aggregate status of Yahoo membership logins. Each time series contains around 1500 instances.

2) Parameter Settings: In our work, we split each dataset into 60% for training, 10% for validation, and 30% for testing. The hyperparameters of models are rigorously tuned to ensure the optimal performance under each dataset as shown in Table III. As we discussed in Section I, filters in depthwise convolution-based models are increased to remedy the accuracy loss, but lead to higher peak memory consumption.

Prediction-based anomaly detection model normally takes a historical window of time series to capture the temporal dynamics and tags the predicted result as a deviant or normal event.

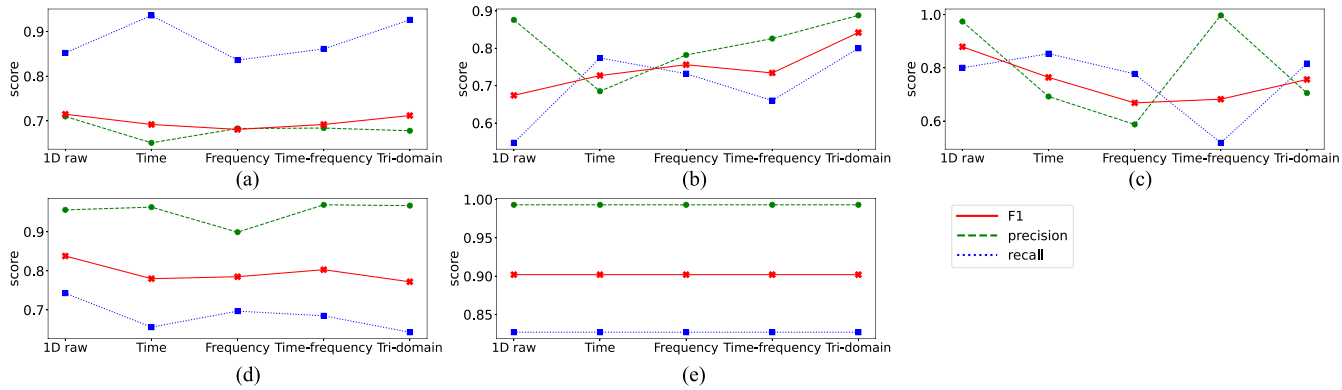


Fig. 7. Experiment on depthwise separable CNN: impact of different time series features.

TABLE III
HYPERPARAMETER SETTING IN THE EXPERIMENTS

Dataset	Kernel size	Number of filters	
		Regular conv	Depthwise conv
Yahoo	$(4, 4)$	32	32
SWaT(1)	$(5, 5)$	64	64
SWaT(2)	3	32	64
SWaT(3)	3	64	128
SKAB	3	16	32

To verify the efficiency of the memory rescheduling framework under the same input settings, we simply choose a consistent time window. Specifically, we take the records of the past 20 min for datasets SWaT and SKAB, where the input length is $L = 1200$. Given limited instances in the Yahoo dataset, we restrict the input length to $L = 200$.

B. Performance and Memory-Efficiency Comparison

In this section, we first discuss the impact of the multidomain time series features. Then, we illustrate the results of memory optimization by applying in-place, patch-by-patch, and the fused technique TinyAD accordingly. We conduct the experiment based on the same model structure as we mentioned in Fig. 4, followed by two fully connected layers. We use 1-D convolutional kernels for 1-D raw time series, and 2-D convolutional kernels for multidomain time series features.

1) Impact of Different Time Series Features: In this experiment, we compare the performance of anomaly detection by utilizing raw time series and multidomain features including time domain, frequency domain, time-frequency domain, and tridomain features which is a combination of the above three. As the result in Fig. 7 shows, different datasets can experience the advantages of different time series features. Particularly, 1-D raw input can fulfill the expectation of accurate anomaly detection on four datasets out of five by achieving higher F1 scores. Meanwhile, tridomain features outperform other types of input features in dataset Yahoo and SWaT(1) by achieving both higher scores of F1 and recall. However, dataset SKAB shows insensitivity towards feature engineering in anomaly detection tasks, given its striking difference between anomalies and normal records under scheduled cyber attacks. Thereby, We exploit tridomain features in the anomaly detection model for datasets Yahoo A1 and SWaT(1), and 1-D raw time series as input for the other datasets.

