Received 20 June 2023, accepted 15 July 2023, date of publication 24 July 2023, date of current version 28 July 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3298305*

RESEARCH ARTICLE

# MTLFormer: Multi-Task Learning Guided Transformer Network for Business Process Prediction

**JIAOJIAO WANG[1,2], JIAWEI HUANG[3], XIAOYU MA[1,2], ZHONGJIN LI[4], YAQI WANG[3], AND DINGGUO YU[1,2]**
[1]Institute of Intelligent Media Technology, Communication University of Zhejiang, Hangzhou 310018, China
[2]Key Laboratory of Film and TV Media Technology of Zhejiang Province, Hangzhou 310018, China
[3]College of Media Engineering, Communication University of Zhejiang, Hangzhou 310018, China
[4]School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China

Corresponding author: Dingguo Yu (zjydg@163.com)

This work was supported in part by the National Natural Science Foundation, China, under Grant 62002316, Grant 61802095, and Grant 62206242; in part by the Key Science and Technology Project of Zhejiang Province under Grant 2021C03138; in part by the National Natural Science Foundation of Zhejiang Province, China, under Grant LY22F020021; and in part by the Public Welfare Technology Application Research Project of Zhejiang Province, China, under Grant LGF21F010001.

**ABSTRACT** The predictive business process monitoring mainly focuses on the performance prediction of business process execution, i.e., predicting the next activity, the execution time of the next activity, and the remaining time, respectively, for an ongoing process instance based on the knowledge gained from historical event logs. Although there is a specific relationship between these three tasks, recent research has focused on training separate prediction models for each task, resulting in high costs and time complexity. Additionally, existing technologies are limited in their ability to capture long-distance dependent features in process instances, further impeding prediction performance. To address these issues, this paper proposes the MTLFormer approach, which leverages the self-attention mechanism of the Transformer network and conducts multi-task parallel training through shared feature representation obtained from different tasks. Our approach reduces the time complexity of model training while simultaneously improving prediction performance. We extensively evaluate our approach on four real-life event logs, demonstrating its capability to achieve multi-task online real-time prediction and effectively improve prediction performance.

**INDEX TERMS** Multi-task learning, predictive business process monitoring, self-attention, Transformer.

## I. INTRODUCTION

Process mining, an indispensable and vital part of business process management, bridges data mining and business process management. As the core of enterprise operations gradually shifts to the application of the Process-aware Information System (PAIS), more and more process execution logs become available. Process mining technology extracts essential knowledge from these logs to provide effective means for process discovery, monitoring, and improvement [1]. With the development of data mining and profound learning techniques, Predictive (business) Process Monitoring (PPM) has become another hot research topic in process mining. Unlike traditional business process monitoring techniques, predictive business process monitoring aims to ensure the smooth execution of business processes by predicting the performance of executing process instances in real time [2]. Predictive business process monitoring techniques focus on the prediction of the next activity [3], the execution time of the next activity, the remaining execution time [4], and the outcome [2], [5] for an executing process instance. This technology enables users to predictably perceive the current status of business process execution and possible future situations in real-time. In this way, they can make decisions and responses

The associate editor coordinating the review of this manuscript and approving it for publication was Alex James.

in advance to improve the company's services' efficiency, reduce labor costs, and increase market competitiveness.

Most of the techniques currently used for PPM focus on traditional machine learning-based techniques and deep learning-based methods [2], [6]. No matter what you use, it requires supervised learning of historical process execution instances, i.e., predictive models are obtained based on historical process instances trained using some learning technique. With the increasing intelligence, most scholars are researching PPM techniques based on deep learning [6]. The research object of this technique is the event log recorded in PAIS, which is complex sequential data. To solve the feature learning problem of sequential data, Hochreiter and Schmidhuber [7] proposed a Long-short Term Memory (LSTM) network in the development of deep learning technology. LSTM is a variant of the Recurrent Neural Network (RNN) [8], and it still inherits the drawback of RNNs that rely too much on previous data states, which makes it slightly inadequate for training prediction models in the event log with long sequences, as demonstrated in [9]. In response to the limitations of LSTM, Vaswani et al. [10] proposed the Transformer sequence model based on the attention mechanism. The model can be applied to long-distance dependent feature learning and concurrent execution of multiple tasks in the data and can well solve the problems of LSTM in training models, thus improving the model's prediction performance. On this basis, Bukhsh et al. [11] proposed the ProcessTransformer model, i.e., modifying the Transformer structure according to the specific process prediction task, to achieve the desired prediction effect.

As a branch of machine Learning, Multi-task Learning (MTL) [12] can train multiple learning tasks simultaneously and make good use of the commonalities and differences between different tasks to improve the learning efficiency and prediction performance of the model. Inspired by them, we first introduce MTL in PPM by considering the strong correlation between some tasks. This parallel migration model can be used to share information between multiple tasks, i.e., it allows knowledge obtained from different tasks to migrate to each other. In this case, it has the advantage of reducing the time complexity of model training and improving prediction performance. Therefore, to further improve the prediction performance of each task in PPM, we propose an approach called MTLFormer (Multi-Task Learning Guided for Transformer Network) driven by multi-task learning in this paper. It utilizes the specific attention mechanism in the Transformer network to capture the long-distance dependent features in the data and share the feature representations obtained from different tasks. Meanwhile, position coding and residual networks are used in this approach to enhance the model's ability to perceive position information and solve the problems of gradient disappearance and weight matrix degradation, respectively. While providing higher performance for the business process prediction, this paper also provides a possible reference for combining multi-task learning and the Transformer network in other fields.

In summary, this article makes the following contributions:

- We innovatively propose an approach called MTLFormer by integrating the Transformer network with multi-task learning to improve the prediction performance and reduce the time complexity of model training in PPM tasks.
- We investigate the specific self-attention mechanism of the Transformer network in terms of the long-distance dependencies feature representation for the input historical process instances in terms of the prediction tasks in PPM.
- We focus on the three associated tasks, the next activity prediction, the next event time prediction, and the remaining time prediction, and then explore the effect of the hard-parameter sharing of multi-task learning for parallel multiple task training.

The rest of this paper is organized as follows. Firstly, we introduce the related research works and briefly discuss them in Section II. Then, in Section III, we define some basic concepts and research questions and introduce the multi-task learning model. Secondly, in Section IV, we introduce the proposed MTLFormer approach in detail. In Section V, we conduct extensive comparative experiments and evaluate the performance of MTLFormer. Finally, the paper is summarized and future research direction is briefly discussed in Section VI.

## II. RELATED WORK

Predictive (business) process monitoring belongs to business process execution/monitoring. Its research scope can be mainly divided into predicting future execution events in the process [3], predicting the execution time of future events and the remaining execution time of the process [4], and predicting the final execution result of the business process [2], [5] and so on. We performed an analysis and classification of these methods based on the specific method type, the prediction target, and the input data in Table 1. Especially, to demonstrate the motivation behind our work, we use the *separate* and *joint* to distinct whether the prediction model is trained on a single task or on multiple tasks. Especially, we divide the predictive business process monitoring technology into the following two types according to different learning methods.

### A. PREDICTION BASED ON TRADITIONAL MACHINE LEARNING

Regarding business process prediction, different researchers have put forward different views at different times. Van der Aalst et al. [13] proposed a method to predict the remaining execution time of business processes by constructing a Transition System (TS). However, only predicting the remaining time of the business process cannot monitor the future execution of the business process well. On this basis, Polato et al. [14] improved the Transition System and proposed a method based on Data-aware Transition System

**TABLE 1.** The studies collected for predictive business process monitoring.

| **Author, Year** | **Reference** | **Method type** | **Prediction target** (S: separate / J: joint) | **Input data** |
|---|---|---|---|---|
| Van der Aalst et al., 2011 | [13] | Transition System | the remaining time (S) | activity, attributes, timestamp |
| Polato et al., 2018 | [14] | Support Vector Regression, Transition System | the sequences of future activities and the remaining time (S) | activity, attributes, timestamp |
| Breuker et al., 2016 | [15] | Probabilistic Finite Automaton (PFA) | the next activity (S) | activity, attributes, timestamp |
| Lakshmanan et al., 2015 | [16] | Markov Chain | the sequences of future activities (S) | activity, attributes, timestamp |
| Conforti et al., 2013 | [17] | Decision Tree | the sequences of future activities and the remaining time (S) | activity, attributes, timestamp, resource |
| Leontjeva et al., 2016 | [18] | Random Forest | outcome (S) | activity, attributes |
| Appice et al., 2019 | [4] | Shallow Machine Learning | the next activity and its timestamp (J), the remaining time (S) | activity, attributes, timestamp (extended time features) |
| Evermann et al., 2016 | [21] | RNN | the next activity (S) | activity, attributes |
| Tax et al., 2017 | [9] | LSTM | the next activity and its timestamp (J), the sequences of future activities, the remaining time (S) | activity, timestamp (extended time features) |
| Camargo et al., 2019 | [24] | LSTM | the next activity and its timestamp (J), the sequences of future activities, the remaining time, and the related resource pool (S) | activity, timestamp (extended time features), resource |
| Navarin et al., 2017 | [25] | LSTM | the remaining time (S) | activity, attributes, timestamp (extended time features) |
| Francescomarino et al., 2017 | [26] | LSTM | the sequences of future activities (S) | activity, linear temporal logic, timestamp (extended time features) |
| Hinkka et al., 2018 | [27] | GRU | the next activity in checkpoints (S) | activity, attributes |
| Bukhsh et al., 2021 | [11] | Transformer | the next activity and its timestamp (J), the remaining time (S) | activity, timestamp |
| Wickramanayake et al., 2022 | [28] | LSTM+ the shared/specialized attention | the next activity (S) | activity, resource, timestamp (extended time features) |
| Taymouri et al., 2020 | [3] | GAN, LSTM | the next activity and its timestamp (J) | activity, resource, timestamp (extended time features) |
| Mauro et al., 2019 | [29] | CNN | the next activity (S) | activity, timestamp (extended time features) |
| Pasquadibisceglie et al., 2019 | [30] | CNN | the next activity (S) | activity, timestamp (extended time features) |
| Khan A et al., 2021 | [31] | Differential Neural Computer (DNC) | the next activity and its timestamp (J), the sequences of future activities (S) | activity, timestamp (extended time features) |
| Theis et al., 2019 | [32] | Deep Feedforward Neural Network | the next activity (S) | activity, attributes, timestamp (extended time features), process model |
| Chen et al., 2022 | [33] | BERT and transfer learning | the next activity and outcome (S) | activity, timestamp |

