The Journal of Systems and Software 217 (2024) 112167

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

PMTT: Parallel multi-scale temporal convolution network and transformer for predicting the time to aging failure of software systems☆

Kai Jia [a,b], Xiao Yu [a,b], Chen Zhang [a], Wenzhi Xie [a], Dongdong Zhao [a,b], Jianwen Xiang [a,b,*]

[a] Engineering Research Center of Transportation Information and Safety, ERCTIS, MoE of China, Hubei Key Laboratory of Transportation of Internet of Things, School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan 430070, Hubei, China
[b] Chongqing Research Institute, Wuhan University of Technology, Chongqing 401120, China

ARTICLE INFO

Keywords:
Software aging
Rejuvenation scheduling
Performance degradation
TTAF prediction
Temporal convolution network
Transformer

ABSTRACT

Software aging is one of the significant factors affecting the reliability and availability of long-running software systems, such as Android, Cloud systems, etc. The time to aging failure (TTAF) prediction for software systems plays a crucial role in proactive rejuvenation scheduling through machine learning or statistical analysis techniques, due to its ability to determine when to perform rejuvenation to mitigate the aging effects. However, software aging characterization is relatively complicated, and only fitting the variations for a single aging indicator cannot grasp the comprehensive degradation process across different case systems; moreover, since software systems often exhibit long and short-term inherent degradation characteristics, existing prediction models possess a poor ability for modeling both global and local information simultaneously. To tackle the above problems, a novel TTAF prediction framework based on the parallel multi-scale temporal convolution network and transformer (named PMTT) is proposed, by mapping various system running indicators reflecting the software aging to TTAF. PMTT possesses the following distinctive characteristics. First, a local feature extraction module that contains multiple channel TCNs with different scales is developed to extract inherent local information from the raw input. Second, in a parallel manner, a global feature extraction module integrating transformer blocks is built to extract global information representation synchronously using the self-attention mechanism. Afterward, high-level global–local features extracted from different channels are fused, and TTAF is estimated through two fully connected regression layers using the fused features. The proposed PMTT has been compared to seven competitors using run-to-failure data collected from Android and OpenStack systems. The experiments have demonstrated the superiority of PMTT, showing an average improvement of 11.2%, 9.0%, and 9.3% in performance across three evaluation metrics compared with the optimal baseline model.

1. Introduction

Software aging, a phenomenon observed in long-running software systems, is primarily attributed to the accumulation of errors and the depletion of computational resources, such as physical memory (Cotroneo et al., 2014), within the system. This phenomenon occurs in many software systems, including Linux system (Cotroneo et al., 2010), Android system (Cotroneo et al., 2022; Qiao et al., 2019; Jia et al., 2022), Java virtual machine (Cotroneo et al., 2007), Web services (Grottke et al., 2006), deep learning framework (e.g., TensorFlow (Du et al., 2020)), Internet-of-Vehicle (Bai et al., 2021), Cloud/edge computing (Andrade et al., 2023; Pietrantuono and Russo, 2020; Bai et al., 2023), etc. To address this trouble posed by aging effects, a measure named software rejuvenation was put forward in 1995 (Huang et al., 1995). Software rejuvenation (Huang et al., 1995) strives to alleviate the aging effects by proactively refreshing the internal conditions of the system, effectively restoring them to a state similar to when the system was newly deployed, minimizing the influence degree of aging on its performability and reliability. Therefore, as a critical component of the whole software rejuvenation schedule (Cotroneo et al., 2022; Levitin et al., 2020), an accurate time to aging failure (TTAF) prediction can effectively achieve better rejuvenation decision-making before the failure occurs to the software systems, thus reducing costs as well as

☆ Editor: Raffaela Mirandola.

* Corresponding author at: Engineering Research Center of Transportation Information and Safety, ERCTIS, MoE of China, Hubei Key Laboratory of Transportation of Internet of Things, School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan 430070, Hubei, China.
E-mail addresses: jiakai@whut.edu.cn (K. Jia), xiaoyu@whut.edu.cn (X. Yu), zhangchenorange@whut.edu.cn (C. Zhang), xiewenzhi@whut.edu.cn (W. Xie), zdd@whut.edu.cn (D. Zhao), jwxiang@whut.edu.cn (J. Xiang).

https://doi.org/10.1016/j.jss.2024.112167
Received 29 February 2024; Received in revised form 4 June 2024; Accepted 25 July 2024
Available online 31 July 2024
0164-1212/© 2024 Elsevier Inc. All rights are reserved, including those for text and data mining, AI training, and similar technologies.

improving reliability and avoiding catastrophic accidents (Dohi et al., 2018; Marshall, 1992). In this study, TTAF refers to the remaining usage time of a software system that gradually degrades in performance and eventually fails due to aging effects. However, due to the labile workload conditions and the complicated degradation process of software systems, how to exploit useful system running condition information and accurately predict TTAF is still a challenging task.

Currently, several models have been employed for the prediction tasks of software aging, such as the autoregressive integrated moving average model (ARIMA) (Pereira et al., 2018), multi-layer perceptron (MLP) (Yan, 2020), and long short-term memory (LSTM) (Qiao et al., 2018; Vinícius et al., 2022), and among others. However, the task of software aging prediction is challenging, since the aging indicator data is dynamically changed in various workloads and possesses complex interdependent correlations between aging indicators. Traditional prediction models, such as the auto-regressive (AR) (Grottke et al., 2006) model, autoregressive moving average model (ARMA) (Li et al., 2002), and ARIMA (Pereira et al., 2018), have been widely developed for software aging prediction. Nevertheless, these models typically assume a linear correlation among the variations in aging indicator data. Meanwhile, they may not fully leverage the potential interdependence between different aging indicators.

Moreover, traditional machine learning techniques, e.g., MLP (Yan, 2020), are also widely used in software aging prediction tasks. However, these models can sometimes encounter challenges during the training process, as they are prone to getting trapped in local optimal solutions. This can potentially give rise to a decrease in the prediction performance of the model. Recently, deep learning techniques, known for their powerful representation learning ability, have also been employed in aging prediction tasks. These models include recurrent neural network (RNN) (Meng et al., 2021) and its variants, such as long short-term memory (LSTM) (Qiao et al., 2018; Vinícius et al., 2022) and gate recurrent unit (GRU) (Jia et al., 2023), convolutional neural network (CNN) (Tan and Liu, 2021), and their hybrid models (Tan and Liu, 2021; Liu et al., 2019). In these models, CNNs are applied to model spatial relationships in aging indicator data (Tan and Liu, 2021). RNN or its variants are utilized to extract temporal dependencies in aging indicator data (Qiao et al., 2018; Meng et al., 2021). For the existing aging prediction models mentioned above, they use system running indicators as the input variables, since these indicators, e.g., CPU (Tan and Liu, 2021; Liu et al., 2019), Free memory (Qiao et al., 2018; Vinícius et al., 2022; Jia et al., 2023), swap space (Grottke et al., 2006), Proportional Set Size (PSS) of system_server (Cotroneo et al., 2022), and response time (Meng et al., 2021), can be used to represent the software aging process. However, existing models only take the single indicator as input and predict future evolution through the historical information of this indicator, and as we expected, achieve satisfactory prediction accuracy. But for decision-makers, a precise rejuvenation schedule will need to be further combined with carefully designed empirical thresholds. When the predicted result of this indicator reaches the predefined aging threshold, the rejuvenation actions can be triggered as needed (Levitin et al., 2020; Tan and Liu, 2021), but thresholds will not be changeless across different software systems. Therefore, for existing prediction-based rejuvenation scheduling, two open problems still need to be solved urgently.

The ***first problem*** is the complicacy of software aging characterization (Cotroneo et al., 2022, 2019b). Software aging prediction based on a single indicator may not fully capture the monolithic tendency of the aging process, leading to untimely action in the rejuvenation schedule, e.g., missed or false alarms. For example, Cotroneo et al. (2022, 2019b) highlighted the importance of assigning different confidence levels to every aging indicator. Additionally, the variations in a single indicator, such as seasonality (Grottke et al., 2006; Jia et al., 2023), heavily depend on the intensity of load and usage behavior mode of the user. Therefore, short-term transient variations, such as sudden threshold exceedances, may not confirm the presence of aging effects. Consequently, the determination of aging indicators used for aging prediction is usually case by case. For example, some literature focuses on memory indicators (Jia et al., 2023; Meng and Zhang, 2024) as model input, while others may focus on CPU (Tan and Liu, 2021; Liu et al., 2019) or response time (Meng et al., 2021; Meng and Zhang, 2024), and many others. However, since different aging indicators may exhibit different characteristics, e.g., linearity, non-linearity, and seasonality, the prediction method designed for the characteristics contained in a specific indicator may not be directly applicable to the aging prediction task of other systems (i.e., aging indicators on other systems may not appear similar features), thereby limiting the applicability of existing models. To address this, it is necessary to capture the overall degradation trend from raw running data of software systems, avoiding bias towards a specific aging indicator.

The ***second problem*** is that, the validity of these models has yet to be verified when processing the multi-dimensional system running data as input. Generally, system running data tends to contain long-term and short-term potential degradation features simultaneously (Jia et al., 2023) hidden in the entire life cycle, which needs the designed model to learn both global degradation features and local context information to achieve better prediction. Besides, the raw system running data often contains local random noises due to the complex workload conditions. However, RNN or its variants suffer reduced computational efficiency due to their lack of parallel computing capability. Additionally, they face the challenges of the gradient vanishing or exploding, when capturing long-term temporal dependency (Wang et al., 2024). These issues may give rise to the loss of relevant and vital degradation-related information in the prediction process. Moreover, CNN-based methods often have limited modeling ability for global and temporal dependencies because of the restricted receptive field imposed by convolution kernel sizes (Fu et al., 2024; Zhang et al., 2023a). As a result, they may struggle to extract long-term degradation and temporal features effectively (Zhang et al., 2023d; Deng et al., 2022). Therefore, the developed model needs to possess comprehensive global and local feature extraction capacities to achieve better predictions.

To this aim, this work proposes a novel framework, named PMTT, by combining the multi-scale temporal convolution network (TCN) and transformer to equip the global and local features extraction ability to tackle the problems above in TTAF prediction tasks. TCN weakens the impact of limited receptive fields and can effectively model inherent local features and temporal dependencies (Fu et al., 2024; Bai et al., 2018), which is widely applied in natural language processing (NLP) (Bai et al., 2018), intelligent transportation systems (Sheng et al., 2022), prognostic and health management (Fu et al., 2024) and other research fields (Zhao et al., 2022). TCN adopts hierarchically structured temporal convolutional filters (e.g., dilated one-dimensional convolutions) with larger temporal receptive fields and fewer parameters to compensate for CNN's weaknesses (Bai et al., 2018). However, TCN only captures localized parts of the input sequence by the convolutional filters. It presents that the extracted features by TCN are still local features, but the ability to extract global features for TCN is an intractable task (Wang et al., 2024; Peng et al., 2023). The transformer network (Peng et al., 2023; Vaswani et al., 2017) is a novel model that only utilizes the self-attention mechanism to substitute recurrent or convolutional layers for sequence data modeling tasks. The transformer is prevailing in many research scenarios, e.g., NLP (Vaswani et al., 2017), sequence prediction (Zhou et al., 2021), etc. In our prediction tasks, it enables the network to handle any part of the system running data without distance constraints, which urges it to be more capable of modeling long-term dependency couplings.

In the proposed framework with PMTT, we improve the traditional feature extraction technological process to connect various subnetworks serially. In contrast, we adopt a parallel architecture to perform TTAF predictions. Specifically, we build three TCN channels

Table 1
Comparisons of difference between the existing models and our proposed model.

Categories	Models	System type	Input	Output
Statistical analysis models	AR (Grottke et al., 2006)	Web server	Swap space	Values of future resource consumption
	ARIMA (Pereira et al., 2018)	Android/Eucalyptus	Memory	
	ARMA (Li et al., 2002)	Web server	Swap space	
	Nonlinear dynamic model (Jia et al., 2008)	Web server	Load, buffer, swap space	
	Linear trend model (Araujo et al., 2016)	Streaming video player	Resident memory, virtual memory[a]	
Machine learning models	LSTM (Qiao et al., 2018)	Android system	Memory	
	LSTM (Vinícius et al., 2022)	Docker platform	Memory	
Hybrid models	MLP with ridge and GSO (Yan, 2020)	Commercial applications	Memory	
	ARIMA-RNN (Meng et al., 2021)	Cloud	CPU, response time[a]	
	GRU with time series decomposition (Jia et al., 2023)	Cloud	Memory	
	CNN-LSTM-Attention (Tan and Liu, 2021)	Virtual machine monitor	CPU, memory[a]	
	ARIMA-LSTM (Liu et al., 2019)	Cloud	CPU	
Ours	Parallel multi-scale TCNs and transformer	Cloud/Android system	Various running condition indicators	TTAF

[a] Indicates that the work uses different aging indicators as various cases to measure model performance.

with different filter sizes to respectively extract fine-grained local feature information at different scales from the raw data. These TCN channels can adopt dilated convolution to extend the receptive field to model temporal dependencies and combine multi-scale convolution to process different scale local features. In parallel, we build the transformer blocks to emphasize the helpful features while suppressing the useless features, thereby reaching the purpose of extracting global degradation information. In this way, the loss of useful information can be reduced effectively to increase the robustness of the model on the premise of intact information. Subsequently, the extracted high-level features of different modules are fused and further input to two fully connected layers to achieve TTAF prediction. The numerical analysis has verified the effectiveness of PMTT, showing an average improvement of 11.2%, 9.0%, and 9.3% in prediction performance across three evaluation metrics, including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-square (R^2), compared with the optimal baseline model.