2) Memory Efficiency of In-Place Memory Rescheduling: Fig. 8 shows the memory efficiency of depthwise separable convolution under different memory rescheduling methods. In-place scheduling can operate effectively for depthwise convolution-dominant models, as it aims to reduce the memory consumption during depthwise convolution, but is not applicable to regular convolutional layers. In our experiment, with depthwise multiplier $K = 2$, in-place rescheduling can reduce the peak memory by 30%–36%, but fails to reduce the peak memory if it is dominated by regular convolutional layers.

3) Memory Efficiency of Patch-by-Patch Memory Rescheduling: Patch-by-patch significantly breaks the memory bottleneck by reducing the peak memory to at least half as compared to the nonrescheduled model, regardless of the type of memory-dominant layers. In our experiment, we divide the input time-series data into three patches and scale down the memory usage by two to three times solely with the patch-by-patch memory rescheduling method.

4) Memory Efficiency of TinyAD: Combining Patch-by-Patch and In-Place Memory Scheduling: With the joint design of patch-by-patch and in-place method, TinyAD unlocks the full potential of memory rescheduling techniques by reducing the peak memory usage two to five times as compared to the nonscheduled model. Based on the preliminary analysis of the memory distribution of layers, the memory bottleneck is normally given by the most memory-consuming convolutional layer, which has the largest sum of input and output activation sizes. On the one hand, if the memory bottleneck is determined by regular convolutional layers (e.g., on SWaT datasets), the overall memory reduction mainly relies on patch-by-patch memory rescheduling. Although the depthwise convolution layer still reaps benefits from the full model, the most memory-consuming layer (regular convolution) remains unaffected in this case. On the other hand, for the depthwise convolution-dominant models (e.g., on Yahoo and SKAB dataset), the peak memory is largely reduced when compared with both the patch-by-patch method and the in-place method.

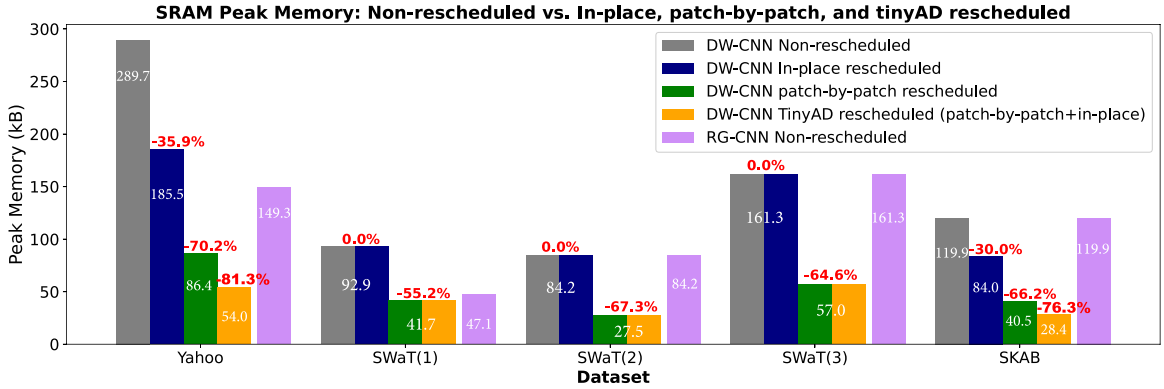


Fig. 8. Comparison of memory efficiency under different memory rescheduling methods (number of patches = 3).

TABLE IV
COMPARISON BETWEEN REGULAR CNN (RG-CNN), DEPTHWISE SEPARABLE CNN (DW-CNN) WITH AND WITHOUT TINYAD FOR MEMORY RESCHEDULING (NUMBER OF PATCHES = 3)