(DATS). At the same time, the SVR+TS method is proposed based on the traditional transition system and Support Vector Regression (SVR) to predict the future execution activity sequence and the remaining execution time, respectively. In addition, Breuker et al. [15] also proposed a method to predict future execution events of business processes using probabilistic finite automaton, which provided some feasible schemes for business process monitors to a certain extent. Similarly, Lakshmanan et al. [16] proposed using a Markov chain to construct an instance-specific Probabilistic Process Model to predict future execution activities of business processes. In addition, Conforti et al. [17] used the decision tree model to estimate the remaining execution time of business processes and predict the events of future process execution to support risk-informed decisions during business process execution. Similarly, Leontjeva et al. [18] also proposed using a random forest model to predict the final execution results of business processes. Besides, Appice et al. [4] investigated facets of shallow machine learning as an accurate data-centric approach to predict business process behaviour. Shallow machine learning is part of a holistic approach that combines feature construction, local and global learning, classification and regression algorithms. However, one drawback of such methods is that, in the case of low-level feature representation, their performance heavily relies on manual feature engineering [2], [19].

## B. PREDICTION BASED ON DEEP LEARNING

Deep learning algorithms have gradually replaced the method of selecting and extracting features manually in the past two decades due to the continuous development of deep learning technology research. Due to the autonomous learning characteristics of neural networks, deep learning techniques applied in predictive business process monitoring can significantly improve the process prediction performance of business process execution logs with large amounts of data. In [20], Rama-Maneiro et al. provided both a systematic literature review of these approaches and performed an exhaustive experimental evaluation. For example, Evermann et al. [21] proposed using a recurrent neural network (RNN) to predict the next business activity to be executed, composed of two hidden RNN layers. However, RNN has some performance defects due to its network structure, such as the inability to capture the long-distance dependence features in sequential data and the easy occurrence of gradient explosion and disappearance [22]. Therefore, the LSTM network, a variant of RNN, was proposed, Tax et al. [9] proposed the method of using an LSTM network to predict the type of event to be executed next in the process and its duration. In addition, Teinemaa et al. [23] analyzed and evaluated existing methods for predicting process results based on machine learning and LSTM. At the same time, Camargo et al. [24] also proposed a way to predict the event sequence, time, and related resource pool of future execution based on the LSTM network architecture. Similarly, Navarin et al. [25] utilized LSTM network to predict the remaining time in PPM. Francescomarino et al. [26] also leveraged knowledge about the structure of the process execution traces as well as a-priori knowledge about how they will unfold in the future for predicting the sequences of future activities based on LSTM.

In contrast to the starting point of the above approaches, Hinkka et al. [27] point out that existing RNN-based process prediction studies do not sufficiently use event attribute information. They believe that for attributes and attribute values of events in a process, the length of the encoded vector is often substantial, affecting the performance of process prediction methods based on RNNs to a certain extent. Therefore, they proposed a new clustering technology, which clustered the event attributes and their values before coding. Then, they input the obtained vector into the RNN before training the prediction model to realize the model training that can allow the trade-off of prediction accuracy, model training, and time required for prediction.

Based on these mainstream process prediction methods based on RNN and LSTM networks, some studies still use the attention mechanism as an optimization strategy to improve the performance of prediction models. For example, Bukhsh et al. [11] proposed the Process Transformer model, that is, to modify the structure of the Transformer network according to specific process prediction tasks to achieve ideal prediction results. Similarly, Wickramanayake et al. [28] proposed two types of attention for the prediction task of future activities: the event-level attention to capturing the impact of specific events on the prediction task and the attribute-level attention to reveal which attributes of events affect the prediction task. And they designed two different attention models, the shared and specialized attention-based models. The difference between them is that the attribute-level attention value is calculated for an input feature (specialized attention model), or the attribute-level attention value is constructed using the connected feature tensor of all input feature vectors (shared attention model). Finally, the specific prediction task is realized by combining an LSTM network.

In addition to the above approaches, some methods based on other neural networks have been gradually proposed. For example, Taymouri et al. [3] presented a novel adversarial training framework based on an adaptation of Generative Adversarial Networks (GANs) to the realm of sequential temporal data to predict the next activity and its timestamp. Mauro et al. [29] studied how to use the stacked inception Convolutional Neural Networks (CNN) module to predict the next activity and compared its performance with the LSTM network. Similarly, Pasquadibisceglie et al. [30] applied a CNN network to predict and analyze the business process. Besides, Khan et al. [31] proposed a memory-augmented Neural Network (MANN) to recommend the sequence of events to be executed in the subsequent process. Theis and Darabi [32] took the execution time of activities in the process as a primary variable. They used the time-decay function to enhance the Petri net process model that was constructed by the process mining algorithm to construct continuous process state samples. Then, on this basis, deep learning technology is used to train the prediction model to predict future activities.

Recently, Chen et al. [33] proposed a multi-task prediction method based on BERT and transfer learning. It's novelty lies in utilizing a new pre-training task to learn a generic representation of historical process instances (i.e., traces) and exploring the impact of masking strategies and masking probabilities on the performance of different prediction tasks. Different from it, our approach focuses on the multi-task prediction simultaneously. As shown in Table 1, we can find that the prediction target of most current methods usually trains the next activity and its timestamp together, but it is generally separate from other tasks. In addition, the input data of most methods is mostly derived from complex processing of business process event logs, such as the extended time features other than the original timestamp attribute. On the one hand, these existing methods are designed to handle the specified task. Once the specified task changes, we cannot effectively evaluate the performance of these methods. On the other hand, there is a strong correlation between some tasks in predictive process monitoring, but existing studies have paid little attention to this. Some knowledge can be shared and transferred among these tasks as they learn feature representations. In this way, we can reduce the prediction cost and time complexity and improve the prediction performance to some extent.

Therefore, this paper proposes an approach called MTLFormer, based on multi-task learning and the Transformer network for multiple tasks in PPM. Meanwhile, it uses very few attributes with minimal preprocessing of event logs. To a certain extent, MTLFormer utilizes the attention model in the Transformer network to better capture the long-distance dependency features in the historical process executions. Under the guidance of multi-task learning, it can improve the efficiency and accuracy of prediction by sharing the knowledge learned from different tasks.

## III. PROBLEM DEFINITION

### A. EVENT LOG

The process execution in PAIS can generate many event logs, which record detailed information about each business process execution. Each complete execution of a business process can obtain a specific process instance (case). Each of them consists of a series of events, and each event corresponds to all the information about the execution of an activity in this business process. The relevant definitions of the event log are as follows.

*Definition 1 (Event):* An **event** is the smallest unit in the event log and can be defined as $e = (a, caseID, eventID, t_{start}, t_{end}, r, d_1, \ldots, d_m)$, in which $a$ denotes the activity name (attribute) in a business process of this event $e$, $caseID$ denotes the ID number of the process instance that the event $e$ occurred in, $eventID$ denotes the ID number of this event, $t_{start}$ and $t_{end}$ represent the timestamp attribute at which the event starts and completes respectively, $r$ represents the resource attribute of event $e$, $d_1, \ldots, d_m$ represent the other attributes of the event $e$. As for different events, the values of the above attributes are different.

*Definition 2 (Process Trace):* Each historical process instance (case) in the event log corresponds to a **process trace**, which consists of an ordered sequence of several events, denoted as $\sigma =< e_1, e_2, \ldots, e_{|\sigma|} >$, in which $|\sigma|$ is the number of events contained in this case.

*Definition 3 (Prefix Trace):* A **prefix trace** is an ordered sequence of the first $k$ events in a process trace $\sigma$ that can be defined as $prefix\,(\sigma, k) =< e_1, e_2, \ldots, e_k >, k \in [0, |\sigma|]$, where $k$ represents the length of the prefix trace, i.e., the length of the sequence intercepted from the process trace $\sigma$, and the timestamp attribute of all the events in the prefix trace $\sigma$ increases sequentially.

### B. PREDICTION TASKS

Take a process trace $\sigma =< e_1, e_2, \ldots, e_n >$ for an example, the concepts of the three prediction tasks studied in this paper are defined as follows:

*Definition 4 (Next Activity Prediction Task):* As for the trace $\sigma$, the **Next Activity Prediction Task** can be defined as a function $f_{na}\,(prefix\,(\sigma, k)) = e_{k+1}.a, k\in [1,n-1]$ to predict the activity attribute of the next event (denoted as *next_event*) at the current stage of process execution where the first $k$-th events are executed. Since the name of the next event to be predicted is an element in the set of activities corresponding to the whole process, the task is considered a multi-classification prediction task in this paper.