The contributions in this paper can be summarized as below:

- A TTAF prediction framework with PMTT is proposed to synchronously extract global and local degradation information hidden in raw aging failure data of software systems, which constructs a mapping to estimate TTAF.
- Multi-scale TCNs are constructed to enhance the feature extraction capability of single-scale TCNs to extract local feature information and temporal dependencies at different scales. The transformer uses the self-attention mechanism in a parallel manner to extract global degradation features, which can promote the TTAF prediction accuracy.
- We collect multiple sets of run-to-failure data as experimental datasets on two mainstream software systems. The proposed PMTT is confirmed to be effective, compared with its seven competitors on six TTAF prediction tasks.

The rest of the paper is organized as follows. In Section 2, we introduce the research on software aging and rejuvenation. Section 3 details the construction process of the PMTT framework, while Section 4 analyzes the numerical results of the models by performing ablation experiments and sensitivity analysis. Finally, Section 5 expounds on the threats to validity, and Section 6 includes the conclusion and further work plan.

2. Background and related works

2.1. Background

In a continuously running system, the activation of aging-related bugs (ARBs) is the main cause of software aging, leading to gradual resource depletion, increased failure rates, and system crashes (Qin et al., 2023b). The unplanned system failure caused by aging will reduce user experience, increase system maintenance costs, and even endanger personnel lives (Marshall, 1992).

Currently, there are many studies devoted to discovering software bugs by developing bug prediction methods and reported relatively high accuracy in their works (Qin et al., 2023b; Yang et al., 2024; Zhang et al., 2023c). For instance, Qin et al. (2023b) proposed an ARB prediction method by constructing an aging-related dependency network to model the flow of aging-related information in the software. Yang et al. (2024) studied the unsupervised effort-aware defect prediction by using the K-medoids clustering technique. Zhang et al. (2023c) propose an improved Fuzzy C-means clustering method named IFCM, designed to mitigate class overlap in ARB prediction tasks. However, locating and repairing ARBs during the development phase is still a challenging task due to the complexity of their underlying causes and the requirement for prolonged execution to manifest (Cotroneo et al., 2013). During the testing and debugging phases, the low reproducibility of this type of bug makes existing testing and debugging techniques costly and even ineffective (Nie et al., 2024; Yu et al., 2023). Therefore, some proactive preventive maintenance measures have also received widespread attention during the software deployment phase (Huang et al., 1995; Meng and Zhang, 2024).

2.2. Related work

Software aging and rejuvenation (SAR) have continued to be studied for nearly 30 years (Huang et al., 1995). To date, many measures, e.g., prediction-based and threshold-based methods, have been proposed to schedule reasonable rejuvenation actions (Cotroneo et al., 2014; Levitin et al., 2020). The rejuvenation actions can be implemented at different levels, including application-level restart (Qiao et al., 2019), virtual machine (VM) restart (Machida et al., 2013, 2014), OS reboot (Xiang et al., 2020), cluster failover (Wang et al., 2007), Java container rejuvenation (Cotroneo et al., 2022), and among others. Currently, some prediction methods have been broadly adopted, because they are capable of giving auxiliary information regarding the performance evolution of the system by prediction (Jia et al., 2023; Tan and Liu, 2021; Liu et al., 2019). In this part, we retrospect the related work from three directions: statistical analysis-based, machine learning-based, and hybrid models in software aging prediction. Table 1 summarizes the difference between our work and existing works from 4 dimensions, i.e., models, analyzed systems, input, and output, and the difference summary of our work is also described.

Statistical analysis models: There is a long history of software aging prediction using statistical models. Li et al. (2002) applied the ARMA models to predict resource exhaustion of swap space indicator, and this model was confirmed to be more effective and less computationally complex than previous works. Grottke et al. (2006) adopted the AR model to perform the resource usage prediction on the web server, and they emphasized that based on this model, proactive rejuvenation can be triggered. Later, Jia et al. (2008) proposed to use the non-linear dynamic model to predict resource usage, and the simulation results

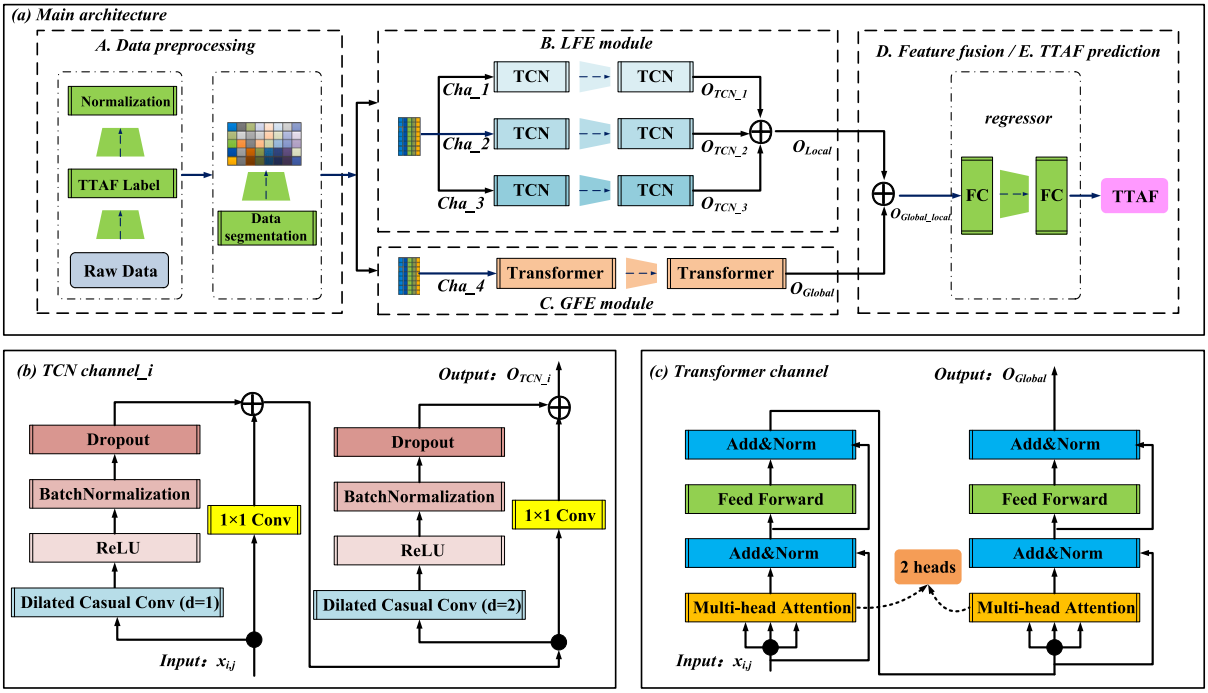


Fig. 1. The working flow of PMTT, (a) main architecture, (b) TCN channel, and (c) Transformer channel.

can roughly reflect the evolution of software aging. Recently, Araujo et al. (2016) employed four time-series models, such as the linear trend model and quadratic trend model, to analyze the software aging issues in streaming video player. Pereira et al. (2018) also applied six time-series models to implement prediction on Android and Eucalyptus systems. They confirmed that ARIMA has a lower prediction error among these models, and the rejuvenation moment is determined based on the prediction.

Machine learning models: For the Android system, Qiao et al. (2018) adopted the LSTM model to perform aging prediction using memory indicators, and numerical results confirmed that the prediction accuracy of LSTM outperforms comparative models, such as ARIMA and MLP. Lately, for the docker platform, Vinícius et al. (2022) also adopted the LSTM model to predict the behaviors of software aging. In their study, they compared the prediction ability of LSTM with other models, such as ARIMA, Drift, Holt, and Holt-Winters, by using memory usage as the aging indicator, and finally, the effectiveness of LSTM is verified.

Hybrid models: Hybrid aging prediction models are more prevalent among researchers because they possess satisfactory prediction ability. For example, Liu et al. (2019) proposed the ARIMA-LSTM prediction model, which effectively captures both linear and non-linear variations in aging indicator data. Comparative experiments demonstrated that the ARIMA-LSTM model outperforms both the ARIMA and LSTM models in terms of prediction accuracy. Yan (2020) introduced a combination of MLP with the ridge and glowworm swarm optimization (GSO) algorithm to enhance the prediction capability of the traditional MLP model. Recently, Tan and Liu (2021) combined CNN, LSTM, and attention mechanism serially to construct the aging prediction model ACML, which is capable of modeling the temporal–spatial features contained in the aging indicator (e.g., memory or CPU utilization). Similar to Liu's work (Liu et al., 2019), Meng et al. (2021) integrated ARIMA and RNN models to perform cloud server aging prediction; however, the proposed model used the prediction output of ARIMA and historical window data as input of the RNN simultaneously, achieving better performance. Jia et al. (2023) integrated GRU and time series decomposition techniques to tackle the seasonal variations in memory resource usage for cloud services, and the superiority of their model is verified by performing extensive experiments.

Summary: Different from these works that use a specific aging indicator in a single input manner without considering multiple system running indicators and only predict future evolution for this indicator, in this work, we use different techniques to capture the complicated degradation progress for software systems, and the output is specific TTAF value, which gives more intuitive auxiliary information for when to scheduling rejuvenation. According to the comprehensive effects of different modules, a thorough experimental evaluation is performed to analyze the final prediction performance of the proposed PMTT model.

3. Methodology

In this section, we detail the constructed framework PMTT to handle the problem in TTAF prediction for software systems. The framework PMTT constructs the relational mapping directly from the system running indicators $x_{n,m}$ to the $TTAF_n$, as depicted in Eq. (1). It should be mentioned that the prediction target of PMTT is the specific TTAF value, which is essentially a regression prediction problem, which is different from the target of the classification task. The whole architecture of PMTT is shown in Fig. 1(a), which consists of five main modules, including data preprocessing, global feature extraction module, local feature extraction module, feature fusion module, and TTAF prediction module.

$$\begin{bmatrix} x_{1,1},x_{1,2},x_{1,3},\ldots,x_{1,m} \\ x_{2,1},x_{2,2},x_{2,3},\ldots,x_{2,m} \\ x_{3,1},x_{3,2},x_{3,3},\ldots,x_{3,m} \\ \cdots \\ x_{n,1},x_{n,2},x_{n,3},\ldots,x_{n,m} \end{bmatrix} \xrightarrow{F(\Phi)} \begin{bmatrix} TTAF_1 \\ TTAF_2 \\ TTAF_3 \\ \cdots \\ TTAF_n \end{bmatrix} \tag{1}$$

First, we perform the data preprocessing operations before inputting them to networks. This process includes TTAF labeling, normalization, and slide window sampling techniques. Then, we separately construct the GFE module and LFE module in a parallel manner. The GFE adopts stacked transformer layers to capture the global degradation information of raw data. Meanwhile, the LFE adopts multiple channels with stacked TCN layers to extract multi-scale local information and temporal dependencies hidden in raw data. Afterward, the extracted

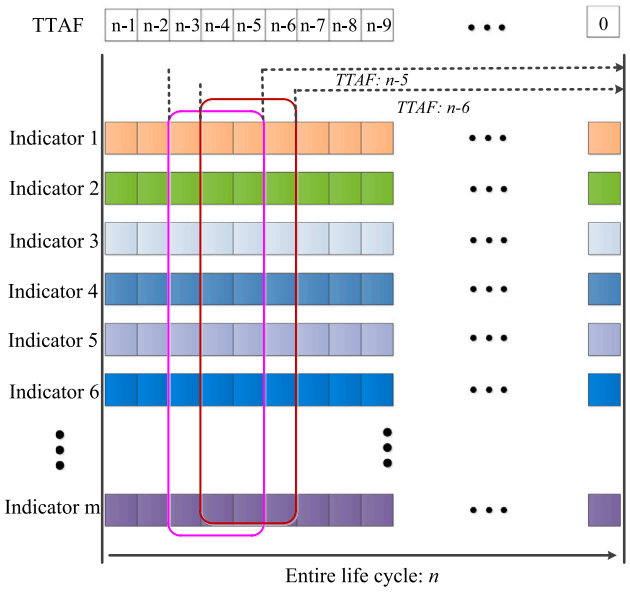


Fig. 2. The illustrative example of TTAF label.

features from two modules are further fused to complete the extraction process of global–local features effectively. Finally, the fused high-level feature representations are fed into the TTAF prediction module built by two fully-connected layers, to obtain the predicted TTAF value.