Dataset	Type	F1	MACs(M)	Model Size(kB)	PeakMem(kB)-float32	SRAM≤64kB
Yahoo (2-D CNN)	RG-CNN	0.691	1.80	60.2	149.3	✗
	DW-CNN	0.707	1.02 (↓ 43%)	22.4 (↓ ×3)	289.7	✗
	DW-CNN(TinyAD)	0.707	1.02	22.4	54.0 (↓ ×5)	✓
SWaT(1) (2-D CNN)	RG-CNN	0.891	1.12	110.8	47.1	✓
	DW-CNN	0.842	0.45 (↓ ×2.5)	18.4 (↓ ×5)	92.9	✗
	DW-CNN(TinyAD)	0.842	0.45	18.4	41.7 (↓ ×2)	✓
SWaT(2) (1-D CNN)	RG-CNN	0.887	0.52	22.8	84.6	✗
	DW-CNN	0.879	0.09 (↓ ×6)	11.5 (↓ ×2)	84.6	✗
	DW-CNN(TinyAD)	0.879	0.09	11.5	27.5 (↓ ×3)	✓
SWaT(3) (1-D CNN)	RG-CNN	0.838	3.79	110.08	161.3	✗
	DW-CNN	0.836	0.18 (↓ ×21)	13.3 (↓ ×9)	161.3	✗
	DW-CNN(TinyAD)	0.836	0.18	13.3	57.0 (↓ ×3)	✓
SKAB (1-D CNN)	RG-CNN	0.902	0.96	45.1	119.9	✗
	DW-CNN	0.902	0.11 (↓ ×9)	39.3 (↓ 12.8%)	119.9	✗
	DW-CNN(TinyAD)	0.902	0.11	39.3	28.4 (↓ ×4)	✓

Table IV exhibits the overall performance and memory efficiency of regular CNN and depthwise separable CNN models. It is evident that using depthwise separable CNN models assisted with TinyAD can reduce the model size by three to nine times, as well as peak memory usage by two to five times. After deploying TinyAD for depthwise separable models, all models become applicable on NodeMCU embedded devices (Flash memory $\leq$1 MB and SRAM $\leq$64 kB). Besides, there is no visible increase of MACs for depthwise separable models when executing patch-by-patch. This computation efficiency profits from the preevaluation of receptive fields with minimum overlaps and the limited convolutional layers of anomaly detection models.

The optimal choice of patch size varies under different datasets and different model parameter settings. As we can observe from Table IV, dataset Yahoo ($\text{patch\_size} = (60, 20)$) and SWaT(3) [$\text{patch\_size} = (421, 1)$] achieved optimal patch size under the SRAM budget of 64 kB. Although a smaller patch size can dramatically reduce peak memory usage, it also introduces more iterations of loading the parameters of each convolutional layer, as all patches' outputs need to be stitched together to obtain the full outputs. Thus, to balance the inference latency and memory budgets, the best patch size is the maximum patch size that can fit in the given SRAM memory budget of MCU.

C. Comparison of Inference Latency

Although TinyAD largely alleviates the memory bottleneck of deploying deep neural networks (DNNs) on MCUs, the inference time can be negatively impacted by the iterative flash-related data streaming and in-place rescheduling-related memory shift. Therefore, to estimate the time overhead during inference on MCU, we conduct experiments by simulating the workflow of DNN inference on MCU utilizing TinyAD, and compare the results with the on-CPU inference process without using TinyAD. For the MCU simulation, we explore the impact of data transmission between flash and SRAM under two settings: 1) single-thread; and 2) multithread. The single-thread process loads and performs layer-specific computation in a pipeline, while the multithread process allows loading the parameters of the next layer during the computation of the current layer.

Table V shows that the inference time is increased by the flash-related operational processes. It is worth mentioning that the on-CPU computation does not account for the potentially high communication cost from the data source (e.g., a sensor) to a computation node (e.g., a server) in an industrial scenario. As the result of patch-by-patch operation within TinyAD, the on-MCU simulation faces the iterative process of reading and

TABLE V
COMPARISON OF INFERENCE TIME BETWEEN ON-CPU AND ON-MCU SIMULATION (NUMBER OF PATCHES = 3)

Dataset	Method	Total inference (ms)	Data preparation (ms)	Forward (ms)
Yahoo	On-CPU (without TinyAD)	22.06	2.58	15.84
	MCU simulation (single-thread)	110.77	56.19	48.16
	MCU simulation (multi thread)	73.88 (↓33%)		
SWaT(1)	On-CPU (without TinyAD)	9.54	2.21	3.74
	MCU simulation (single-thread)	87.45	49.33	31.45
	MCU simulation (multi thread)	65.64 (↓25%)		
SWaT(2)	On-CPU (without TinyAD)	21.12	1.41	16.10
	MCU simulation (single-thread)	52.01	29.33	15.42
	MCU simulation (multi thread)	39.83 (↓25%)		
SWaT(3)	On-CPU (without TinyAD)	21.59	1.52	16.07
	MCU simulation (single-thread)	76.42	34.40	35.70
	MCU simulation (multi thread)	47.12 (↓38%)		
SKAB	On-CPU (without TinyAD)	23.76	4.13	16.03
	MCU simulation (single-thread)	116.68	97.64	12.14
	MCU simulation (multi thread)	97.64 (↓16%)		

Note: The percentage marked with "↓" indicates the efficiency gain from the multithread setting compared with the single-thread.