*Definition 5 (Next Event Time Prediction Task):* As for the trace $\sigma$, the **Next Event Time Prediction Task** can be defined as a function $f_{nt}\,(prefix\,(\sigma, k)) = (e_{k+1}.t_{end} - e_k.t_{end})/24, k\in [1,n-1]$ to predict the difference in days (denoted as *next_time*) between the end timestamp of the next event $e_{k+1}$ (i.e., $e_{k+1}.t_{end}$) and that of the currently executed event $e_k$(i.e., $e_k.t_{end}$) where the end timestamp of $e_k$ is determined. Since the time to be predicted is also a continual value, the task is considered a regression prediction task in this paper.

*Definition 6 (Remaining Time Prediction Task):* As for the trace $\sigma$, the **Remaining Time Prediction Task** can be defined as a function $f_{rt}\,(prefix\,(\sigma, k)) = (e_n.t_{end} - e_k.t_{end})/24, k\in [1,n]$ to predict the difference in days (denoted as *remain_time*) between the end timestamp of the last event $e_n$ (i.e., $e_n.t_{end}$) and that of the currently executed event $e_k$(i.e., $e_k.t_{end}$) where the end timestamp of $e_k$ is determined. Since the time to be predicted is also a continual value, the task is considered a regression prediction task in this paper.

### C. MULTI-TASK LEARNING

Generally speaking, as long as a model has more than one objective task and there is a particular connection among these tasks can be called Multi-task Learning (MTL) [12]. The related training process is to put the inputs of all tasks into a multi-task learning model after data pre-processing and then get the set of outputs of all tasks. During the training process of multi-task learning, different tasks will influence and learn from each other, thus improving the model's generalization ability. The core of multitask learning is how to share parameters between models corresponding to different tasks. The modes of parameter sharing are usually divided into hard- and soft- parameter sharing [34].

*Definition 7 (Multi-Task):* The three prediction tasks in Definitions 4 to 6 are called Task A, Task B, and Task C. The **multi-task** mentioned in this paper refers specifically to the three tasks, Task A, Task B, and Task C.

*Definition 8 (Multi-Input, Multi-Output):* The input to the multi-task learning model (i.e., **Multi-input**) is obtained by splicing the inputs of Task A, Task B, and Task C and then removing the duplicate parts, while the output to the multi-task learning model (i.e., **Multi-output**) is concatenated by the related outputs of Task A, Task B, and Task C. Suppose the inputs to each task are $A_{in} = [a, b, c]$, $B_{in} = [b, c, d]$, and $C_{in} = [c, d, e]$, the **Multi-input** can be defined as $inputs = [a, b, c, d, e]$. If the outputs of each task are $A_{out} = f$, $B_{out} = g$, and $C_{out} = h$, the **Multi-output** can be defined as $outputs = [f, g, h]$.

*Definition 9 (Hard-Parameter Sharing):* As shown in Fig. 1(a), **hard-parameter sharing** refers to the input of multiple tasks to the same shared layer, i.e., sharing the same parameter $W$, and then outputting each task separately in the deep layer with the number of outputs equal to the number of tasks. The parameter $W$ is a variable parameter that is continuously updated during the back-propagation of the neural network, not an artificial hyper-parameter set such as the learning rate and the dimension of the hidden layer.

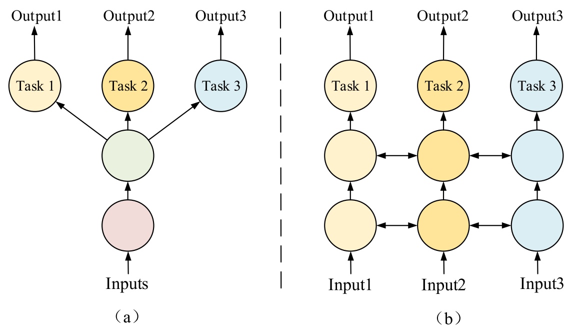

**FIGURE 1. Two parameter sharing modes: (a) hard-parameter sharing and (b) soft-parameter sharing.**

*Definition 10 (Soft-Parameter Sharing):* As shown in Fig. 1(b), **soft-parameter sharing** refers to each task having a separate model space in multi-task learning. Unlike single-task learning, it sets the $L2$ distance norm regularization between each model parameter, i.e., the difference between each model parameter $W$ does not exceed a certain threshold. Given two tasks, $x$ and $y$, the distances between their corresponding model parameters are expressed as:

$$||x, y||_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$
$$where \quad ||x||_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{1}$$

in which $||x||_2$ denotes the $L2$ parametrization of the input vector $x$ (i.e., the norm of the vector), $n$ is the vector dimension, and $||x, y||_2$ (the $L2$ parametrization of $x$ and $y$) represents the distance between these two vectors (i.e., the distance between different task model parameters within a certain range).

## IV. MTLFORMER APPROACH

In this paper, we present an approach called MTLFormer (Multi-Task Learning Guided for Transformer Network) based on multi-task learning and the Transformer network in Predictive Process Monitoring. Specifically, for multiple associated tasks (i.e., the three prediction tasks mentioned above: the next activity prediction, the next event execution time prediction, and the remaining time prediction), a multi-task fusion prediction model can be constructed using MTLFormer. In the prediction phase, the multi-task parallel prediction online can come true by exploiting this model. The approach consists of two main phases: the data pre-processing and the construction and training of the neural network model.

### *A. DATA PRE-PROCESSING*

According to Definitions 1 to 3, assume that the event log dataset for a business process is $L = <\sigma_1, \sigma_2, \ldots, \sigma_n>$, where $n$ represents the number of process traces contained in the event log, and $\sigma$ represents the process trace described in Definition 2. In the data pre-processing stage, the event log $L$ must be extracted first. Specifically, some operations can be conducted based on the original attributes of each event in each process instance: (1) for each event, a prefix trace *prefix* attribute is added to indicate the current moment of the process instance; (2) for each event, add three additional temporal attributes based on the start and end timestamp attributes of the event, *recent_time* (denotes the time interval between the two most recent events and the current event), *latest_time* (denotes the time interval between the current event and an event before the previous event), *time_pass* (denotes the time interval between the start time of the process instance and the occurrence of the current event); (3) add three additional label attributes corresponding to each of the three prediction tasks *next_event*, *next_time*, and *remain_time*. After that, a process trace dataset can be obtained for the fusion of the three tasks. For each of these traces, each event can be viewed as a set of attributes noted as $e = \{c, i, prefix, recent\_time, latest\_time, time\_pass\}$ and a set of three prediction task label attributes $\{next\_event, next\_time, remain\_time\}$. Next, the attributes of each event in the process instance are encoded and normalized according to the data type to which the attribute values belong. Finally, the vector obtained from the encoding of the process instances is used as the input to the neural network model, and the labeled attributes of the entire process instances are used as the target output of the model.

### *B. NEURAL NETWORK BUILDING AND TRAINING*

In order to solve the problem that single-task learning cannot capture the potential feature between multiple tasks and the current LSTM-based (RNN-based) approaches cannot capture the long-distance dependency in sequential data, this paper combines Transformer network and multi-task learning to capture the association relationship between the feature learning of the different tasks. Moreover, the particular attention mechanism in the Transformer network used here can further learn the long-distance dependent features in the data. In addition, our approach also uses location coding and residual network to enhance the model's ability to perceive location information as well as address the gradient disappearance and degradation of the weight matrix, respectively. The structure of the neural network model constructed by MTLFormer proposed in this paper mainly consists of the (process) Trace Embedding Layer, the Feature Extraction Layer, the Hard-param Sharing Layer, and the MTL Output Layer, as shown in Fig. 2.

Taking a process trace $\sigma = <e_1, e_2, \ldots, e_i, \ldots, e_m>$ ($i \in [1, m]$) as an example ($m$ denotes the number of events in the process trace) to illustrate in detail how the MTLFormer approach is trained to obtain the prediction model. First, the process trace $\sigma$ is pre-processed to obtain an encoded feature vector of its attributes (except for the additional temporal attributes) $x_p$ and a vector of temporal feature attributes $x_t$ and then concatenated to obtain the vector $X = [x_p, x_t]$ as the input to the neural network. Here, $x_p$ and $x_t$ correspond

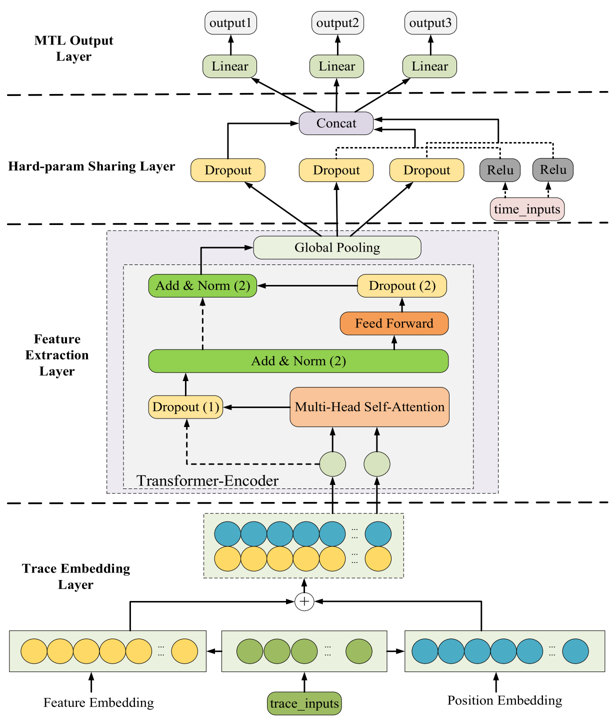

FIGURE 2. The structure of the neural network model constructed by MTLFormer.