3.1. Data preprocessing

TTAF labeling: We collect system running condition data, such as Free Memory, CPU, Swap space, and load, during the whole life cycle for tested software systems, and designate n as the number of run-to-failure samples. We label the ground-truth label of each sample, which represents the actual TTAF value at moment t, as depicted in Fig. 2. Each sample that includes the observed values of different indicators at the same moment in Fig. 2 corresponds to a real TTAF and is labeled in the manner from back to front. Among them, the direction of the arrow represents the evolution direction of time, indicator m represents the collected system indicators, and L represents the time lag (window). Therefore, the TTAF value for the ith data sample can be logically set to $n - i$. For example, the TTAF label of a sample $[x_{1,1},x_{1,2},x_{1,3},...,x_{1,m}]$ corresponds to $n - 1$.

Data Normalization: Because the raw data originated from the software systems containing different running indicators, so these data have different scales. To reduce the impact on model training effects, we use data normalization operations to constrain these data within a certain range (Wang et al., 2024). We designate m to represent the number of collected indicators, and designate n to indicate the data length. Formally, the raw system running indicators data can be represented as $X = [x_1,...,x_i,...,x_n]^T$, $x_i = [x_{i,1},\dots,x_{i,j},...,x_{i,m}]$, where $x_{i,j}$ indicates the jth indicator at the ith moment. This paper adopts min–max normalization (Wang et al., 2024) to scale the raw data in the scope for $[0, 1]$ as Eq. (2).

$$x_{(i,j)norm} = \frac{x_{(i,j)} - x_{(i,j)min}}{x_{(i,j)max} - x_{(i,j)min}} \tag{2}$$

where $x_{(i,j)}$ and $x_{(i,j)norm}$ represent the raw and scaled data, $x_{(i,j)max}$ and $x_{(i,j)min}$ represent the upper and lower limits of the sequence, respectively. In addition, TTAF data also needs to be normalized to the scope for $[0, 1]$, using Eq. (3).

$$TTAF_{i-norm} = \frac{TTAF_i - TTAF_{min}}{TTAF_{max} - TTAF_{min}} \tag{3}$$

where the value of $TTAF_{i-norm}$ is between 0 and 1.

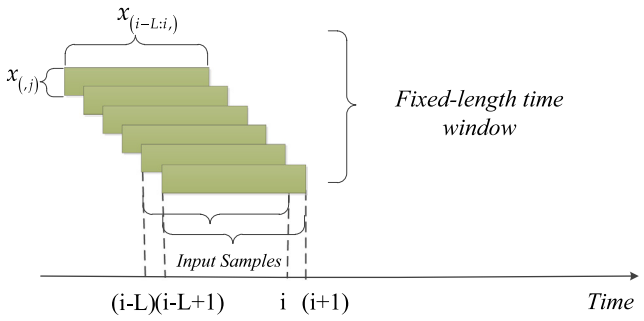


Fig. 3. A schematic diagram of the slide window sampling.

Slide window sampling: To fully exploit the temporal dependency in time series and increase training samples, this paper uses the slide window sampling technique (Fu et al., 2024) to create the input features for the model in a consecutive time manner, as displayed in Fig. 3. During the process, the input features for training consists of data samples within the time window, while the target vector to be predicted is the true TTAF at the current moment (Fu et al., 2024).

3.2. Local Feature Extraction (LFE)

This part processes fine-grained local features and temporal dependencies hidden in the system running data, and we use TCNs to model the above two relationships. TCN, as a variant of classical CNN architecture, enables modeling the long-range temporal relationships in the sequence data, in comparison with the traditional CNN architecture (Bai et al., 2018). However, single-scale TCNs that adopt a fixed filter size, can only learn feature representations at a specific scope of scales, and the feature information at other scales has to be discarded.

In our prediction task, we build an LFE module containing multi-channel TCNs with different kernel sizes to capture local features at different scales, respectively, making the extracted information more intact. As shown in Fig. 1(a), three parallel TCN channels are used to extract local information and temporal dependencies from the raw input data. TCN architecture we used in this paper is given in Fig. 1(b), which includes two residual connection blocks, and each block has a dilated causal convolutional (DCC), normalization, activation, and dropout layers. The architecture for the DCC is shown in Fig. 4, in which dilated coefficient d is set to 1, 2, and 4, and kernel size is set to 3, as an example. Moreover, 1×1 convolution for the cross-layer path is utilized, since the input and output dimensions in the residual connection block are disparate. As an essential component of TCN, causal convolution under causal constraint ensures that the element of the next layer at moment t only relies on the element of the preceding layer at moment t and the elements in front of it. While the dilated convolution works by following the concept of dilation, which controls the CNN's filter to sample at intervals based on the dilated coefficient d across a more extensive range, thereby increasing the receptive field. Given an input sequence $x \in R^n$, for a filter f, the calculation of DCC for element s at moment t is formulated as in Eq. (4) (Bai et al., 2018):

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{p-1} f(i) \cdot x_{s-d \cdot i} \tag{4}$$

where d represents the dilated coefficient, p is the kernel size, $*$ indicates the convolution operation, and $s - d \cdot i$ accounts for the orientation of the past. For the lth layer DCC of a residual block for a specific channel, the calculation is as follows:

$$DCC^{(l)} = Conv1D(W^{(l)}, b^{(l)}, x_{i,j}, kernel\_size = p_c, dilation\_rate = d) \tag{5}$$

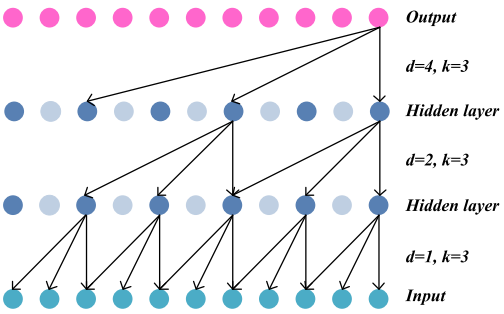


Fig. 4. A diagram of dilated causal convolution.

After normalization and dropout layers, the final output result of *DCC* is obtained. Finally, the output after 1 × 1 convolution in the residual connection is added to *DCC* to get the final output O_{TCN}, as in Eq. (6).

$$O_{TCN} = DCC + F\left(x_{i,j}\right) \tag{6}$$

In this study, the number of TCN blocks for each channel is set to 2. The structure of each block is identical, but the sizes of the convolution kernel for the three channels are diverse (i.e., $p_1 \neq p_2 \neq p_3$), which allows the LFE module to extract the local feature of different scales, and the calculation process follows Eqs. (5)–(6). Finally, the outputs of the three channels can be indicated as O_{TCN_1}, O_{TCN_2}, and O_{TCN_3}, respectively. Then, these outputs can be fused to obtain the extracted local feature O_{Local} for the LFE module, as in Eq. (7). These extracted local features at different scales will be used for further information fusion, as described in the following subsection.

$$O_{Local} = \text{Add}\left(O_{TCN\_1}, O_{TCN\_2}, O_{TCN\_3}\right) \tag{7}$$

3.3. Global Feature Extraction (GFE)

As mentioned above, TCNs have the ability to model local features and temporal dependencies. However, when processing long time series, the TCN struggles to extract global degradation features, since it keeps the weight for every input variable invariable during the feature extraction process. To compensate for the above weakness in feature extraction by the TCN, the transformer network is adopted to extract global degradation features (Vaswani et al., 2017). Further, this paper adopts the parallel architecture to capture the global and local degradation features separately. This multi-channel pattern can mitigate the risk of information interference with each other to enhance the model's feature extraction capabilities, thereby enabling better predictions.

The transformer network is designed with an encoder–decoder architecture, utilizing stacked multi-head self-attention and feed-forward layers (Vaswani et al., 2017). Nevertheless, the decoder part has a more complicated construction and often requires the output of the historical sequence, resulting in significant computational costs and memory usage (Peng et al., 2023; Zhang et al., 2021, 2023b). Therefore, this paper only applies the encoder part for the traditional transformer architecture, making the model structure lighter and more suitable for global feature learning in TTAF prediction. The encoder is constructed by stacking multiple identical blocks, as depicted in Fig. 1(c), and the residual connection is used to connect different layers, followed by the normalization operation.

(1) Multi-head self-attention: The core building block of the transformer network is multi-head self-attention (MSA), which is used to calculate the similarity between the input sequence (Zhang et al., 2023b). First, a set of input x is embedded and further transformed to d_{model}-dimensional keys, values, and queries, as below:

$$Q = xW^q, K = xW^k, V = xW^v \tag{8}$$

where $W^q, W^k \in R^{d_{model} \times d_k}$ and $W^v \in R^{d_{model} \times d_v}$, indicate learnable weight matrices, respectively, and d_k is the dimension of key matrices.

Next, the self-attention matrices are computed by using the scaled dot product function with scale factor $1/\sqrt{d_k}$, and the calculation process is expressed as:

$$\begin{aligned} S &= Attention(Q, K, V) \\ &= Softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \end{aligned} \tag{9}$$

Since feature extraction using single-head attention only is often not adequate, so feature information derived from diverse representation subspaces should be considered, thereby ensuring the integrity of information extraction. Here, the MSA mechanism is introduced, which performs H times calculation for the single-head attention, and the outputs of these heads are then concatenated. Thus, MSA is computed as follows:

$$\begin{aligned} MSA &= MultiHead(Q, K, V) \\ &= Concat\left(head_1, \ldots, head_i, \ldots, head_H\right) \end{aligned} \tag{10}$$

$$head_i = Attention(Q, K, V) \tag{11}$$

Among them, H represents how many heads. After the computations of the MSA mechanism are completed, the residual connection and layer normalization will be performed continuously to get the output o_{MSA} for this sub-block.

(2) Feed-forward network: To proceed, except for the MSA component, the feed-forward network is also an indispensable part for each transformer block. The feed-forward network we employed includes two fully connected layers, and uses the ReLU activation function between them, expressed as:

$$FC\left(o_{MSA}\right) = W_2 \bullet ReLU\left(W_1 \bullet o_{MSA} + b_1\right) + b_2 \tag{12}$$

where W_1 and W_2 are the weights of two fully connected layers, respectively. Moreover, the residual connection is used for stable training here, and we can get the final output O_{Global} of a transformer block:

$$O_{Global} = o_{MSA} + FC(o_{MSA}) \tag{13}$$

In this study, there are two transformer blocks designed, and each block is identical. The calculation process follows Eqs. (8)–(13).

3.4. Multi-scale information fusion

After the LFE and GFE modules with four channels complete the degradation feature extraction of software, the extracted high-level feature representations need to be fused. In the study, since different modules focus on different feature information hidden in raw input, the element-wise addition operation is adopted to merge the feature vector extracted from previous layers. This fusion procedure is given as follows:

$$O_{Global\_local} = \text{Add}\left(O_{Local}, O_{Global}\right) \tag{14}$$

By using the addition operation, the dimension of O_{Global_local} is not augmented, in comparison with the dimension of the feature extracted by each channel.

3.5. TTAF prediction

We forward these fused features O_{Global_local} into a regressor with two fully connected layers, which allows the degradation features extracted by PMTT to be mapped to TTAF labels. This prediction process can be described as:

$$TTAF_{pred} = W_2 \bullet ReLU\left(W_1 \bullet O_{Global\_local} + b_1\right) + b_2 \tag{15}$$

where $TTAF_{pred}$ is the predicted TTAF value, during the training stage, the mean square error (MSE) is used as the loss function to evaluate and optimize the network, which is computed as in Eq. (16):

$$MSE_{Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(TTAF_i^{pred} - TTAF_i^{true}\right)^2 \tag{16}$$

Among them, N indicates the sample number, then the weight parameters of the PMTT are optimized by the *Adam optimizer* (Peng et al., 2023), and we configure the learning rate to 0.001.

4. Experiment and results

In this section, we assess the prediction ability of our PMTT through extensive experiments conducted on two case software systems (i.e., Android and OpenStack), encompassing six test prediction tasks. Our evaluation aims to address the following four research questions (RQs):

- How does the prediction ability of PMTT in comparison with the state-of-the-art methods on two case software systems?
- What impact do the different key modules of PMTT have on its performance?
- What impact do different parameter configurations and feature fusion methods have on the overall performance of PMTT?
- Which aging indicator contributes to improving the prediction performance of PMTT on two case software systems?