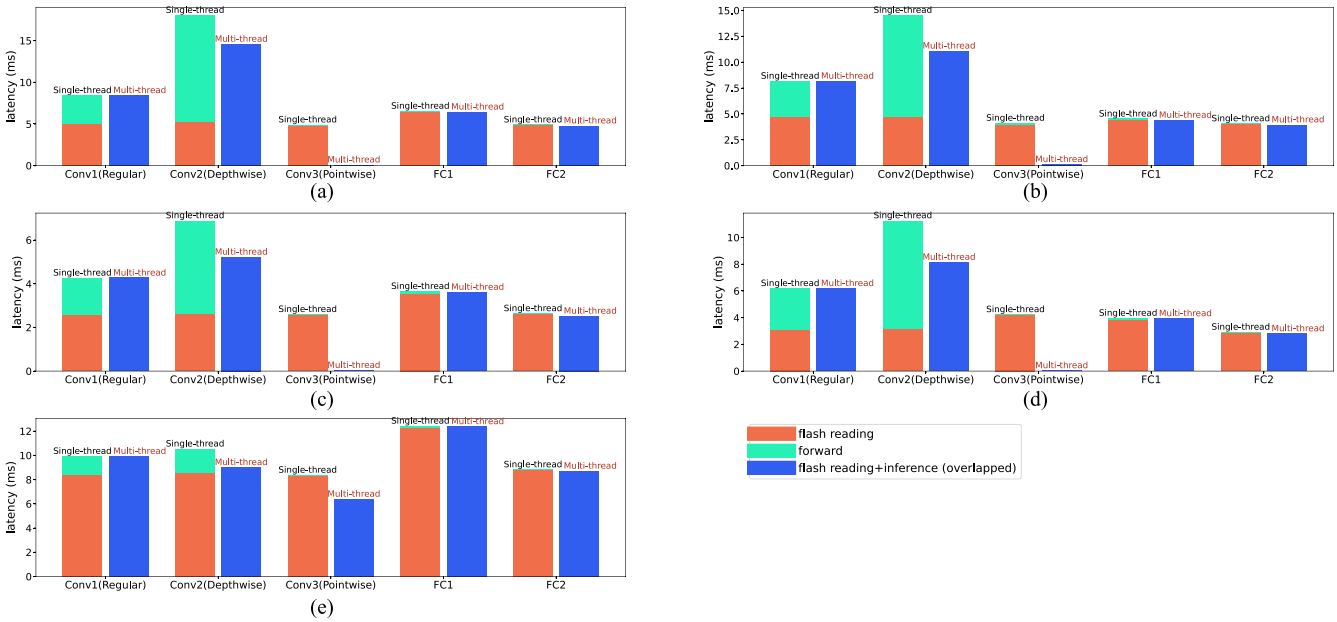


Fig. 9. Inference latency of each layer (number of patches = 3). The *y*-axis measures the latency of both data loading and forward computation (each patch) for all convolutional layers, and the latency of all fully connected layers.

decoding model parameters from the saved file onto the SRAM. Apart from this, rather than loading the whole file into SRAM at once, the flash-related operation latency mainly comes from retrieving the corresponding profile for each layer from flash before loading it to SRAM. Compared with the on-CPU full model, there is also a slight increase in the forward time caused by the in-place rescheduling approach. Notably, by utilizing multithread strategies for the on-MCU inference, the total inference time can be reduced by 16%–38%.