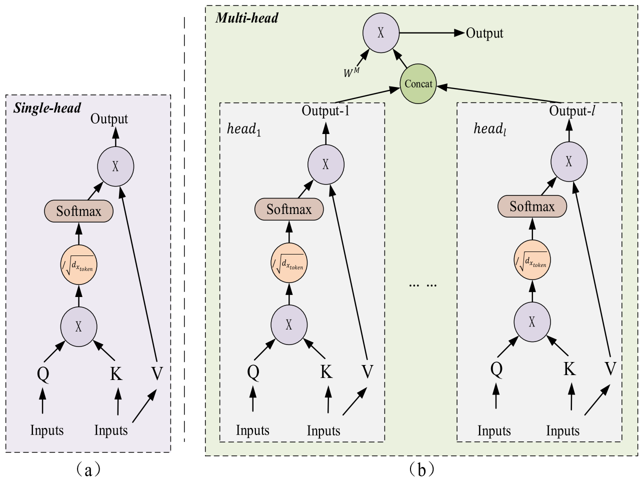

FIGURE 3. The structures of two different attention mechanisms: (a) Single-head self-attention mechanism and (b) Multi-head self-attention mechanism.

to the *trace_inputs* and *time_inputs*, respectively, in Fig. 2. The main structure of the neural network constructed by the MTLFormer approach is shown below.

**Trace Embedding Layer.** The process trace embedding layer consists of two main parts, one for feature embedding and the other for position embedding. Since the one-hot encoding [35] uses $N$ spaces to represent $N$ states, it is also called one-bit effective coding, that is, only one bit is valid and this position is set to 1, while the rest of the positions are set to 0. In terms of the classification tasks, the feature space of the encoding vector increases dramatically as the category increases, resulting in an intractable high-dimensional vector problem. Therefore, *trace_inputs* can be linearly mapped into a larger vector space to obtain the feature embedding vector $x_{attr} = [x_{attr,1}, x_{attr,2}, \ldots, x_{attr,m}]$. In this way, it not only solves the current high-dimensional problem encountered with one-hot encoding but also maps adjacent events to adjacent positions in vector space. As the moment-to-moment state information of the recurrent neural network is discarded in the MTLFormer model, it is difficult to determine the relative position of the data on the whole. Thus, the position of *trace_inputs* should be embedded simultaneously. The position embedding is similar to feature embedding in that the position information of the events is mapped into a larger vector space, resulting in a position embedding vector $x_{pos} = [x_{pos,1}, x_{pos,2}, \ldots, x_{pos,m}]$. Finally, the feature embedding vector is concatenated with the position embedding vector to obtain the output $x_{token} = [x_{attr}, x_{pos}] = [x_{token,1}, x_{token,2}, \ldots, x_{token,m}]$($d_{x_{token}}$ denotes the dimension) of the Trace Embedding Layer.

**Feature Extraction Layer.** The feature extraction layer is made up of the Encoder of the Transformer network and the global average pooling layer (Global Pooling) [36]. In order to learn long-distance dependent features of data, the output vector $\boldsymbol{x_{token}}$ of the Trace Embedding Layer can be fed into the multi-headed attention and then computed to obtain the output. At the same time, to reduce the probability of overfitting, the output of the multi-headed attention is dropped through the Dropout (1) layer, and some features are then summed using the residual [37], [38] for layer normalization [39] (Add & Norm (1)). The obtained intermediate vectors are then fed into the Feed Forward Network [40], [41] with a dropout layer (Dropout (2)) to discard some of the features and then perform the residual summation and the layer normalization.. (Add & Norm (2)) to obtain the output of the Transformer-Encoder. Finally, the output vector of the feature extraction layer is obtained by the global average pooling layer (Global Pooling). The multi-headed attention mechanism is shown in Fig. 3(b).

As shown in Fig. 3(a), the (single-head) self-attention [10] is a structure that can be executed concurrently, almost regardless of distance. It scores each part of the input separately, with higher scores being more important for the prediction task and lower scores being given less weight during training. Particularly, the attention mechanism used here is the Scaled Dot-Product Attention model, which is calculated by:

$$att(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_{x_{token}}}}\right)V \qquad (2)$$

in which, the *softmax*() denotes the normalization function, $d_{x_{token}}$ is the dimension of the input vector $x_{token}$, $Q$ is the *Query* vector, $K$ is the *Key* vector, $V$ is the *Value* vector, and *att* is the attention score. Particularly, $Q$, $K$, and $V$ are obtained by:

$$Q = W^{Q}X + b^{Q} \tag{3}$$
$$K = W^{K}Y + b^{K} \tag{4}$$
$$V = W^{V}Y + b^{V} \tag{5}$$

in which, $W^{Q}$, $W^{K}$, and $W^{V}$ are the weight parameters of the corresponding vectors $Q$, $K$, and $V$, $b^{Q}$, $b^{K}$, and $b^{V}$ are the bias of the corresponding vectors $Q$, $K$, and $V$, and $X$, $Y$ are the two inputs of this attention mechanism ($X = Y = x_{token}$).

Similar to the (single-head) attention mechanism, the multi-head self-attention mechanism [35] described in Fig. 3(b) is used in this paper where the parameters $W$ in the single-head attention mechanism can be calculated several times based on the number of heads in the multi-head attention mechanism, and then combined to obtain the multi-head attention score. Given the multi-attention mechanism with $l$-heads, the multi-head attention score $attMulti\,(Q, K, V)$ with the related weight parameters $W_i^{Q}, W_i^{K}, W_i^{V} (i = 1, 2, \ldots, l)$ can be calculated by:

$$\begin{aligned} attMulti\,(Q, K, V) &= concat\,(head_1, head_2, \ldots, head_l)\, W^{M} \\ where\ \ head_i &= att\,(Q_i, K_i, V_i)\,, i = 1, 2, \ldots, l \end{aligned} \tag{6}$$

where $concat()$ is the vector concatenated function, $W^{M}$ (the dimension is the multiplication of the dimension of the input vector by the number of heads) is the weight parameter to be trained for multi-head attention, $Q_i = QW_i^{Q}$, $K_i = KW_i^{K}$, and $V_i = VW_i^{V}$ are the *Query* vector, *Key* vector, and *Value* vector corresponding to each head, respectively. Finally, the attention output vector of the feature extraction layer in the MTLFormer model can be obtained with $attMulti\_output = attMulti\,(x_{token}, x_{token}, x_{token})$.

Then, the output vector of the multi-head attention mechanism is processed by Dropout (1) to obtain $attMulti\_output = dropout(attMulti\_output)$. After that, the output values $attMulti\_output$ are then summed with its input values and then performed by the layer normalization, as described in:

$$output\_a = norm(attMulti\_output + x_{token}) \tag{7}$$

where $output\_a$ is the output after the Add & Norm (1) process, and $norm()$ is the normalization function.

Similarly, the output vectors obtained above are processed by Feed Forward, Dropout (2), and Add & Norm (2) again in turn to obtain:

$$FF\_output = FF\,(output\_a) \tag{8}$$
$$FF\_output = dropout(FF\_output) \tag{9}$$
$$output\_t = norm(FF\_output + output\_a) \tag{10}$$

where $FF\_output$ is the output vector after the Feed Forward neural network, $FF$ is the related function, $output\_t$ is the final output of the Transformer-Encoder. Finally, the dimension of $output\_t$ is reduced after the global average pooling operation, and then the final output $output_{fe}$ of the feature extraction layer can be obtained.

**Hard-Param Sharing Layer.** In order to achieve multi-task learning, feature learning for the three tasks is shared and processed at this layer. Feature learning is performed differently for different types of tasks. For instance, tasks related to predicting time require the input of temporal features, while the next activity prediction task does not. The detailed computation is as follows.

$$N = dropout(output_{fe}) \tag{11}$$
$$T = concat(dropout\left(output_{fe}\right), relu(x_t)) \tag{12}$$
$$R = concat(dropout\left(output_{fe}\right), relu(x_t))) \tag{13}$$
$$output_{hs} = concat(N, T, R) \tag{14}$$

In the above equations, $N, T, R$ are the feature vectors for the three tasks of the next activity prediction, the next event time prediction, and the remaining time prediction, respectively, $concat()$ is the vector concatenated function, $x_t$ is the input vector of temporal features obtained in the pre-processing stage, $relu\,(x) = \max(0, x)$ is the linear activation function, and $output_{hs}$ is the final output vector of the hard-parameter sharing layer.

**MTL Output Layer.** After the hard-parameter sharing layer, separate outputs are required for each task to obtain three separate prediction results, as shown in:

$$\begin{aligned} &\hat{y}^{(1)}(output_1), \hat{y}^{(2)}(output_2), \hat{y}^{(3)}(output_3) \\ &\quad = linear\left(output_{hs}\right) \end{aligned} \tag{15}$$

where $\hat{y}^{(i)}(output_i)$ denotes the vector of outputs (i.e., the predicted results) for all events in terms of the $i$-th task after the calculation above and $linear()$ is the activation function of the Fully Connected (FC) layer. The activation function is described as:

$$linear(X) = \sum_j W_I x_j + b \tag{16}$$

where $W_I$ is the weight parameter, and $b$ is the bias value.