4.1. Experimental data description and evaluation metrics

To date, we have not found any publicly available datasets that can be used for TTAF predictions. To assess the effect of the model, we implement accelerated life testing experiments on two popular software systems, i.e., Android and OpenStack, which simulate user usage behavior instead of artificially injecting aging-related bugs to generate actual workloads to accelerate software aging. The data collection process is described in detail as follows:

Android system: We utilize Monkey to generate test cases, and use the Android Debug Bridge (ADB) tool to establish a connection between the device under test and the host (Cotroneo et al., 2022). Monkey is a command-line tool developed by Google,[1] that can perform automated testing for Android systems. Monkey can generate pseudo-random streams of user events, such as trackball, touch, and motion, as well as a number of system-level events, to achieve stress testing purposes on the Android system. It allows the simulation of user actions quickly, reliably, and randomly. We perform stress testing on three applications, including Sohu News, Camera, and Reader. We continuously send 1000 events with a 300 ms time interval, including trackball, touch, and motion, to each application, and then wait 10 s before executing the stress test task of the next application, which uses the same number of events and waiting time. Concretely, the command line for the Camera application has the following form:

Stress testing script for Android

<adb shell monkey -p com.huawei.camera –ignore-security-exceptions –ignore-timeouts –pct-trackball –pct-touch –pct-motion –throttle 300 -v 1000>

The smartphone is equipped with an Android OS, version 6.0, and 3 GB of RAM. Note that, several recent studies have reported software aging issues in different Android versions, such as Android 5, 6, 7, 9, and 12 (Cotroneo et al., 2022; Nie et al., 2024; Chen et al., 2023; Cotroneo et al., 2020), empirically indicating that software aging is prevalent across different versions. In this paper, we only choose one of the versions, i.e., Android 6.0, as an example to perform stress testing, because the configuration of this version can accelerate the occurrence of aging phenomena (Cotroneo et al., 2022). We use data collection scripts by "adb shell" commands, including "dumpsys meminfo/cpuinfo/battery" and "cat /proc/vmstat", to export the logs of the system running indicators every 1 min. Finally, there are four sets of run-to-failure data are collected to perform the subsequent experiments. A detailed description of collected datasets is introduced in Table 2. It can be found that the indicators used in this article have been reported in a variety of literature (Nie et al., 2024; Chen et al., 2023; Cotroneo et al., 2020), such as memory, pgfault, swap, etc. Moreover, we select four data series in the collected system running indicators as examples to visualize, as depicted in Fig. 5.

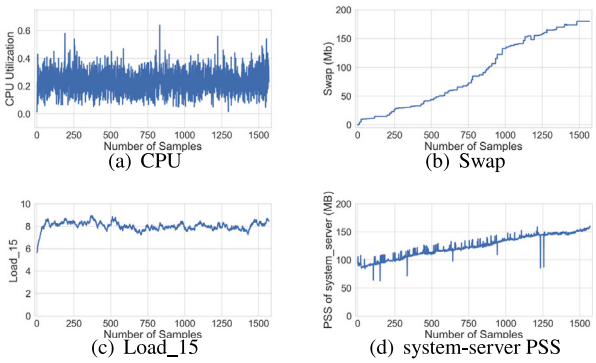


Fig. 5. Four data series selected from collected system indicators.

OpenStack platform: As the current mainstream cloud platform, OpenStack can manage the virtual or physical machines it hosts (Cotroneo et al., 2019a). The hardware and VM configuration for the testbed includes: Intel Core CPU (i5-8500 @ 3.00 GHz); 12 GB RAM; 80 GB storage; Ubuntu 20.04. To expose the extent of aging effects, a workload generation scheme is designed to accelerate system aging. The stress testing scripts are designed by creating and deleting VMs in a continuous, repetitive manner, as in Vinícius et al. (2022) and Cotroneo et al. (2019a). In our experimental scenario, we choose the number of virtual machines created each time to be 6, and the memory of each virtual machine is set to 512 MB. After each VM's creation and deletion step is completed, wait 15 s to enter the execution of the next cycle. The command line has the following form:

Stress testing script for OpenStack

- VM creation: <openstack server create –image cirros –flavor m1 –network physnet1 demo (i)>
- VM deletion: <openstack server delete (i)>

Afterward, we use the data collection scripts: "vmstat", "iostat", and "cat /proc/loadavg", to collect the system indicators in 30 s intervals. Ultimately, we also collect four sets of run-to-failure data to perform experiments, as described in Table 2.

Datasets division: The details of how the training set and test set are divided, as described in Table 3, and we perform the six groups of test tasks in total. Considering the randomness of the deep neural network, every test experiment is repeatedly executed ten times, and this paper records and reports the average results.

Evaluation metrics: We adopt three performance evaluation metrics, including Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R^2, to measure whether the model is effective, and the formulae are defined below:

$$MAE = \frac{1}{n}\sum_{t=1}^{n}\left|TTAF_t^{pred} - TTAF_t^{true}\right|; \tag{17}$$

[1] 2024. [Online]. https://developer.android.com/studio/test/monkey.

Table 2
Descriptive statistics of run-to-failure datasets.

Analyzed systems	Android		OpenStack	
System indicators	CPU, Load1, Load5, Load15, Temperature, PSS of system_server, Free memory, Swap, Used memory, Naïve Heap, Dalvik Heap, pgfault, pgfree		CPU, Load1, Load5, Load15, Free memory, inact, active, Swap, r/s, w/s	
Numbers of samples	dataset_1	1867	dataset_5	3044
	dataset_2	1570	dataset_6	2613
	dataset_3	1626	dataset_7	2918
	dataset_4	1810	dataset_8	2564

Table 3
TTAF prediction tasks for Android and OpenStack.

Systems	Task	Training set	Testing set
Android	Task_1	dataset_1	dataset_2
	Task_2	dataset_1	dataset_3
	Task_3	dataset_1	dataset_4
OpenStack	Task_4	dataset_5	dataset_6
	Task_5	dataset_5	dataset_7
	Task_6	dataset_5	dataset_8

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\left(TTAF_t^{pred} - TTAF_t^{true}\right)^2}; \tag{18}$$

$$R^2 = 1 - \frac{\sum_{t=1}^{n}\left(TTAF_t^{pred} - TTAF_t^{true}\right)^2}{\sum_{t=1}^{n}\left(\overline{TTAF}^{true} - TTAF_t^{true}\right)^2}; \tag{19}$$

For the metrics MAE and RMSE, lower values indicate better model performance. In contrast, for the metric R^2, higher values indicate better model performance, with an upper bound value of 1.

4.2. Baselines and parameters settings

Given that TTAF prediction is essentially a regression task, the proposed PMTT aims to compare with several state-of-the-art regression models on six prediction tasks. More model details are included as below:

Multilayer Perceptron (MLP) (Yan, 2020): It is also known as artificial neural network (ANN), which is capable of accurately fitting a wide range of non-linear functions. It has been applied in the field of software aging prediction (Yan, 2020) and is constructed in a three-layer network structure.

One-dimensional Convolutional Neural Network (1DCNN) (Espinosa et al., 2021): It adopts a one-dimensional convolution layer to capture local correlations in time series, which has been applied in the field of air quality prediction (Espinosa et al., 2021) and software aging prediction (Jia et al., 2023), etc. Therefore, the 1DCNN model is included in this article using a single convolution layer for comparison purposes.

Long Short-Term Memory (LSTM) (Qiao et al., 2018; Vinícius et al., 2022) and Bi-directional LSTM (BiLSTM) (Ullah et al., 2023) neural networks: They are specific variants of recurrent neural networks designed to address long-term temporal dependence issues. BiLSTM combines forward LSTM and backward LSTM to access bidirectional long-range context in time series. They have been applied in multiple research areas, e.g., software aging prediction (Qiao et al., 2018; Vinícius et al., 2022) and workload prediction (Ullah et al., 2023), and are structured in the form of a single layer.

Gate Recurrent Unit (GRU) (Jia et al., 2023; Chung et al., 2014) and Bi-directional GRU (BiGRU) (Zhang et al., 2022) neural networks: They are also specific variants of recurrent neural networks. GRU has simpler network structures than LSTM without reducing their effectiveness, and sometimes even outperforming LSTM (Chung et al., 2014). They have also been widely used in many fields, e.g., software aging prediction (Jia et al., 2023), sequence modeling (Chung et al., 2014), industrial processes prediction (Yao et al., 2023), and remaining useful life prediction (Zhang et al., 2022). They are also structured in the form of a single layer.

1DCNN-BiGRU-Attention (Wang et al., 2022): It is a hybrid model that can model local and global dependencies of sequences by integrating 1DCNN, BiGRU, and the attention mechanism, which is also receiving widespread attention in sequence prediction problems, e.g., service price prediction (Wang et al., 2022), health state prediction (Mazzi et al., 2024), and shale gas prediction (Fargalla et al., 2024).

To ensure a fair comparison, the parameters that significantly influence model performance are optimized using a grid search strategy, while other network configurations remain consistent. The parameters involved in this paper incorporate time lag (L), epoch (M), hidden unit (N), the number of filters (f), kernel size (p), and head (H). Further details regarding the models and their corresponding parameters can be found in Table 4. In addition, PMTT and all baseline methods were conducted in the identical experimental environment, which adopted a desktop equipped with a Core i5-8500 CPU and 8G RAM. To facilitate research on software TTAF prediction problems, we have made the source code available on GitHub to ensure reproducibility.[2]

4.3. Prediction performance analysis (RQ1)

To investigate the effectiveness of PMTT, we perform extensive comparison experiments with prevalent regression prediction models. We can obtain the following observation.

First, we analyze the numerical results achieved by each model on six prediction tasks, as shown in Table 5. It is not difficult to find that performing TTAF prediction is an intricate task, and all models have large errors across different prediction tasks, especially the tasks on Android systems. This may also be the reason why most existing models only focus on the prediction for a single indicator. For example, existing aging prediction models (e.g., MLP (Yan, 2020), LSTM (Qiao et al., 2018; Vinícius et al., 2022)) show weak prediction performance when generalized to the TTAF prediction task, because these models cannot comprehensively model the complex software performance degradation process. Despite this challenge, we can notice that PMTT outperforms the baseline models in terms of three metrics in most cases (12/18). Nevertheless, the baseline models cannot obtain consistent prediction performance; for example, BiGRU obtains the best performance on Task_3 (104.0/131.7/0.9342), and BiLSTM obtains the optimal prediction performance on Task_5 (53.62/65.80/0.9937), while the performance on other tasks is unsatisfactory. We can also observe that although 1DCNN has the ability to model local features, it still cannot reach satisfactory prediction performance due to its lack of global modeling ability. Furthermore, RNN variants, i.e., GRU and LSTM, perform worse than their bidirectional structures, i.e., BiGRU and BiLSTM, on some prediction tasks, which illustrates the significance of

[2] https://github.com/agingdata/JSS.

Table 4
Models and their parameter settings for each software system.

Models	Android system	OpenStack platform
MLP (Yan, 2020)	L = 10, M = 50, N = 16;	L = 10, M = 50, N = 64;
LSTM (Qiao et al., 2018; Vinícius et al., 2022)	L = 30, M = 80, N = 32;	L = 20, M = 100, N = 32;
GRU (Jia et al., 2023)	L = 5, M = 50, N = 8;	L = 20, M = 100, N = 32;
1DCNN (Espinosa et al., 2021)	L = 5, M = 50, f = 8; p = 3;	L = 10, M = 100, f = 48; p = 3;
BiGRU (Yao et al., 2023)	L = 5, M = 80, N = 8;	L = 20, M = 100, N = 64;
BiLSTM (Ullah et al., 2023)	L = 30, M = 80, N = 32;	L = 20, M = 80, N = 64;
1DCNN-BiGRU-Attention (Wang et al., 2022)	L = 5, M = 80, N = 16, f = 16, p = 3;	L = 10, M = 80, N = 32, f = 64, p = 3;
PMTT (ours)	L = 10, M = 100, N = 64, f = 16, p = (2,3,5), H = 2	L = 20, M = 100, N = 64, f = 32, p = (2,3,5), H = 2

Table 5
Prediction results of different methods on six prediction tasks.