To better understand the benefits of introducing multithread processing during inference, we also explore the layer-wise inference latency, including the flash page read latency and other computations involved in the data loading process. The flash read latency t_{read} denotes the time the device needs to load a page of data for subsequent use after any read command is issued for a certain page address. Thus, the total read latency is $N_{\text{page}} \times t_{\text{read}}$, where N_{page} is the number of retrieved pages. In our experiments, $t_{\text{read}} = 25\mu$s, and the page storage size is 8 kB [19]. Fig. 9 shows the inference latency of each layer under both single-thread and multithread settings. As each layer only contains a small number of parameters that occupy only one to two units of flash page, the flash page read time is trivial compared to the overall data preparation process (3–12 ms), which also includes the document decoding and parameter retrieval process. Generally,

the multithread method can largely benefit both depthwise and pointwise convolutional layers since the flash-related operations can be processed simultaneously with the model forward computation. However, when the flash-related process dominates the execution duration, it becomes more challenging to reduce the inference time by utilizing multithread processing. Thus, experiments on the dataset SKAB show the smallest improvement, whereas SWaT(3) reaps the best results by completing most data reading tasks within the time of forward computation.

On average, the inference time of multithread TinyAD for on-MCU deployment is only three times slower than on-CPU deployment, showing a reasonable memory-speed tradeoff where low memory consumption is prioritized for MCU. Besides, the inference time for all datasets using TinyAD is under 0.1 s, which is considered tolerable for real-time anomaly detection tasks.

V. Related Work

A. Anomaly Detection for Time Series

There are a variety of machine learning techniques that can identify anomalous points from the normal time series in an unsupervised and supervised manner. Supervised learning utilizes generative and discriminative models to find differences between abnormal and normal instances. However, the scarcity and unbalance of labels hinder the utilization of supervised methods [20]. Therefore, unsupervised learning is proposed to reconstruct the time-series data from historical records, and capture the deviation of reconstruction and ground truth as anomaly scores. CNNs and recurrent neural networks (RNNs) are widely used as unsupervised anomaly detection techniques, in companion with autoencoder [4], [5], [21]. Unlike RNN-based models, the attention mechanism shows its robustness in sequence modeling by parallel processing the series data. In sequential recommendation systems, the attention mechanism is widely used to extract relevant information from users' historical behaviors and predict their future preferences [22], [23]. Motivated by the effectiveness of self-attention in modeling long-term trends, an attention-based anomaly detection model is also proposed for anomaly detection, by utilizing multimodal features extracted from broader temporal sequences [24].

1) Tiny Machine Learning: The deployment of machine learning on IoT devices faces the challenges of limited computation and memory resources. Existing methods employ quantization [25] and channel pruning [26] to reduce the model size and floating-point operations. However, these strategies cannot alleviate the memory bottleneck induced by intermediate activation size, which forces the utilization of smaller models and low-fidelity inputs. To address this problem, MCUNet is proposed to reschedule the memory distribution of convolutional layers and shift the receptive fields to eliminate the computation overhead [11], [12]. The experimental results of MCUNet reveal its significant improvement of memory efficiency on ResNet [27] for image classification tasks. In time series, ResNet is also one of the promising frameworks for time series prediction and anomaly detection tasks. Given its CNN-based structure, our proposed TinyAD is also applicable to the model but requires additional memory space to hold the input of residual blocks for identity mapping.

Additionally, low power and energy consumption are other obstacles to deploying machine learning on edge devices. Thus, the power-efficient neural network implementation is proposed to relieve the power budget of the sensor node. The authors in [28] jointly designed a hardware-oriented DNN model with accelerators to reduce power consumption and accelerate inference computation. Also, Eciton [29] is proposed for real-time inference of LSTM on low-power edge devices without accuracy loss.

1) On-device model update: Given the trend of deploying DNNs on edge devices, updating postdeployed DNN models on these memory-constrained devices is inevitable in future work. Given the costly communication and deficient label annotation in IoT systems, the most recent works explored the feasibility of updating the model on MCU devices by updating a compressed version of the model from server [30], and retraining the model on the device with streaming data [31]. However, the problem of updating anomaly detection models on MCU devices is left unaddressed, which is well worth the investigation.

VI. Conclusion

In this article, we proposed TinyAD to reduce the peak memory consumption of anomaly detection models on IIoT devices. The experimental results proved the prediction effectiveness and memory efficiency of depthwise separable convolution, and illustrated that TinyAD could further optimize the memory distribution among convolutional layers based on the specific characteristics of convolutions. Experimental results showed that our framework effectively reduced peak memory usage by two to seven times with competitive prediction performance and smaller model size. As the implementation largely alleviated the memory budget of IIoT devices, it unsealed the capability of real-time anomaly detection on low-cost sensor nodes.