The true label vector $Y = [y^{(1)}, y^{(2)}, y^{(3)}]$ can be obtained from the three prediction task label attributes tagged in the data pre-processing stage where $y^{(1)}$ denotes the actual result of the next activity prediction task, $y^{(2)}$ denotes the true result of the next event time prediction task, and $y^{(3)}$ denotes the true result of the remaining time prediction task. For each task, the loss value of the entire multi-task model can be calculated separately from the output of the MTLFormer prediction model and the actual results (i.e., the label vectors). Finally, the model is continuously optimized using training samples to obtain a fixed set of parameter values that minimize the loss function, i.e., the final prediction model to be trained.

As the MTLFormer prediction model is designed for multiple tasks, it is necessary to calculate the loss values for each task and then combine them to obtain the multi-task loss function. Specifically, the first task dealt with in this paper is essentially a multi-classification prediction, for which a multi-classification cross-entropy loss function is often used to calculate the error. In contrast, the other two tasks are essentially regression predictions, for which a *LogCosh* loss function is used to calculate the error. The multi-classification cross-entropy loss function is described as:

$$loss(\sigma) = -\frac{1}{m}\sum_{i=1}^{m} \hat{y}_i \log(y_i) \tag{17}$$

where $loss(\sigma)$ denotes the loss value of the process trace $\sigma$ in the next activity prediction task, $m$ denotes the total number of events contained in this trace, $\hat{y}_i$ denotes the predicted value of the $i$-th event in this trace, and $y_i$ denotes the actual value (i.e., the label value). Accordingly, the loss value of an event log $L = <\sigma_1, \sigma_2, \ldots, \sigma_n>$ (where $n$ represents the number of process traces contained in this event log) fed into the MTLFormer prediction model can be obtained for the next activity prediction task by:

$$Loss_1 = \sum_{j=1}^{n} loss(\sigma_j) \tag{18}$$

where $Loss_1$ denotes the sum of the loss values of all process traces in the event log $L$ for the next activity prediction task. For the rest two prediction tasks, the *LogCosh* loss function is used to calculate their error values in the event log $L$ by:

$$logcosh\,(\sigma) = \frac{1}{m}\sum_{i=1}^{m} log \frac{e^{(\hat{y}_i - y_i)} + e^{(y_i - \hat{y}_i)}}{2} \tag{19}$$

$$Loss_2 = \sum_{j=1}^{n} logcosh(\sigma_j) \tag{20}$$

where $logcosh\,(\sigma)$ denotes the loss value of the process trace $\sigma$ for the next event time prediction task, $m$ is the total number of events contained in this trace, $\hat{y}_i$ is the predicted value of the $i$-th event in this trace, and $y_i$ is the actual value (i.e., the label value), and $Loss_2$ denotes the sum of the loss values of all process traces in the event log $L$ for the next event time prediction task. Similarly, $Loss_3$ that denotes the sum of the loss values of all process traces in the event log $L$ for the remaining time prediction task can be calculated. Based on them, the total loss function of the MTLFormer model can be denoted as:

$$Loss = (\omega_1 * Loss_1 + \omega_2 * Loss_2 + \omega_3 * Loss_3) \tag{21}$$

where $\omega_1$, $\omega_2$, and $\omega_3$ are the weights of the corresponding loss functions for each task, respectively.

After the prediction model has been trained, the sequence of events that are executed in the on-going process instance can be used as input to the prediction model to obtain multi-task prediction values for achieving online prediction.

## V. EXPERIMENTAL EVALUATION

In order to verify the effectiveness of the MTLFormer proposed in this paper, we conduct experiments on four real-life event logs. The specific experimental environment and experimental setup are described as follows. Moreover, we conduct the ablation experiments with multi-task learning and the Transformer network as the baseline. The experimental results are collated for each prediction task separately to comprehensively evaluate the different approaches' prediction results. The main comparison is performed among some approaches, MTLFormer, the ProcessTransformer approach (called STLFormer) proposed in [11], and the proposals in other references. In this paper, we choose the four metrics of accuracy, precision, recall, and F-score for evaluating the performance of different approaches in the next activity prediction task. As for the rest tasks of the next event time prediction and the remaining time prediction, we choose the Mean Absolute Error (MAE) metric to evaluate their performance.

### A. EXPERIMENTAL SETUP

The datasets used in our experiment are derived from real-life event logs in the 4TU research data repository (https://researchdata.4tu.nl/home/). For comparison purposes, we choose the following event logs, Helpdesk and BPIC2012. The detailed information is shown below.

(1) Helpdesk: This dataset is derived from the ticket management system of an Italian software company. It contains 4,580 process instances, 21,348 events, and 14 activities.

(2) BIPC2012: This dataset originates from a Dutch financial institution and it records detailed information about the business process of a loan application. It contains 13,087 process instances, 262,200 events, and 23 activities. As it is a merging of three intersecting sub-processes, the log can be further divided into three logs, BPIC2012_A, BPIC2012_O, and BPIC2012_W, based on the initial letter identification of the different names for the different tasks.

The environment of our experiment was configured as shown in Table 2. We implemented the methods in this paper on Python 3.8 using Tensorflow 2.5.0. All experiments in this paper are performed on Windows 10, a 2×12 Intel Xeon 5118 CPU @2.30GHz 256GB, and 3× NVIDIA Tesla V100.

The evaluation metric for predictive models is to substitute the predicted and actual values into the evaluation formula and evaluate the performance in terms of the output value. Standard metrics evaluated for multi-classification prediction tasks are as follows.

(1) accuracy

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{22}$$

(2) precision

$$precision = \frac{TP}{TP + FP} \tag{23}$$

(3) recall

$$recall = \frac{TP}{TP + FN} \tag{24}$$

**TABLE 2.** **The environment of our experiment.**

| Configuration Item | Configuration Parameter |
|---|---|
| Development Language | Python 3.8 |
| Development Tool | PyCharm |
| Development Framework | Tensorflow 2.5.0 |
| CPU | 2 × 12 Intel Xeon 5118 CPU @2.30GHz 256GB |
| GPU | 3 × NVIDIA Tesla V100 |

(4) F-score

$$F-score = 2*\frac{precision*recall}{precision + recall} \tag{25}$$

In these equations above, *TP* (*True Positive*) refers to a sample with a positive predicted value and a positive true value, *FP* (*False Positive*) refers to a sample with a positive predicted value and a negative true value, *TN* (*True Negative*) refers to a sample with a negative predicted value and a negative true value, and *FN* (*False Negative*) refers to a sample with a negative predicted value and a positive true value. All of these indicators are always used to assess classification prediction tasks. Note that even for multi-classification problems, there are only correct and incorrect assessments, so this assessment indicator can again be considered a binary assessment problem. The higher the value of the first three indicators, the more effective the method is. The indicator that is always used in regression prediction tasks is the Mean Absolute Error (MAE):

$$MAE = \frac{1}{m} * \sum_{i=1}^{m} \left|\left(y_i - \hat{y}_i\right)\right| \tag{26}$$

where $m$ denotes the total number of events, $y_i$ denotes the actual value, and $\hat{y}_i$ denotes the predicted value. The smaller the value of *MAE*, the smaller the prediction error (i.e., the better the prediction performance of the model).

## B. DATA PRE-PROCESSING AND PARAMETER SETTING

This section feeds four real-life event log datasets into the built neural network model for training. Each dataset is first pre-processed and then used as the training set for this model by extracting the first 80% according to the order the events occurred and the last 20% as the test set to evaluate the performance of the prediction models obtained from different approaches of training. For each process instance, separate data pre-processing is performed as described in Section III-A. For instance, the attribute *recent_time* of an event is obtained by subtracting its end timestamp from that of the two most recent events in this process instance and dividing it by 24 hours and rounding down, the attribute *latest_time* of an event is obtained by subtracting its end timestamp from that of the previous event in this process instance and dividing it by 24 hours and rounding down, and the attribute *time_pass* of an event is obtained by subtracting its end timestamp from that of the first event in this process instance and dividing it by 24 hours and rounding down. Additionally, 20% of the data from the training set is used for validation and hyper-parameter optimization during the multi-task learning. The model based on MTFormer is trained for 100 epochs with an Adam [42] optimizer and a learning rate of 0.002. In addition, the prediction model (i.e., classifier) is developed with a single transformer-encoder block (i.e., Feature Extraction Layer) that consists of a specific four-head attention mechanism, followed by feed-forward layers having residual connections, dropout, and a normalization layer in this paper.[1] Specifically, a *linear* connection layer with a dimension of 128 is used in its MTL Output Layer, and a 36- and 128-dimensional *Relu* linear activation function is also used in the Hard-parameter Sharing Layer, respectively. The optimized hyper-parameters on the validation set are a batch size of 64 and a list of loss weight $\{\omega_1$: 0.6, $\omega_2$: 2, $\omega_3$: 0.3$\}$. As for the model based on STLFormer, we trained it according to [11].