Methods	Metrics	MLP	LSTM	GRU	1DCNN	BiLSTM	BiGRU	1DCNN-BiGRU-Attention	PMTT (ours)
Task_1	MAE	324.4	269.6	238.6	285.9	228.5	289.1	237.1	**180.7**
	RMSE	368.2	306.9	264.1	311.2	261.2	315.0	273.1	**210.1**
	R^2	0.3245	0.4982	0.6357	0.5166	0.6346	0.506	0.6058	**0.7666**
Task_2	MAE	238.9	190.6	195.0	221.0	181.1	172.9	187.5	**148.6**
	RMSE	284.5	248.6	222.1	265.3	239.1	213.3	234.8	**184.6**
	R^2	0.6192	0.6886	0.7693	0.6742	0.7238	0.7854	0.745	**0.8344**
Task_3	MAE	134.5	142.3	156.9	166.8	131.6	**104.0**	111.3	107.9
	RMSE	172.2	184.6	186.1	203.9	173.3	**131.7**	142.4	148.0
	R^2	0.8867	0.8632	0.8635	0.8426	0.8835	**0.9342**	0.9214	0.9115
Task_4	MAE	87.62	87.55	92.29	85.24	84.43	86.03	83.01	**76.62**
	RMSE	107.1	107.6	116.4	102.4	103.2	109.4	98.93	**92.09**
	R^2	0.9793	0.9794	0.9754	0.9811	0.9809	0.9785	0.9826	**0.9846**
Task_5	MAE	60.43	56.49	59.52	58.77	**53.62**	53.93	57.47	53.80
	RMSE	77.38	72.02	74.87	72.84	**65.80**	66.23	77.03	69.13
	R^2	0.9915	0.9924	0.9919	0.9923	**0.9937**	**0.9937**	0.9915	0.9930
Task_6	MAE	72.41	59.35	59.03	66.77	64.21	62.64	64.99	**57.34**
	RMSE	88.86	75.10	71.08	81.23	78.92	76.38	82.14	**70.31**
	R^2	0.9851	0.9896	0.9906	0.9877	0.9883	0.9891	0.9875	**0.9907**
Count		0	0	0	0	3	4	0	12

The best results for six prediction tasks are shown in bold.

modeling context dependence during the software degradation process. In addition, the traditional MLP model, without exception, shows poor prediction performance across six prediction tasks, since its shallow feature extraction capability cannot model complex software degradation characteristics. Interestingly, although 1DCNN-BiGRU-Attention has the ability to model local and temporal dependencies of software performance degradation, the prediction performance is still unsatisfactory. The reason may be that due to its serial architecture, the integrity of information extraction cannot be guaranteed, resulting in the loss of useful information. Compared with these models, the proposed PMTT uses a parallel architecture to extract feature representations at different scales, thereby ensuring the integrity of information extraction, which is beneficial to better predictions.

Secondly, to more intuitively demonstrate the prediction performance of PMTT and its generalization ability on the two software systems, the average improvement proportion is listed in Table 6. Overall, the proposed PMTT has considerable performance improvements compared to these baseline models. Specifically, compared to BiGRU, three performance metrics of PMTT achieve improvements of 11.2%, 9.0%, and 9.3%, respectively, and compared to BiLSTM, PMTT achieves improvements of 12.7%, 12.3%, and 6.6%, respectively. Meanwhile, for the hybrid model 1DCNN-BiGRU-Attention, its average prediction performance is even worse than that of BiGRU. As mentioned above, the main reason is that its serial architecture uses layer-by-layer feature extraction that easily causes useful information loss. In addition, compared with MLP, our model can achieve more considerable improvements, which are 24.4%, 22.9%, and 29.2%, respectively. In conclusion, numerical results verify the superiority of our proposed PMTT, since the well-designed structure can capture abundant global and local feature information from the software degradation process.

Finally, Fig. 6 shows the fitting effect of PMTT and its comparison model on Task_4. We can see from the intuitive fitting results that

Table 6
The proportion of average improvement of PMTT in six prediction tasks.

	Metrics	MLP	LSTM	GRU	1DCNN	BiLSTM	BiGRU	1DCNN-BiGRU-Attention
Imp.	MAE	24.4%	16.6%	18.1%	22.9%	12.7%	11.2%	12.2%
	RMSE	22.9%	17.0%	14.6%	19.8%	12.3%	9.0%	12.0%
	R^2	29.2%	13.6%	5.9%	13.5%	6.6%	9.3%	6.4%

in the early prediction stages, all models get astray due to complex and dynamic software degradation characteristics. This is also the main reason why the existing TTAF prediction model has large errors. Even so, in the later stages of the prediction, PMTT is able to "get on track", maintaining smaller change amplitudes with real TTAF. Instead, other comparative models still show large fluctuations, cannot achieve a more stable fitting effect, and are unsuitable for the current tasks. Moreover, to verify the performance overhead for each model, we analyze and evaluate the training time and testing time of all models on the OpenStack prediction tasks, and the specific details are shown in Fig. 7. We have the following observations from Fig. 7. Under the same training set, most of the models took a long time to train offline, especially for PMTT, which took an average of 158.75 s to train on the three prediction tasks. It should be mentioned that the model training time is also affected by the size of the time lag and epoch, etc. That is, the larger the epoch, the longer it takes; the larger the time window means that the data length of the input model increases, which also increases the computational complexity of the model. Table 7 includes the trainable parameter size of all models under the prediction tasks of OpenStack, which is printed using the summary() method in the Keras framework. We find from Table 7 that PMTT does not have large differences in model parameters compared with most baseline

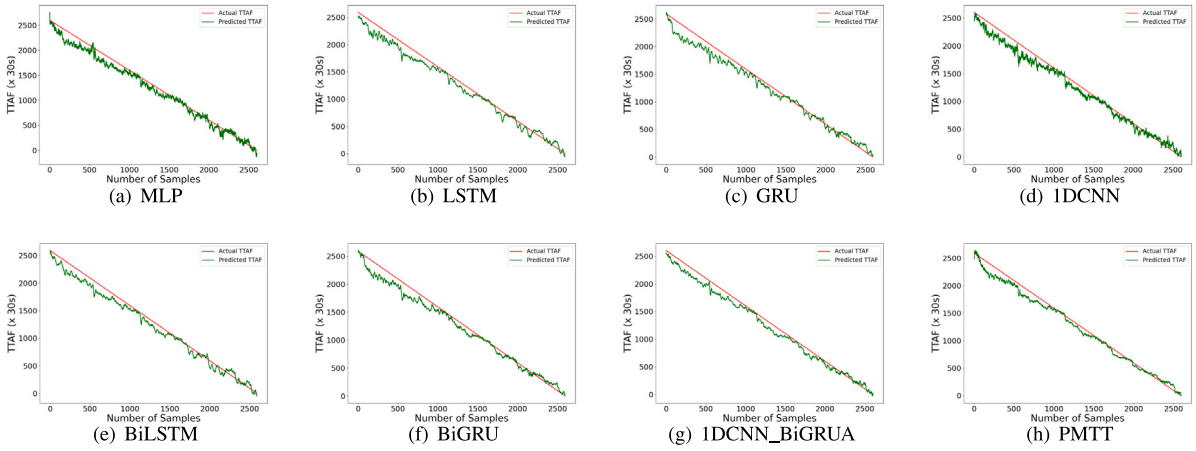


Fig. 6. The fitting results in Task_4 for the proposed PMTT and comparative models.

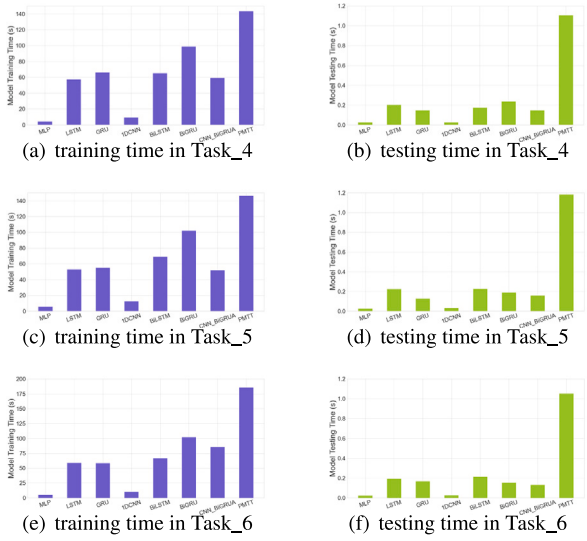


Fig. 7. The comparisons of the training time and testing time for all models (time is measured in seconds.).

Table 7
Trainable parameter size of each model on OpenStack.

Models	Parameter size
MLP	1345
LSTM	5537
GRU	4161
1DCNN	1969
BiLSTM	38 529
BiGRU	28 929
1DCNN-BiGRU-Attention	21 359
PMTT	50 465

models, but is able to achieve better prediction performance. However, during the evaluation phase of the testing set, the online test time of all models was very small. Specifically, the average test time of PMTT on three prediction tasks is 1.11 s, which has a small gap relative to baseline models. Therefore, a tradeoff between prediction performance and time cost is comparable and acceptable when PMTT is deployed in a real-world scenario.

The above fine-grained comparison reveals the effectiveness of our adopted "divide and conquer" strategy. Also, it shows that achieving effective information extraction for the constructed model is the key to achieving superior performance.

4.4. Ablation study of PMTT (RQ2)

In this section, the importance of each module is verified by performing ablation experiments. Specifically, we selectively retain a module in the overall architecture or simplify the model architecture to form the new PMTT variants. Afterward, we use these variants to perform experiments on six prediction tasks and observe the performance variations of each variant. In this subsection, four variants of PMTT and one baseline model (i.e., 1DCNN-BiGRU-Attention) are used to compare with PMTT. The detailed information of PMTT variants is as follows:

(1) PMTT_1: This variant only adopts the LFE module to extract the software degradation feature used for TTAF prediction, which retains the top half of Fig. 1(a).

(2) PMTT_2: Similarly, this variant only adopts the GFE module to extract the software degradation feature for performing TTAF prediction, which retains the bottom half of Fig. 1(a).

(3) PMTT_3: This variant only adopts single-scale TCNs (i.e., one channel with fixed kernel size) and GFE module, but it reserves the parallel architecture.

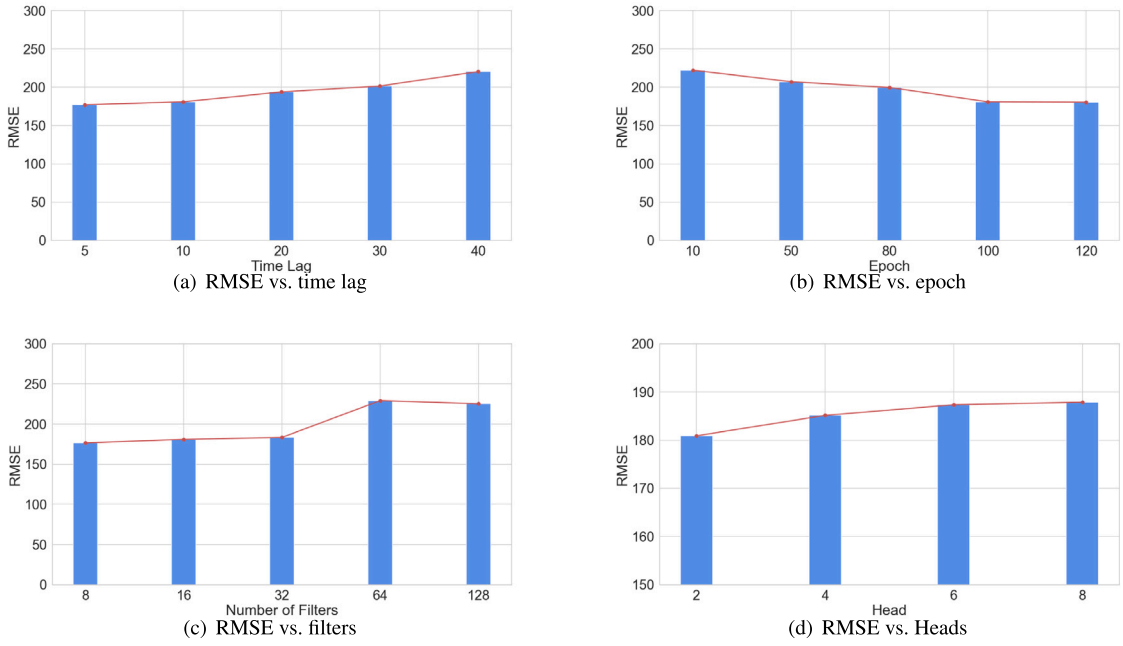
(4) PMTT_4: To verify the rationality of the parallel architecture, this variant modifies the parallel architecture of PMTT to the serial connection pattern of LFE and GFE modules.

Table 8 includes the prediction results for each variant, and we use bold to highlight if the variants bring performance improvements on six prediction tasks. The following observations can be obtained:

• **PMTT can integrate the advantages of each module well.** From the comparisons in Table 8, we can observe that, PMTT_1 achieves improvements in (3/18) cases, PMTT_2 can make improvements in (0/18) cases, and PMTT_3 can bring improvements in (6/18) cases while PMTT_4 can obtain improvement in (0/18) cases in terms of MAE, RMSE, and R^2, in six prediction tasks. However, the proposed PMTT makes performance improvements in (9/18) cases. The above observations verify that both local context and global information are essential for promoting prediction performance to some degree.

• **The variants fail to achieve consistent performance on different prediction tasks.** For example, PMTT_1 achieves the best performance on Task_5 for three metrics (52.06/64.52/0.9939); PMTT_3 achieves the best prediction performance on Task_2 for MAE metric (144.8), Task_3 for RMSE (139.6) and R^2 (0.9267), and Task_6 (55.41/67.66/0.9915); while the performance on the other tasks is unsatisfactory. Furthermore, neither of the remaining two variants achieves compelling predictions in comparison with the above two variants. Therefore, the observation demonstrates that the prediction performance of different variants fluctuates greatly, making it impossible to perform reliable predictions.