Acknowledgment

The authors would like to thank Dr. Thomas Taimre and Dr. Radislav Vaisman (the University of Queensland) for their valuable discussions and helpful comments on this research.

References

[1] A. Diro, N. Chilamkurti, V.-D. Nguyen, and W. Heyne, "A comprehensive study of anomaly detection schemes in IoT networks using machine learning algorithms," *Sensors*, vol. 21, no. 24, Dec. 2021, Art. no. 8320.

[2] F. Jan, N. Min-Allah, S. Saeed, S. Z. Iqbal, and R. Ahmed, "IoT-based solutions to monitor water level, leakage, and motor control for smart water tanks," *Water*, vol. 14, no. 3, Jan. 2022, Art. no. 309.

[3] K. Namrata, A. Dayal, D. Tolia, K. Arun, and A. Ranjan, "IoT-integrated smart grid using PLC and NodeMCU," in *Control Applications in Modern Power System*. Berlin, Germany: Springer, 2021, pp. 241–250.

[4] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "DeepAnt: A deep learning approach for unsupervised anomaly detection in time series," *IEEE Access*, vol. 7, pp. 1991–2005, 2018.

[5] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. 23rd Eur. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn.*, 2015, Art. no. 89.

[6] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, "A review on outlier/anomaly detection in time series data," *ACM Comput. Surv.*, vol. 54, no. 3, Apr. 2022, Art. no. 56.

[7] Z. Zheng, Y. Yang, X. Niu, H.-N. Dai, and Y. Zhou, "Wide and deep convolutional neural networks for electricity-theft detection to secure smart grids," *IEEE Trans. Ind. Informat.*, vol. 14, no. 4, pp. 1606–1615, Apr. 2018.
[8] Y. Wang, X. Du, Z. Lu, Q. Duan, and J. Wu, "Improved LSTM-based time-series anomaly detection in rail transit operation environments," *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 9027–9036, Dec. 2022.
[9] Y. Wang, L. Hou, K. C. Paul, Y. Ban, C. Chen, and T. Zhao, "ARCNet: Series AC arc fault detection based on raw current and convolutional neural network," *IEEE Trans. Ind. Informat.*, vol. 18, no. 1, pp. 77–86, Jan. 2022.
[10] O. BeratSezer and A. MuratOzbayoglu, "Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach," *Appl. Soft Comput.*, vol. 70, pp. 525–538, Sep. 2018.
[11] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny deep learning on IoT devices," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 11711–11722.
[12] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," in *Proc. 35th Conf. Neural Inf. Process. Syst.*, 2021, pp. 2346–2358.
[13] J. Wang, Y. Chen, R. Chakraborty, and S. X. Yu, "Orthogonal convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11505–11515.
[14] J. Guo, Y. Li, W. Lin, Y. Chen, and J. Li, "Network decoupling: From regular to depthwise separable convolutions," in *Proc. 29th Brit. Mach. Vis. Conf.*, 2018, Art. no. 248.
[15] R. Zhao, D. Wang, R. Yan, K. Mao, F. Shen, and J. Wang, "Machine health monitoring using local feature-based gated recurrent unit networks," *IEEE Trans. Ind. Electron.*, vol. 65, no. 2, pp. 1539–1548, Feb. 2018.
[16] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in *Proc. Int. Conf. Crit. Inf. Infrastructures Secur.*, 2016, pp. 88–99.
[17] I. D. Katser and V. O. Kozitsin, "Skoltech anomaly benchmark (SKAB)," 2020. [Online]. Available: https://www.kaggle.com/dsv/1693952
[18] N. Laptev, S. Amizadeh, and Y. Billawala, "Yahoo anomaly detection dataset s5," 2015. [Online]. Available: https://webscope.sandbox.yahoo.com/catalog.php?datatype=sdid=70
[19] J. Heidecker, "Flash memory reliability: Read, program, and erase latency versus endurance cycling," Jet Propulsion Laboratory, National Aeronautics and Space Administration, Pasadena, CA, USA, Tech. Rep. JPL-Publ-10-19, 2011.
[20] S. Schmidl, P. Wenig, and T. Papenbrock, "Anomaly detection in time series: A comprehensive evaluation," in *Proc. VLDB Endowment*, 2022, vol. 15, pp. 1779–1797.
[21] C. Yin, S. Zhang, J. Wang, and N. N. Xiong, "Anomaly detection based on convolutional recurrent autoencoder for IoT time series," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 52, no. 1, pp. 112–122, Jan. 2022.
[22] H. Wang, G. Liu, A. Liu, Z. Li, and K. Zheng, "DMRAN: A hierarchical fine-grained attention-based network for recommendation," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 3698–3704.
[23] H. Wang, G. Liu, Y. Zhao, B. Zheng, P. Zhao, and K. Zheng, "DMFP: A dynamic multi-faceted fine-grained preference model for recommendation," in *Proc. IEEE Int. Conf. Data Mining*, 2019, pp. 608–617.
[24] S. Tuli, G. Casale, and N. R. Jennings, "TranAD: Deep transformer networks for anomaly detection in multivariate time series data," in *Proc. VLDB Endowment*, 2022, pp. 1201–1214.
[25] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-aware automated quantization with mixed precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8612–8620.
[26] Z. Liu, H. Mu, X. Zhang, and Z. Guo, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3295–3304.
[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[28] C. Hao et al., "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. 56th Annu. Des. Automat. Conf.*, 2019, pp. 1–6.
[29] J. Chen, S. Hong, W. He, J. Moon, and S.-W. Jun, "Eciton: Very low-power LSTM neural network accelerator for predictive maintenance at the edge," in *Proc. 31st Int. Conf. Field-Programmable Log. Appl.*, 2021, pp. 1–8.
[30] B. Chen, A. Bakhshi, G. Batista, B. Ng, and T.-J. Chin, "Update compression for deep neural networks on the edge," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 3076–3086.
[31] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256 Kb memory," in *Proc. 36th Conf. Neural Inf. Process. Syst.*, 2022.