## C. EXPERIMENTAL RESULTS

A comprehensive comparison between MTLFormer and STLFormer is carried out to demonstrate the effectiveness of multi-task learning. The currently available other approaches are then evaluated against these two approaches separately for different prediction tasks to demonstrate the effectiveness of the attention mechanism in the Transformer network (i.e., the approaches that do not use the Transformer network *vs.* the two approaches of MTLFormer and STLFormer) to demonstrate the effectiveness of the Transformer network in PPM. First, the comparison results of MTLFormer and STLFormer are shown in Table 3.

It is easy to see from Table 3 that MTLFormer outperforms STLFormer on Helpdesk and BPIC2012_A in terms of prediction performance, with an average improvement of about 7% in accuracy and precision as well as that of about 12% in recall and f-score for the next activity prediction, with an average decrease of about 10% and 22% in MAE for the time prediction tasks. However, on the other two datasets, the accuracy and precision of MTLFormer are slightly lower than that of STLFormer, and the MAE is slightly larger for the event time prediction task but the MAE is much smaller for the remaining time prediction task. By jointly learning from multiple tasks, MTLFormer introduces the concept of task sharing and transfer learning within the Transformer model. It can capture shared information to improve the

[1] https://github.com/jiaojiaowang1992/MTLFormer

**TABLE 3.** The comparison results of MTLFormer and STLFormer on the different datasets.

| Approach | Dataset | next activity | | | | next event time | remaining time |
|---|---|---|---|---|---|---|---|
| | | accuracy/% | precision/% | recall/% | f-score/% | MAE/day | MAE/day |
| MTLFormer | Helpdesk | **90** | **86** | **89** | **87** | **2.84** | **3.70** |
| | BPIC2012_A | **84** | **75** | **78** | **76** | **1.01** | **3.12** |
| | BPIC2012_O | 83 | **89** | 85 | **87** | 1.17 | **3.69** |
| | BPIC2012_W | 88 | 85 | **90** | 89 | 0.46 | **1.86** |
| | *Mean* | **86** | **84** | **86** | **85** | **1.37** | **3.09** |
| STLFormer [11] | Helpdesk | 86 | 82 | 85 | 81 | 3.14 | 4.04 |
| | BPIC2012_A | 77 | 68 | 64 | 65 | 1.12 | 4.72 |
| | BPIC2012_O | **87** | 85 | **87** | 84 | **1.13** | 5.28 |
| | BPIC2012_W | **90** | **89** | 85 | **90** | **0.40** | 5.03 |
| | *Mean* | 85 | 81 | 80 | 80 | 1.45 | 4.77 |

overall performance compared to separate single-task models (i.e., STLFormer) and the shared layers of the model learn representations that are beneficial for multiple tasks, enabling the model to transfer knowledge and generalize across related tasks. Therefore, it can leverage the inherent relationships between tasks and potentially enhance overall performance. However, in our experiments, the performance of the observed BPIC2012_O and BPIC2012_W datasets did not improve but decreased. The reason may be that these two datasets are manually operated processes with much uncertainty, resulting in the learning of shared features by multiple tasks making the feature representations obtained from the learning of different tasks become harmful noise to each other. In general, the average metrics of MTLFormer improved across the board compared to STLFormer, thus demonstrating that multi-task learning can improve the performance of tasks with similar features.

Next, the performance of the different approaches will be analyzed and evaluated separately for each prediction task.

### 1) THE NEXT ACTIVITY PREDICTION TASK

This task is a multi-classification task. Since accuracy is the basic metric used in other methods, we chose accuracy for statistical analysis. The results of the experimental comparison are shown in Table 4. The table shows that MTLFormer has a higher average accuracy on different datasets than any other approaches in the next activity prediction task. For the two datasets of Helpdesk and BPIC2012_A, MTLFormer has the highest accuracy compared to the others. However, on the BPIC2012_O dataset, STLFormer has the highest accuracy, and on the BPIC2012_W dataset, the method in Chen et al. [33] is the most accurate. It is easy to see from this table that there is an advantage to using the Transformer network in this task because the three methods mentioned-above are based on this network.

### 2) THE NEXT EVENT TIME PREDICTION TASK

This task is a regression prediction task, so the MAE metric is used here to evaluate the prediction results. The prediction results of the different approaches for the four datasets mentioned above are shown in Table 5. On the whole, the table shows that MTLFormer improves prediction performance compared to STLFormer in the next event time prediction task, with an average reduction in the error metric MAE of approximately 0.08 days on the four datasets. For the datasets of Helpdesk and BPIC2012_A, the MAE error of MTLFormer is reduced by approximately 0.3 and 0.1 days, respectively. On the other two datasets, its MAE error improves somewhat, about 0.05 days on average. Moreover, from the perspective of each dataset, MTLFormer achieves optimal results only on the Helpdesk and is slightly inferior to the other approaches on the other three datasets. There may be three main reasons for this phenomenon. First, to improve the generalization of MTLFormer, we trade off multiple prediction tasks simultaneously in hyper-parameter optimization and only consider a set of optimization parameters to make it applicable to all datasets. While the STLFormer is used to optimize the hyper-parameters for each prediction task. The second reason may be that the learning features of the other tasks in these datasets (i.e., BPIC2012_A, BPIC2012_O, and BPIC2012_W) are less suitable to share with the next event time prediction task.

So it is not conducive to the advantage of MTLFormer sharing the learning features of multiple tasks. The last one may lie in the different input data when training models based on these approaches. We mainly perform minimal preprocessing on the raw event log data to obtain the input data of MTLFormer and STLFormer. In contrast, other approaches use extended time features as input data in addition to the raw event log. However, in general, MTLFormer is the most effective among the above comparison methods. In addition, from the perspective of the neural network used, the average prediction performance of the Transformer-based approaches is much better than that of the others. The average MAE error is reduced by approximately 0.8 days. Especially for the Helpdesk dataset, the MAE of MTLFormer is only 2.84 days, while that of the method in [31] is 6.36 days, with a difference

**TABLE 4. The comparison in the next activity prediction task (higher is better).**

| Approach | accuracy/% | | | | *Mean* |
|---|---|---|---|---|---|
| | Helpdesk | BPIC2012_A | BPIC2012_O | BPIC2012_W | |
| MTLFormer | **90** | **84** | 83 | 88 | **86.25** |
| STLFormer [11] | 86 | 77 | **87** | 90 | 85 |
| Method in Tax et al. [9] | 71 | 77 | 81 | 75 | 76 |
| Method in Evermann et al. [21] | 70 | 74 | 79 | 75 | 74.5 |
| Method in Camargo et al. [24] | 76 | 79 | 85 | 83 | 80.75 |
| Method in Hinkka et al. [27] | 78 | 79 | 86 | 84 | 79 |
| Method in Wickramanayake et al. (shared) [28] | / | 75 | 82 | 84 | 80 |
| Method in Wickramanayake et al. (specialized) [28] | / | 75 | 82 | 84 | 80 |
| Method in Breuker et al. [15] | 81 | / | / | 71 | 76 |
| Method in Mauro et al. [29] | 75 | 78 | 82 | 85 | 80 |
| Pasquadibisceglie et al. [30] | 66 | 71 | 78 | 82 | 74.25 |
| Method in Khan et al. [31] | 69 | 76 | 84 | 87 | 79 |
| Method in Theis et al. (w/o attributes) [32] | 68 | 66 | 82 | 86 | 75.5 |
| Method in Theis et al. (w/ attributes) [32] | 66 | 65 | 74 | 76 | 70.25 |
| Method in Chen et al. [33] | 76 | / | / | **94** | 85 |

**TABLE 5. The comparison in the next event time prediction task (lower is better).**

| Approach | MAE/day | | | | *Mean* |
|---|---|---|---|---|---|
| | Helpdesk | BPIC2012_A | BPIC2012_O | BPIC2012_W | |
| MTLFormer | **2.84** | 1.01 | 1.17 | 0.46 | **1.37** |
| STLFormer [11] | 3.14 | 1.12 | **1.13** | **0.40** | 1.45 |
| Method in Tax et al. [9] | 5.78 | 0.84 | 1.61 | 0.50 | 2.18 |
| Method in Khan et al. [31] | 6.36 | **0.75** | 1.45 | 0.50 | 2.27 |

of 3.52 days. Even though this method in [31] can achieve the minimum MAE for the BPIC2012_A dataset, the difference is only 0.26 days compared with MTLFormer. For the rest datasets, MTLFormer is only 0.04 days and 0.06 days away from the best results.

### 3) THE REMAINING TIME PREDICTION TASK

This task is similar to the next event time prediction task, so we also choose MAE as an evaluation metric. The prediction results of the different approaches on the four datasets mentioned above are shown in Table 6. The table shows that MTLFormer reduces the prediction error compared to STLFormer in the remaining time prediction task by approximately 0.3 days on the Helpdesk dataset, approximately 1.6 days on the BPIC2012_A dataset, approximately 1.6 days on the BPIC2012_O dataset, approximately 3.2 days on the BPIC2012_W dataset. In general, the average reduction in prediction error on these four datasets is about 1.7 days. Therefore, it is sufficient to demonstrate that MTLFormer has an advantage over STLFormer in predicting the remaining time. Compared with the non-Transformer-based approaches, the advantage of MTLFormer and STLFormer is obvious.