• **The contribution of the GFE module is slightly better than the LFE module.** Among the six prediction tasks, PMTT_1 achieves



(a) RMSE vs. time lag (b) RMSE vs. epoch (c) RMSE vs. filters (d) RMSE vs. Heads

Fig. 8. The parameter sensitivity analysis for proposed PMTT.

Table 8
Prediction results of different variants in six prediction tasks.

Methods	Metrics	1DCNN-BiGRU-Attention	PMTT_1	PMTT_2	PMTT_3	PMTT_4	PMTT
Task_1	MAE	237.1	213.2	202.3	183.3	279.8	**180.7**
	RMSE	273.1	247.3	235.9	215.3	311.3	**210.1**
	R^2	0.6058	0.6680	0.7106	0.7517	0.5172	**0.7666**
Task_2	MAE	187.5	255.3	211.0	**144.8**	184.1	148.6
	RMSE	234.8	288.5	249.1	185.0	216.2	**184.6**
	R^2	0.745	0.6077	0.6963	0.8311	0.7807	**0.8344**
Task_3	MAE	111.3	143.0	125.8	111.5	126.3	**107.9**
	RMSE	142.4	179.3	163.5	**139.6**	158.0	148.0
	R^2	0.9214	0.8776	0.8942	**0.9267**	0.8945	0.9115
Task_4	MAE	83.01	85.13	90.96	78.33	84.16	**76.62**
	RMSE	98.93	101.8	108.0	94.22	98.95	**92.09**
	R^2	0.9826	0.9814	0.9789	0.9838	0.9822	**0.9846**
Task_5	MAE	57.47	**52.06**	62.10	55.11	57.88	53.80
	RMSE	77.03	**64.52**	78.60	71.07	75.98	69.13
	R^2	0.9915	**0.9939**	0.9910	0.9927	0.9916	0.9930
Task_6	MAE	64.99	71.39	70.10	**55.41**	61.39	57.34
	RMSE	82.14	85.31	84.31	**67.66**	75.07	70.31
	R^2	0.9875	0.9863	0.9861	**0.9915**	0.9894	0.9907
Count		0	3	0	6	0	9

better prediction performance on tasks 4 and 5, while PMTT_2 achieves better performance on the remaining tasks. Compared with PMTT_1, PMTT_2 achieves average performance improvements of 1.7%, 0.05%, and 3.7%, respectively, in terms of three metrics. This result indicates the importance of modeling global features, especially for prediction tasks on the Android system. Another reason may be that local features often contain noise, and the noise information affects model learning, leading to a severe decline in model performance.

• **The proposed parallel architecture is reasonable compared to the serial architecture in our prediction tasks.** It is not difficult to find that no matter which prediction task, PMTT_4 shows weaker performance compared with PMTT. In addition, compared with PMTT_3, PMTT_4 still does not have any performance advantage. What is worse, on some prediction tasks (e.g., Task_1), its performance (279.8/311.3/0.5172) is even worse than GRU and 1DCNN-BiGRU-Attention. The above results show that a deeper network does not mean better prediction performance, and may even worsen the performance, since layer-by-layer feature extraction increases the network complexity and easily causes the loss of useful information in our prediction tasks. Moreover, the prediction results of the proposed PMTT and PMTT_3 indicate that the design of the parallel structure is reasonable in comparison with other variants.

In conclusion, the numerical results of PMTT shed light on the fact that it is necessary to model global–local degradation information synchronously in a feasible way. Multi-scale and parallel structural design can bring more effective prediction, since the global and local features are conjointly extracted to construct intact knowledge by using LFE and GFE modules.

4.5. Sensitivity analysis for parameter and fusion method (RQ3)

In this section, we expound on how parameter combinations and feature fusion methods affect model performance.

First, to investigate the influence of various parameter choices on model performance, the sensitivity analysis on Task_1, Task_2, and Task_3 is performed, and we adjust each parameter in a reasonable scope in turn. These parameters include time lag, epoch, filter number, and Head number, as they have a marked influence on model performance. The average results on three prediction tasks are depicted in Fig. 8. Note that, we only report the RMSE results, since the remaining metrics show analogical variation trends and page limits. From Fig. 8, we can obtain the following observations:

• **More historical information as input, i.e., increasing the time lag, may not necessarily improve performance in our prediction task.** Intuitively, the more historical data the model inputs, the more effective software degradation characteristics it can learn. However, the optimal lag is not that bigger is better; on the contrary, time lags of 5 and 10 can achieve lower prediction errors. This result is mainly because longer historical data will contain more noise, which will weaken model performance.

• **More epochs, filters, and Heads, also do not mean superior performance.** For the parameter epoch, as the epochs increase, the number of weight updates for the trained network model is also incremental. The model performance will be saturated after certain training times, and continuing to increase epochs may not necessarily improve model performance. In the case of the parameter filters, the parallel

Table 9
Prediction results of PMTT with different feature fusion methods.

Methods	Metrics	Task_1	Task_2	Task_3	Task_4	Task_5	Task_6
PMTTb (Concat)	MAE	244.7	197.1	120.9	87.45	**50.62**	58.32
	RMSE	277.5	237.5	159.9	103.8	**66.09**	71.94
	R^2	0.6149	0.7343	0.9034	0.9806	**0.9936**	0.9903
PMTT (Add)	MAE	**180.7**	**148.6**	**107.9**	**76.62**	53.80	**57.34**
	RMSE	**210.1**	**184.6**	**148.0**	**92.09**	69.13	**70.31**
	R^2	**0.7666**	**0.8344**	**0.9115**	**0.9846**	0.9930	**0.9907**

structure allows for the effective extraction of local features by using fewer filters with multiple scales for each channel, and more filters may cause overfitting problems. For the parameter Head, the number of Heads slightly impacts the model performance. In our prediction tasks, Head = 2 is a better choice.

Second, to assess the impact of feature fusion methods on model performance, we specifically examine two feature-level fusion methods: element-wise addition fusion and concatenation fusion. The numerical results of these evaluations are presented in Table 9. Based on the observations from Table 9, we can make the following findings. After employing the concatenation fusion method, the model performance on Task_5 has significant improvements, compared with the addition fusion. However, on the remaining tasks, the prediction ability of PMTT(Concat) is surpassed by addition fusion. This result may be due to the fact that the concatenation operation directly concatenates the features of different channels, which leads to feature dimension expansion and increases the difficulty of model training. In contrast, since addition fusion does not change the feature dimension through element-wise addition, it more effectively integrates the feature-level information for different channels.

4.6. Importance analysis of aging indicators (RQ4)

In this section, we try to experimentally verify which of these frequently reported aging indicators are relatively significant in TTAF prediction on the basics of the previous experiments. The overall goal of this section is to explore whether the impact of each aging indicator on the prediction performance of PMTT is positive or negative, thereby identifying important aging indicators. Specifically, we perform experiments by removing one indicator at a time and inputting the remaining aging indicators as features into the proposed PMTT for training and prediction. If, after removing this indicator, the performance of PMTT is improved, we claim that this indicator does not contribute to the prediction and is redundant. On the contrary, if after removing a certain indicator, the model performance becomes worse, we believe that this indicator is benefit to prediction and is important for our prediction tasks. We perform experiments on two case software systems and have the following findings.

For the Android system, we perform importance analysis on the 13 indicators used in the experiment, as described in Table 2. We adopt the strategy of removing one specific aging indicator and retaining the remaining 12 indicators as model inputs, and perform such repeated operations 13 times in sequence to verify the impact of each indicator on the prediction performance of PMTT. The experimental results are included in Table 10. From the results in Table 10, it can be found that different indicators do have an impact on the prediction performance of PMTT, including positive and negative effects. For example, after removing *CPU* indicator, the performance of PMTT is improved, obtaining a lower prediction error; but after removing the aging indicators, e.g., *Naïve Heap*, *Dalvik Heap*, and *Load5*, the prediction performance of PMTT tends to deteriorate, showing a higher prediction error. Meanwhile, it is not difficult to find that the impact of most aging indicators on model performance is inconsistent in different tasks. For instance, for the *PSS of system_server* indicator, removing this indicator improves the prediction performance of PMTT on Tasks_1 and Task_3, but on Task_2, the performance of PMTT decreases, indicating an increase in prediction error. Similar situations are also observed in other indicators, such as *Free memory*, *Swap*, and *Used memory*. But overall, it is useful to perform the importance analysis of aging indicators, which can guide the optimization of model architecture in subsequent work. In addition, it can be clearly observed from Fig. 9 that the importance of each indicator varies for different prediction tasks. Note that, we only report the RMSE results, since the remaining metrics show analogical variation trends and page limits. Concretely, on the prediction Task_1, *Temperature*, *Dalvik Heap*, *pgfault*, *Load1*, and *Load5* are important indicators relative to the rest, and the absence of such indicators leads to the deterioration of model performance; for Task_2, *Naïve Heap*, *Load5*, *Temperature*, *Load1*, and *PSS of system_server* indicators are important, while *Temperature*, *Used memory*, *Naïve Heap*, *pgfree*, and *Load15*, are important for Task_3. By further combining the above observations, in future TTAF prediction tasks, the top 7 aging indicators are recommended, including *Temperature*, *Naïve Heap*, *Dalvik Heap*, *Load1*, *Load5*, *pgfree*, and *pgfault*, as the preferred input features of the model.

Table 10
The importance analysis of aging indicators on Android system by removing one indicator and retaining the remaining indicators.

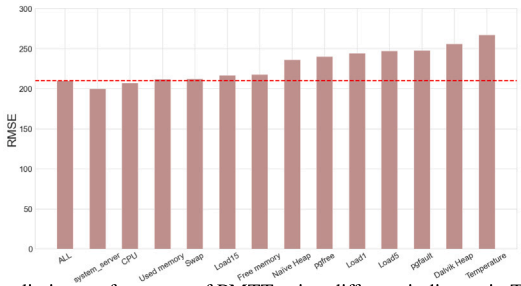
Selected indicator	Metrics	Task_1	Task_2	Task_3
All	MAE	180.7	148.6	107.9
	RMSE	210.1	184.6	148.0
	R^2	0.7666	0.8344	0.9115
Load1 (removed)	MAE	213.7	160.1	123.8
	RMSE	244.4	195.4	153.9
	R^2	0.6973	0.8188	0.9067
Load5 (removed)	MAE	218.2	156.4	115.8
	RMSE	247.5	198.0	**142.3**
	R^2	0.6822	0.8097	**0.9230**
Load15 (removed)	MAE	185.5	149.8	124.8
	RMSE	216.9	185.3	157.1
	R^2	0.7605	**0.8354**	0.9050
CPU (removed)	MAE	**179.9**	152.6	113.8
	RMSE	**206.8**	190.6	**140.2**
	R^2	**0.7821**	0.8293	**0.9250**
Temperature (removed)	MAE	225.3	157.7	168.4
	RMSE	267.3	196.7	209.5
	R^2	0.6207	0.8145	0.8340
PSS of system_server (removed)	MAE	**172.7**	151.0	**97.86**
	RMSE	**199.8**	192.7	**128.8**
	R^2	**0.7954**	0.8202	**0.9373**
Free memory (removed)	MAE	190.1	**143.7**	117.5
	RMSE	217.7	189.6	**147.2**
	R^2	0.7517	0.8319	**0.9163**
Swap (removed)	MAE	192.6	**120.5**	**102.2**
	RMSE	212.4	**158.9**	**130.8**
	R^2	**0.7760**	**0.8809**	**0.9325**
Used memory (removed)	MAE	182.1	**135.8**	136.6
	RMSE	211.7	**178.6**	170.8
	R^2	0.7425	**0.8436**	0.8858
Naïve Heap (removed)	MAE	205.3	184.0	138.6
	RMSE	236.2	228.8	169.7
	R^2	0.7039	0.7523	0.8746
Dalvik Heap (removed)	MAE	224.9	**138.0**	113.6
	RMSE	256.2	**177.0**	150.1
	R^2	0.6705	**0.8415**	**0.9126**
pgfault (removed)	MAE	214.1	156.0	113.1
	RMSE	247.9	185.3	**147.2**
	R^2	0.6815	**0.8375**	**0.9158**
pgfree (removed)	MAE	211.1	**133.0**	128.4
	RMSE	240.0	**175.5**	161.8
	R^2	0.7098	**0.8502**	0.9006

Results are bold, indicating that removing the indicator improves model performance, while results are underlined if removing the indicator worsens model performance.

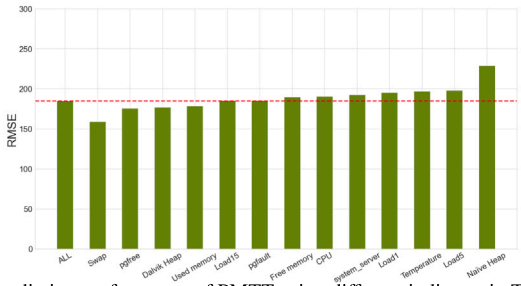
Table 11
The importance analysis of aging indicators on OpenStack platform by removing one indicator and retaining the remaining indicators.