Yuting Sun received the master's degree in data science from the University of Queensland, Brisbane, QLD, Australia, in 2019. She is currently working toward the Ph.D. degree in computer science at the University of Queensland.

Her research interests include data mining, time series anomaly detection, and spatiotemporal modeling.

Tong Chen (Member, IEEE) received the Ph.D. degree in computer science from The University of Queensland, Brisbane, QLD, Australia, in 2020.

He is currently a Lecturer with the Data Science Research Group, School of Information Technology and Electrical Engineering, The University of Queensland. His research interests include data mining, recommender systems, user behavior modeling, and predictive analytics.

Dr. Chen is the recipient of the 2023 ARC Discovery Early Career Researcher Award (DECRA).

Quoc Viet Hung Nguyen received the master's and Ph.D. degrees in Computer Science from the Swiss Federal Institute of Technology Lausanne (EPFL), Lausanne, Switzerland.

He is currently a Senior Lecturer with Griffith University, Southport, Australia. His research interests include data integration, data quality, information retrieval, trust management, recommender systems, machine learning, and big data visualization, with special emphasis on web data, social data, and sensor data.

Dr. Nguyen was the recipient of the Australia Research Council Discovery Early-Career Researcher Award, in 2020.

Hongzhi Yin (Senior Member, IEEE) received the Ph.D. degree in computer science from Peking University, Beijing, China.

He is an Associate Professor with The University of Queensland, Brisbane, QLD, Australia. He authored or coauthored more than 240 papers in the top conferences and prestigious journals and received H-Index 58. He has made notable contributions to the fields of recommendation systems, graph learning, and edge intelligence and has received numerous awards and recognition for his research achievements.

Prof. Yin was the recipient of the Peking University Distinguished Ph.D. Dissertation Award 2014, the Discovery Early Career Researcher Award (DECRA) 2016, Best Paper Award Nomination at International Conference on Data Mining 2018, the Best Paper Award at International Council for Open and Distance Education 2019, Best Student Paper Award at International Conference on Database Systems for Advanced Applications 2020, Australian Research Council Future Fellowship 2022, and Best Paper Award Runner-up at WSDM 2023. He has been named to IEEE Computer Society's AI's 10 to Watch 2022 and Field Leader of Data Mining and Analysis in the Australian Research 2020 magazine. He was featured among the 2022 and 2021 Stanford's World's Top 2% Scientists.