Fig(s). 4 to 7 show the comparison and change trends of the performance of MTLFormer and STLFormer on the datasets of Helpdesk, BPIC2012_A, BPIC2012_O, and BPIC2012_W as the length of the prefix trace increases in different prediction tasks, respectively. In each figure, the subfigures (a), (b), (c), and (d) show the comparisons and change trends of the four metrics (i.e., accuracy, precision, recall, and f-score) in the next activity prediction task as the length of the prefix trace increases. Each subfigure (e) shows the comparison and change trend of the MAE metric in the next event time prediction task (called MAE-nt) as the length of the prefix trace increases. Each subfigure (f) shows the comparison and change trend of the MAE metric in the remaining time prediction task (called MAE-rt) as the length of the prefix trace increases.

As can be seen from Fig(s). 4 and 5, the four metrics (i.e., accuracy, precision, recall, and f-score) of MTLFormer are slightly higher than STLFormer with the different lengths of the prefix trace, while the other two MAE metrics (i.e., MAE-nt and MAE-rt) are lower than the latter. As can be seen in Fig. 6, the precision and f-score of MTLFormer are slightly higher than STLFormer with the different length of the prefix trace, and the MAE in the two time prediction tasks (i.e., MAE-nt and MAE-rt) is lower than the latter, while the other metrics are not as good as the latter. As can be seen from Fig. 7, the recall of MTLFormer is slightly higher than STLFormer with the different length of the prefix trace, and the MAE in the two time prediction tasks (i.e., MAE-nt and MAE-rt) is lower than the latter, while the other metrics are not as good as the latter. Overall, MTLFormer has a good performance than STLFormer for the average

**TABLE 6.** The comparison in the remaining time prediction task (lower is better).

| Approach | MAE/day | | | | *Mean* |
|---|---|---|---|---|---|
| | Helpdesk | BPIC2012_A | BPIC2012_O | BPIC2012_W | |
| MTLFormer | **3.70** | **3.12** | **3.69** | **1.86** | **3.09** |
| STLFormer [11] | 4.04 | 4.72 | 5.28 | 5.03 | 4.77 |
| Method in Tax et al. [9] | 71.50 | 28.55 | 38.09 | 387.81 | 131.24 |
| Method in Camargo et al. (argmax) [24] | 11.15 | 11.90 | 19.78 | 32.03 | 18.72 |
| Method in Camargo et al. (random) [24] | 10.53 | 11.90 | 19.79 | 32.13 | 18.59 |
| Method in Navarin et al. [25] | 10.38 | 6.05 | 6.73 | 6.63 | 7.45 |
| Method in Francescomarino et al. [26] | 273.77 | 36.34 | 52.74 | 329.58 | 173.11 |

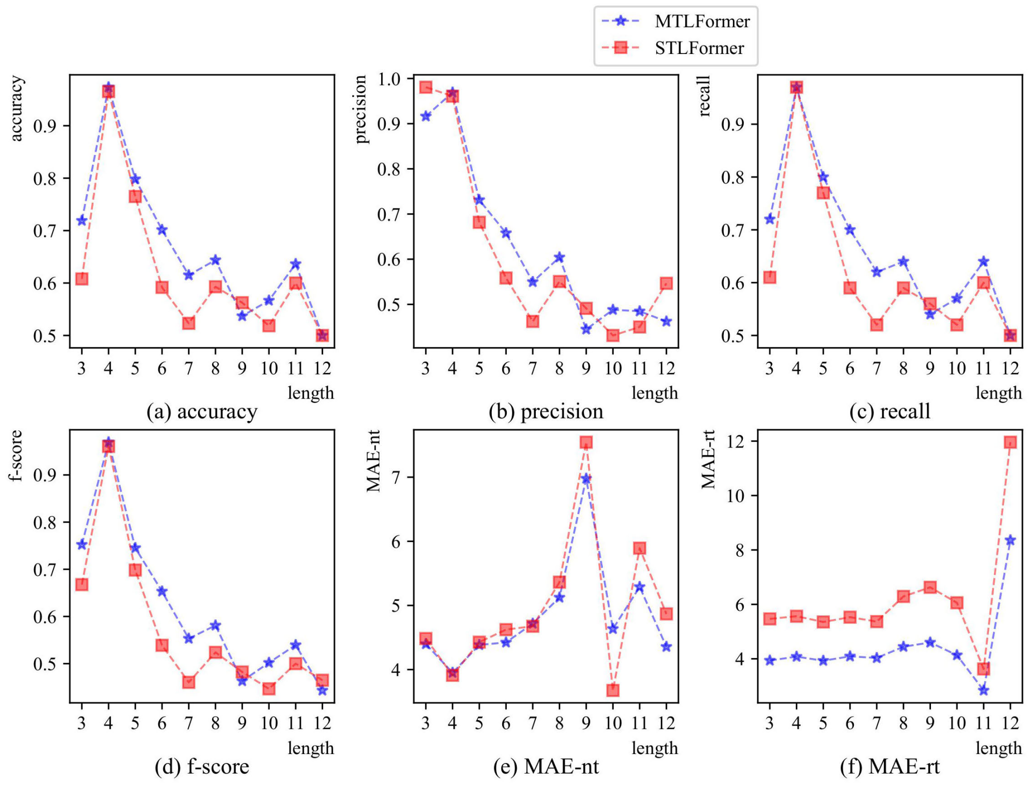

**FIGURE 4.** The prediction performance comparison of MTLFormer and STLFormer in different tasks on Helpdesk dataset.

**TABLE 7.** The comparison of model size for different approaches (1K=1000).

| Approach | #params | | | | *Mean* |
|---|---|---|---|---|---|
| | Helpdesk | BPIC2012_A | BPIC2012_O | BPIC2012_W | |
| MTLFormer | 114K | 114K | 116K | 124K | 117K |
| STLFormer [11] | 102K | 102K | 104K | 116K | 106K |

of all metrics but is also slightly inferior to STLFormer for individual metrics in individual datasets.

Besides, to compare the model size of the different approaches, we conduct a comprehensive analysis of the parameter counts for both MTLFormer and STLFormer models. This analysis allows for a more detailed comparison of the model sizes between the two approaches for different datasets. Specifically, we calculated each model's total number of learnable parameters, including the weights and biases. Table 7 shows the statistics of the number of parameters in the model trained by the two approaches for each dataset. Among them, the parameter number of STLFormer is calculated by the sum of the parameter number of the three tasks. As can be seen from the table, the model size (i.e., the number of parameters) of MTLFormer and STLFormer is of the same magnitude in all datasets, both of

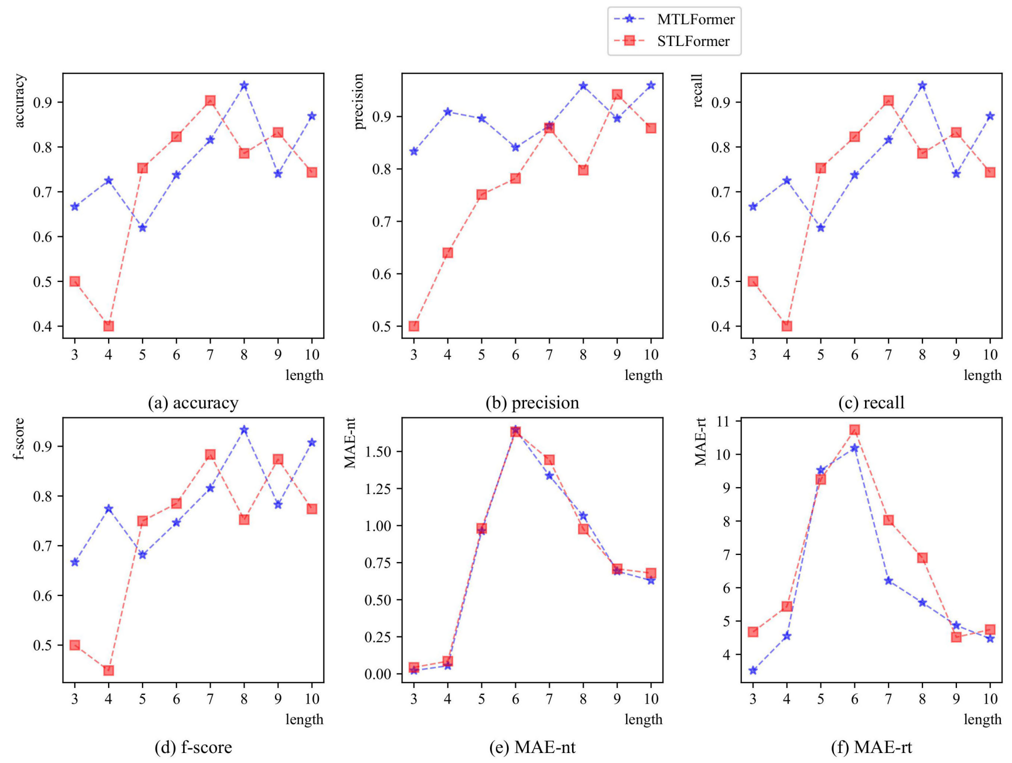

**FIGURE 5.** **The prediction performance comparison of MTLFormer and STLFormer in different tasks on BPIC2012_A dataset.**