Selected indicator	Metrics	Task_4	Task_5	Task_6
All	MAE	76.62	53.80	57.34
	RMSE	92.09	69.13	70.31
	R^2	0.9846	0.9930	0.9907
Load1 (removed)	MAE	**74.23**	**50.30**	59.57
	RMSE	**90.73**	**62.58**	74.49
	R^2	**0.9852**	**0.9943**	0.9896
Load5 (removed)	MAE	77.57	**51.06**	63.92
	RMSE	94.53	**66.38**	77.52
	R^2	0.9837	**0.9936**	0.9887
Load15 (removed)	MAE	92.43	54.88	59.92
	RMSE	110.5	71.42	73.05
	R^2	0.9778	0.9925	0.9900
CPU (removed)	MAE	83.67	**51.64**	**55.19**
	RMSE	99.28	**65.65**	**68.12**
	R^2	0.9820	**0.9937**	**0.9912**
Free memory (removed)	MAE	78.22	**52.41**	**52.49**
	RMSE	93.89	**67.22**	**64.65**
	R^2	0.9838	**0.9935**	**0.9922**
inact (removed)	MAE	79.03	67.95	**53.86**
	RMSE	94.56	82.83	**67.74**
	R^2	0.9839	0.9900	**0.9913**
active (removed)	MAE	83.85	62.70	57.44
	RMSE	99.87	80.02	74.16
	R^2	0.9818	0.9906	0.9897
Swap (removed)	MAE	81.82	65.39	76.28
	RMSE	101.0	80.27	93.58
	R^2	0.9813	0.9907	0.9836
r/s (removed)	MAE	77.09	60.83	**54.98**
	RMSE	**92.08**	77.02	**67.84**
	R^2	**0.9846**	0.9913	**0.9914**
w/s (removed)	MAE	85.35	55.67	61.49
	RMSE	103.8	72.29	74.81
	R^2	0.9805	0.9923	0.9891

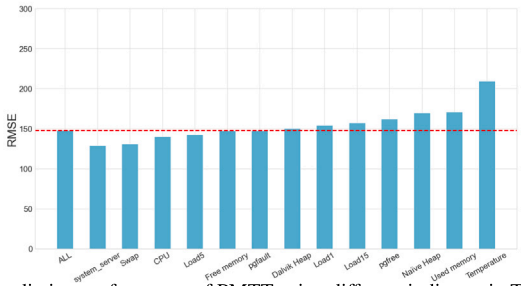
Results are bold, indicating that removing the indicator improves model performance, while Results are underlined if removing the indicator worsens model performance.



(a) Prediction performance of PMTT using different indicator in Task_1



(b) Prediction performance of PMTT using different indicator in Task_2



(c) Prediction performance of PMTT using different indicator in Task_3

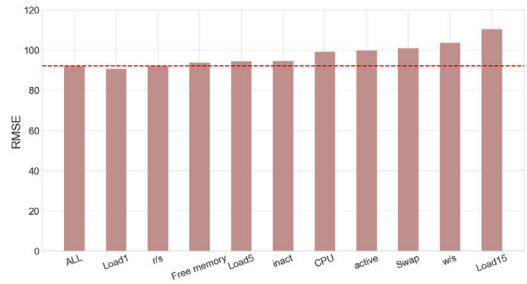
Fig. 9. The prediction performance analysis of PMTT by selecting different aging indicators on the Android system.

For the OpenStack platform, we conduct importance analysis on the 10 indicators used for TTAF prediction, as shown in Table 2. The implementation procedure is identical to that of the Android system and will not be repeated here. The numerical results are depicted in Table 11. From the results in the table, we can observe that different indicators do have a positive or negative impact on the prediction performance of PMTT. For example, after removing the aging indicator, i.e., *Free memory*, the performance of PMTT is improved; however, after removing the aging indicators, e.g., *Swap*, *w/s*, and *active*, the prediction performance of PMTT tends to deteriorate. Also, it is not difficult to find that most aging indicators have inconsistent effects on model performance in different tasks. For example, for the *CPU* indicator, on Task_5 and Task_6, removing this indicator improves the prediction performance of PMTT, but on Task_4, the model performance decreases, showing an increase in the prediction error. For the *Load1* indicator, in Task_4 and Task_5, removing this indicator promotes the prediction performance of PMTT, but in Task_6, the model performance gets worse. Similar phenomena are also observed in other indicators, such as the *r/s* and *inact*. In addition, it can be observed from Fig. 10 that, the importance of each indicator also varies in different prediction tasks. Specially, in Task_4, *Load15*, *w/s*, *Swap*, *active*, and *CPU* indicators are important relative to the rest of the indicators, since the absence of these indicators leads to a deterioration in model performance; for Task_5, *inact*, *Swap*, *active*, *r/s*, and *w/s* are important, while for Task_6, *Swap*, *Load5*, *w/s*, *Load1*, and *active* are significant. By further analyzing the above observations, the top 5 aging indicators are recommended, including *Swap*, *inact*, *active*, *Load15*, and *w/s*, as the preferred input features of the model.

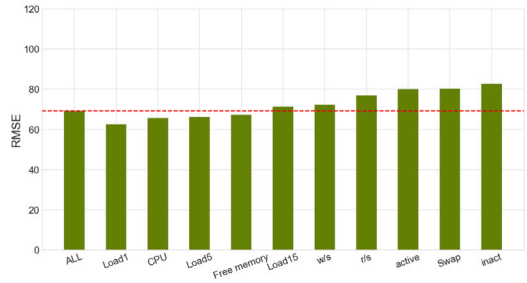
5. Threats to validity

One of the threats to the experiment is the datasets obtained using the accelerated life test (ALT). In our manuscript, we use run-to-failure data obtained through ALT to train PMTT on dataset A, and then predict on dataset B. We believe that it is also feasible for PMTT to use the run-to-failure dataset C under the standard setup scenario to predict the run-to-failure dataset D under the standard setup scenario (if such data can be collected easily), because the running conditions of the software systems for collecting datasets C and D are similar. Therefore, as long as the degradation information implied by the aging indicators can be fully extracted, theoretically, the TTAF of any running state (with or without ALT) of the software system can be inferred by PMTT. It should be mentioned that in engineering fields, such as bearings (Yang et al., 2023), gears (Qin et al., 2023a), and lithium batteries (Bracale et al., 2023), similar ALT schemes are also used to obtain failure data of these products (Yang et al., 2023; Qin et al., 2023a; Bracale et al., 2023). Then, the failure data obtained by ALT is used to train AI-based models and predict the remaining useful life of these products, which also illustrates the rationality of our method to a certain extent.

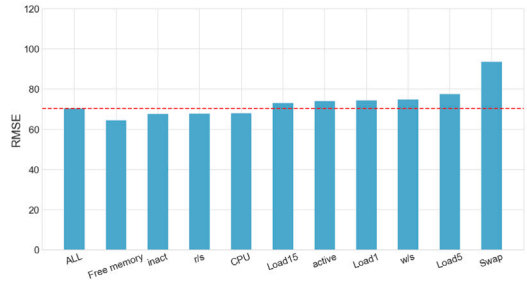
Another potential threat is the number of datasets. We did not collect more data at that time for experimental evaluation, since the data collection process was quite time-consuming, even using the ALT scheme. This is mainly because stress tests for Android and OpenStack



(a) Prediction performance of PMTT using different indicator in Task_4



(b) Prediction performance of PMTT using different indicator in Task_5



(c) Prediction performance of PMTT using different indicator in Task_6

Fig. 10. The prediction performance analysis of PMTT by selecting different aging indicators on the OpenStack platform.

may abort and fail to bring the system to aging failure conditions. Every implementation of ALT on a specific software system does not necessarily mean that a useful dataset can be collected. In this process, many unavoidable non-human errors occurred. For example, the Android system shuts down due to battery exhaustion; in the early stage of stress testing, the system hangs due to unknown causes, and log information cannot be exported; the temperature of the Android system suddenly rises, causing the system to shut down; problems such as network outages caused creation and deletion requests of VM to fail; abnormal exit of the log collection script, resulting in no valid data being collected; and many others. However, it is necessary to collect more data on different versions, different VM hardware configurations, and non-fixed and varied workloads, which can further verify the generalization of the model. Therefore, future research may continue to collect run-to-failure data from different systems to make the collected data more diverse.

Finally, the adaptability problem of PMTT in anomaly detection tasks. In our current work, we did not discuss the anomaly detection problem, since TTAF prediction is a regression prediction task, while anomaly detection is essentially a classification prediction task, which is also a potential threat. In future work, we can try to apply PMTT to the anomaly detection tasks by slightly modifying the input and output parts of the model, in which software aging state detection is an applicable direction for PMTT. Furthermore, by combining the results of anomaly detection and TTAF prediction, more comprehensive and effective auxiliary information can be provided for the selection of rejuvenation scheduling.

6. Conclusion and future work

In this work, we introduce a new framework with PMTT that combines the multi-scale TCNs and transformer to model global and local features hidden in the aging process for mainstream Android and Cloud systems. We implement extensive experiments on two case software systems with six prediction tasks to verify the performance of PMTT. Benefiting from the abundant feature extraction capabilities of PMTT, the evaluation results clearly indicate that our proposed PMTT outperforms multiple competitive models in terms of both prediction accuracy and stability.

Future research may consider fusing multi-source data to enable the model to learn more diverse degradation characteristics. Additionally, we also plan to adopt some feature selection techniques to facilitate more effective predictions.

CRediT authorship contribution statement

Kai Jia: Writing – original draft, Visualization, Methodology, Formal analysis, Conceptualization. **Xiao Yu:** Writing – review & editing. **Chen Zhang:** Writing – review & editing. **Wenzhi Xie:** Data curation. **Dongdong Zhao:** Writing – review & editing. **Jianwen Xiang:** Supervision, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is partially supported by the National Key Research and Development Program (Grant No. 2022YFB3104001), the National Natural Science Foundation of China (Grant No.62202350), the Key Research and Development Program of Hubei Province, China (Grant No. 2022BAA050, 2023BAB016), and the Natural Science Foundation of Chongqing, China (Grant No. cstc2021jcyj-msxmX1146, cstc2021jcyj-msxmX1115).

References

Andrade, E., Pietrantuono, R., Machida, F., Cotroneo, D., 2023. A comparative analysis of software aging in image classifiers on cloud and edge. IEEE Trans. Dependable Secure Comput. 20 (1), 563–573.

Araujo, J., Oliveira, F., Matos, R., Torquato, M., Ferreira, J., Maciel, P., 2016. Software aging issues in streaming video player. J. Softw. 11, 554–568.

Bai, J., Chang, X., Machida, F., Jiang, L., Han, Z., Trivedi, K.S., 2023. Impact of service function aging on the dependability for MEC service function chain. IEEE Trans. Dependable Secure Comput. 20 (4), 2811–2824.

Bai, J., Chang, X., Trivedi, K.S., Han, Z., 2021. Resilience-driven quantitative analysis of vehicle platooning service. IEEE Trans. Veh. Technol. 70 (6), 5378–5389.

Bai, S., Zico Kolter, J., Koltun, V., 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271.

Bracale, A., De Falco, P., Noia, L.P.D., Rizzo, R., 2023. Probabilistic state of health and remaining useful life prediction for li-ion batteries. IEEE Trans. Ind. Appl. 59 (1), 578–590.

Chen, Y., Nie, Y., Yin, B., Zheng, Z., Wu, H., 2023. An empirical study to identify software aging indicators for android OS. In: 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security. QRS, pp. 428–439.

Chung, J., Gulcehre, C., Cho, K., Bengio, Y., 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv:1412.3555.

Cotroneo, D., De Simone, L., Liguori, P., Natella, R., Bidokhti, N., 2019a. How bad can a bug get? An empirical analysis of software failures in the OpenStack cloud computing platform. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. FSE/ESEC, pp. 200–211.

Cotroneo, D., De Simone, L., Natella, R., Pietrantuono, R., Russo, S., 2019b. A configurable software aging detection and rejuvenation agent for android. In: 2019 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, pp. 239–245.

Cotroneo, D., De Simone, L., Natella, R., Pietrantuono, R., Russo, S., 2022. Software micro-rejuvenation for Android mobile systems. J. Syst. Softw. 186, 111181.

Cotroneo, D., Iannillo, A.K., Natella, R., Pietrantuono, R., 2020. A comprehensive study on software aging across android versions and vendors. Empir. Softw. Eng. 25, 3357–3395.

Cotroneo, D., Natella, R., Pietrantuono, R., 2013. Predicting aging-related bugs using software complexity metrics. Perform. Eval. 70 (3), 163–178.

Cotroneo, D., Natella, R., Pietrantuono, R., Russo, S., 2010. Software aging analysis of the linux operating system. In: 2010 IEEE 21st International Symposium on Software Reliability Engineering. ISSRE, pp. 71–80.

Cotroneo, D., Natella, R., Pietrantuono, R., Russo, S., 2014. A survey of software aging and rejuvenation studies. J. Emerg. Technol. Comput. Syst. 10 (1), 1–34.

Cotroneo, D., Orlando, S., Russo, S., 2007. Characterizing aging phenomena of the java virtual machine. In: 2007 26th IEEE International Symposium on Reliable Distributed Systems. SRDS, pp. 127–136.

Deng, F., Bi, Y., Liu, Y., Yang, S., 2022. Remaining useful life prediction of machinery: A new multiscale temporal convolutional network framework. IEEE Trans. Instrum. Meas. 71, 1–13.

Dohi, T., Zheng, J., Okamura, H., Trivedi, K.S., 2018. Optimal periodic software rejuvenation policies based on interval reliability criteria. Reliab. Eng. Syst. Saf. 180, 463–475.

Du, X., Xiao, G., Sui, Y., 2020. Fault triggers in the TensorFlow framework: An experience report. In: 2020 IEEE 31st International Symposium on Software Reliability Engineering. ISSRE, pp. 1–12.

Espinosa, R., Palma, J., Jiménez, F., Kamińska, J., Sciavicco, G., Lucena-Sánchez, E., 2021. A time series forecasting based multi-criteria methodology for air quality prediction. Appl. Soft Comput. 113, 107850.

Fargalla, M.A.M., Yan, W., Deng, J., Wu, T., Kiyingi, W., Li, G., Zhang, W., 2024. TimeNet: Time2Vec attention-based CNN-BiGRU neural network for predicting production in shale and sandstone gas reservoirs. Energy 290, 130184.

Fu, S., Lin, L., Wang, Y., Guo, F., Zhao, M., Zhong, B., Zhong, S., 2024. MCA-DTCN: A novel dual-task temporal convolutional network with multi-channel attention for first prediction time detection and remaining useful life prediction. Reliab. Eng. Syst. Saf. 241, 109696.

Grottke, M., Li, L., Vaidyanathan, K., Trivedi, K., 2006. Analysis of software aging in a web server. IEEE Trans. Reliab. 55 (3), 411–420.

Huang, Y., Kintala, C., Kolettis, N., Fulton, N., 1995. Software rejuvenation: analysis, module and applications. In: Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers. pp. 381–390.

Jia, K., Yu, X., Zhang, C., Hu, W., Zhao, D., Xiang, J., 2022. The impact of software aging and rejuvenation on the user experience for android system. In: 2022 IEEE 33rd International Symposium on Software Reliability Engineering. ISSRE, pp. 435–445.

Jia, K., Yu, X., Zhang, C., Hu, W., Zhao, D., Xiang, J., 2023. Software aging prediction for cloud services using a gate recurrent unit neural network model based on time series decomposition. IEEE Trans. Emerg. Top. Comput. 11 (3), 580–593.

Jia, Y.-F., Zhao, L., Cai, K.-Y., 2008. A nonlinear approach to modeling of software aging in a web server. In: 2008 15th Asia-Pacific Software Engineering Conference. pp. 77–84.

Levitin, G., Xing, L., Xiang, Y., 2020. Cost minimization of real-time mission for software systems with rejuvenation. Reliab. Eng. Syst. Saf. 193, 106593.

Li, L., Vaidyanathan, K., Trivedi, K., 2002. An approach for estimation of software aging in a Web server. In: Proceedings International Symposium on Empirical Software Engineering. pp. 91–100.

Liu, J., Tan, X., Wang, Y., 2019. CSSAP: Software aging prediction for cloud services based on ARIMA-LSTM hybrid model. In: 2019 IEEE International Conference on Web Services. ICWS, pp. 283–290.

Machida, F., Kim, D.S., Trivedi, K.S., 2013. Modeling and analysis of software rejuvenation in a server virtualized system with live VM migration. Perform. Eval. 70 (3), 212–230.

Machida, F., Nicola, V.F., Trivedi, K.S., 2014. Job completion time on a virtualized server with software rejuvenation. ACM J. Emerg. Technol. Comput. Syst. 10 (1), 1–26.

Marshall, E., 1992. Fatal error: How patriot overlooked a scud. Science 255 (5050), 1347.

Mazzi, Y., Ben Sassi, H., Errahimi, F., 2024. Lithium-ion battery state of health estimation using a hybrid model based on a convolutional neural network and bidirectional gated recurrent unit. Eng. Appl. Artif. Intell. 127, 107199.

Meng, H., Tong, X., Shi, Y., Zhu, L., Feng, K., Hei, X., 2021. Cloud server aging prediction method based on hybrid model of autoregressive integrated moving average and recurrent neural network. J. Commun. 42 (01), 163–171.

Meng, H., Zhang, J., 2024. A novel multi-step-ahead approach for cloud server aging prediction based on hybrid deep learning model. Eng. Appl. Artif. Intell. 133, 108588.

Nie, Y., Chen, Y., Jiang, Y., Wu, H., Yin, B., Cai, K.-Y., 2024. A method of multidimensional software aging prediction based on ensemble learning: A case of Android OS. Inf. Softw. Technol. 170, 107422.

Peng, H., Jiang, B., Mao, Z., Liu, S., 2023. Local enhancing transformer with temporal convolutional attention mechanism for bearings remaining useful life prediction. IEEE Trans. Instrum. Meas. 72, 1–12.

Pereira, P., Araujo, J., Matos, R., Preguiça, N., Maciel, P., 2018. Software rejuvenation in computer systems: An automatic forecasting approach based on time series. In: 2018 IEEE 37th International Performance Computing and Communications Conference. IPCCC, pp. 1–8.

Pietrantuono, R., Russo, S., 2020. A survey on software aging and rejuvenation in the cloud. Softw. Qual. J. 28, 7–38.

Qiao, Y., Zheng, Z., Fang, Y., 2018. An empirical study on software aging indicators prediction in android mobile. In: 2018 IEEE International Symposium on Software Reliability Engineering Workshops. ISSREW, pp. 271–277.

Qiao, Y., Zheng, Z., Fang, Y., Qin, F., Trivedi, K.S., Cai, K.-Y., 2019. Two-level rejuvenation for android smartphones and its optimization. IEEE Trans. Reliab. 68 (2), 633–652.

Qin, Y., Yang, J., Zhou, J., Pu, H., Zhang, X., Mao, Y., 2023a. Dynamic weighted federated remaining useful life prediction approach for rotating machinery. Mech. Syst. Signal Process. 202, 110688.

Qin, F., Zheng, Z., Wan, X., Liu, Z., Shi, Z., 2023b. Predicting aging-related bugs using network analysis on aging-related dependency networks. IEEE Trans. Emerg. Top. Comput. 11 (3), 566–579.

Sheng, Z., Xu, Y., Xue, S., Li, D., 2022. Graph-based spatial-temporal convolutional network for vehicle trajectory prediction in autonomous driving. IEEE Trans. Intell. Transp. Syst. 23 (10), 17654–17665.

Tan, X., Liu, J., 2021. ACLM:Software aging prediction of virtual machine monitor based on attention mechanism of CNN-LSTM model. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security. QRS, pp. 759–767.

Ullah, F., Bilal, M., Yoon, S.-K., 2023. Intelligent time-series forecasting framework for non-linear dynamic workload and resource prediction in cloud. Comput. Netw. 225, 109653.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems. Vol. 30, Curran Associates, Inc..

Vinícius, L., Rodrigues, L., Torquato, M., Silva, F.A., 2022. Docker platform aging: a systematic performance evaluation and prediction of resource consumption. J. Supercomput. 78, 12898–12928.

Wang, T., Fu, L., Zhou, Y., Gao, S., 2022. Service price forecasting of urban charging infrastructure by using deep stacked CNN-BiGRU network. Eng. Appl. Artif. Intell. 116, 105445.

Wang, D., Xie, W., Trivedi, K.S., 2007. Performability analysis of clustered systems with rejuvenation under varying workload. Perform. Eval. 64 (3), 247–265.

Wang, Z., Yao, J., Xu, M., Jiang, M., Su, J., 2024. Transformer-based network with temporal depthwise convolutions for sEMG recognition. Pattern Recognit. 145, 109967.

Xiang, J., Weng, C., Zhao, D., Andrzejak, A., Xiong, S., Li, L., Tian, J., 2020. Software aging and rejuvenation in android: new models and metrics. Softw. Qual. J. 28 (1), 85–106.

Yan, Y., 2020. Software ageing prediction using neural network with ridge. IET Softw. 14 (5), 517–524.

Yang, L., Liao, Y., Duan, R., Kang, T., Xue, J., 2023. A bidirectional recursive gated dual attention unit based RUL prediction approach. Eng. Appl. Artif. Intell. 120, 105885.

Yang, P., Zhu, L., Zhang, Y., Ma, C., Liu, L., Yu, X., Hu, W., 2024. On the relative value of clustering techniques for unsupervised effort-aware defect prediction. Expert Syst. Appl. 245, 123041.

Yao, Y., Yang, M., Wang, J., Xie, M., 2023. Multivariate time-series prediction in industrial processes via a deep hybrid network under data uncertainty. IEEE Trans. Ind. Inform. 19 (2), 1977–1987.

Yu, X., Dai, H., Li, L., Gu, X., Keung, J.W., Bennin, K.E., Li, F., Liu, J., 2023. Finding the best learning to rank algorithms for effort-aware defect prediction. Inf. Softw. Technol. 157, 107165.

Zhang, Z., Chen, Y., Zhang, D., Qian, Y., Wang, H., 2023a. CTFNet: Long-sequence time-series forecasting based on convolution and time–frequency analysis. IEEE Trans. Neural Netw. Learn. Syst. 1–15.

Zhang, Y., Feng, K., Ji, J.C., Yu, K., Ren, Z., Liu, Z., 2023b. Dynamic model-assisted bearing remaining useful life prediction using the cross-domain transformer network. IEEE/ASME Trans. Mechatronics 28 (2), 1070–1080.

Zhang, C., Feng, S., Xie, W., Zhao, D., Xiang, J., Pietrantuono, R., Natella, R., Cotroneo, D., 2023c. IFCM: An improved fuzzy C-means clustering method to handle class overlap on aging-related software bug prediction. In: 2023 IEEE 34th International Symposium on Software Reliability Engineering. ISSRE, pp. 590–600.

Zhang, J., Jiang, Y., Wu, S., Li, X., Luo, H., Yin, S., 2022. Prediction of remaining useful life based on bidirectional gated recurrent unit with temporal self-attention mechanism. Reliab. Eng. Syst. Saf. 221, 108297.

Zhang, Z., Wang, J., Wei, D., Xia, Y., 2023d. An improved temporal convolutional network with attention mechanism for photovoltaic generation forecasting. Eng. Appl. Artif. Intell. 123, 106273.
Zhang, Z., Yi, X., Zhao, X., 2021. Fake speech detection using residual network with transformer encoder. In: Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security. IH&MMSec '21, pp. 13–22.
Zhao, Q., Zhang, Y., Feng, X., 2022. Predicting information diffusion via deep temporal convolutional networks. Inf. Syst. 108, 102045.
Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W., 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting. Proc. AAAI Conf. Artif. Intell. 35 (12), 11106–11115.

Kai Jia received his M.S. in Henan Normal University in 2019. He is currently a Ph.D. candidate in the School of Computer and Artificial Intelligence, Wuhan University of Technology. His research is focused on software aging and rejuvenation (SAR) and aging-related bug prediction.

Xiao Yu received the Ph.D. degree from the Department of Computer Science, City University of Hong Kong in 2021. Currently, he is working as an assistant professor in the School of Computer and Artificial Intelligence, Wuhan University of Technology. His research interests include software engineering and deep learning.

Chen Zhang received her M.S. in Wuhan University of Technology in 2020. She is currently a Ph.D. candidate in the School of Computer and Artificial Intelligence, Wuhan University of Technology. Her research is focused on software reliability and defect prediction.

Wenzhi Xie received his B.S. in East China Jiaotong University in 2021. He is currently a M.S. candidate in the School of Computer and Artificial Intelligence, Wuhan University of Technology. His research is focused on software aging-related bug prediction.

Dongdong Zhao received Ph.D. degree from University of Science and Technology of China (USTC) in 2016. He is currently an associate professor of the School of Computer and Artificial Intelligence of Wuhan University of Technology. His research interests include dependable computing, information security, privacy protection and biometrics.

Jianwen Xiang received Ph.D. degrees from Wuhan University and from Japan Advanced Institute of Science and Technology (JAIST) in 2004 and 2005, respectively. He is currently a Professor of the School of Computer and Artificial Intelligence of Wuhan University of Technology, and he was a researcher at NEC Corporation from 2008 to 2014. His research interests include dependable computing, information security, and software engineering.