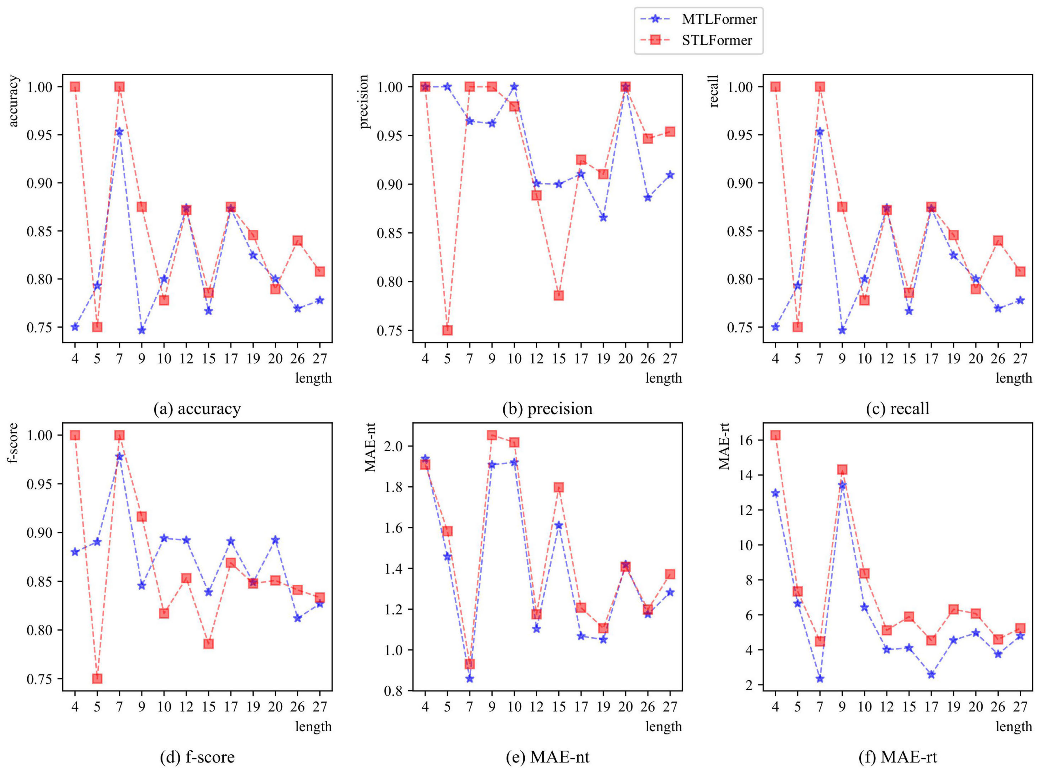

**FIGURE 6.** **The prediction performance comparison of MTLFormer and STLFormer in different tasks on BPIC2012_O dataset.**

which are more than 100K (1K=1000) parameters. In contrast, the number of parameters of the MTLFormer model is more than that of STLFormer on all datasets, with a difference of 11K parameters from the average value. This may

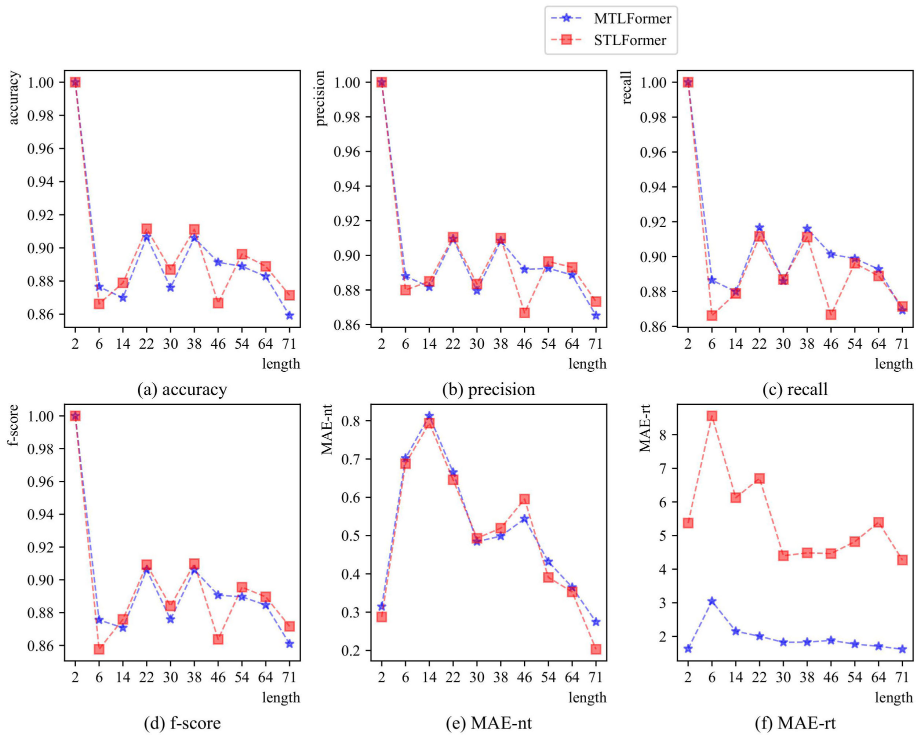

**FIGURE 7.** The prediction performance comparison of MTLFormer and STLFormer in different tasks on BPIC2012_W dataset.

be because MTLFormer has a certain amount of additional shared parameters between multiple tasks when training the model, compared with single-task training of STLFormer. However, this model size difference of parameter number is acceptable considering the current advanced hardware computing power.

## VI. CONCLUSION AND FUTURE WORK

This paper focuses on three prediction tasks in terms of the next activity, the next event time, and the remaining time in Predictive (business) Process Monitoring (PPM) and investigates how to improve the prediction performance further and reduce time complexity. Our proposed approach, MTLFormer, allows multiple tasks to learn concurrently, share, and migrate knowledge, and is built utilizing the Transformer network guided by multi-task learning. Because the hard-parameter sharing used in the construction of the prediction model allows the learning of latent relations between different tasks, MTLFormer can vastly reduce the time required for model training, achieving the goal of mutual learning between multiple tasks through parameter sharing between different tasks. Furthermore, the multi-head attention mechanism of the Transformer network overcomes the challenge of learning long-distance dependent features in process instance data, thereby improving the prediction performance. Extensive comparative experiments are comprehensively conducted for different tasks on different datasets, and the results demonstrate the effectiveness of MTLFormer for multiple tasks.

In future work, we intend to use soft-parameter sharing structures such as the multi-gate Mixture of Experts (mMoE) to implement multi-task learning and to investigate the application of other deep learning techniques in the field of predictive process monitoring. Moreover, we will study how predictive process monitoring tasks in distributed application scenarios can improve collaboration and performance between multiple heterogeneous tasks while enhancing privacy and reducing communication overhead in future work, inspired by [43]. Besides, we also intend to introduce active learning into predictive process monitoring tasks inspired by [44]. It is expected that with the help of active learning strategies, it can be implemented to allow the machine learning algorithm to choose the data it learns so that higher accuracy can be achieved with fewer training labels.

## REFERENCES

[1] W. Van Der Aalst, "Process mining: Overview and opportunities," *ACM Trans. Manage. Inf. Syst.*, vol. 3, no. 2, pp. 1–17, 2012.

[2] W. Kratsch, J. Manderscheid, M. Röglinger, and J. Seyfried, "Machine learning in business process monitoring: A comparison of deep learning and classical approaches used for outcome prediction," *Bus. Inf. Syst. Eng.*, vol. 63, no. 3, pp. 261–276, Jun. 2021.

[3] F. Taymouri, M. L. Rosa, S. Erfani, Z. D. Bozorgi, and I. Verenich, "Predictive business process monitoring via generative adversarial nets: The case of next event prediction," in *Proc. Int. Conf. Bus. Process Manage.* Cham, Switzerland: Springer, 2020, pp. 237–256.

[4] A. Appice, N. Di Mauro, and D. Malerba, "Leveraging shallow machine learning to predict business process behavior," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jul. 2019, pp. 184–188.

[5] I. Teinemaa, M. Dumas, M. L. Rosa, and F. M. Maggi, "Outcome-oriented predictive process monitoring: Review and benchmark," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 2, pp. 1–57, Apr. 2019.

[6] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "A multi-view deep learning approach for predictive business process monitoring," *IEEE Trans. Services Comput.*, vol. 15, no. 4, pp. 2382–2395, Jul. 2022.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[8] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," 2014, *arXiv:1409.2329*.

[9] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.* Cham, Switzerland: Springer, 2017, pp. 477–492.

[10] A. Vaswani, N. Shazeer, and N. Parmar, "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[11] Z. A. Bukhsh, A. Saeed, and R. M. Dijkman, "ProcessTransformer: Predictive business process monitoring with transformer network," 2021, *arXiv:2104.00721*.

[12] R. A. Caruana, "Multitask learning: A knowledge-based source of inductive bias," *Mach. Learn. Proc.*, vol. 10, no. 1, pp. 41–48, 1993.

[13] W. M. P. van der Aalst, M. H. Schonenberg, and M. Song, "Time prediction based on process mining," *Inf. Syst.*, vol. 36, no. 2, pp. 450–475, Apr. 2011.

[14] M. Polato, A. Sperduti, A. Burattin, and M. D. Leoni, "Time and activity sequence prediction of business process instances," *Computing*, vol. 100, no. 9, pp. 1005–1031, Sep. 2018.

[15] D. Breuker, H. Group, M. Matzner, P. Delfmann, and J. Becker, "Comprehensible predictive models for business processes," *MIS Quart.*, vol. 40, no. 4, pp. 1009–1034, Apr. 2016.

[16] G. T. Lakshmanan, D. Shamsi, Y. N. Doganata, M. Unuvar, and R. Khalaf, "A Markov prediction model for data-driven semi-structured business processes," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 97–126, Jan. 2015.

[17] R. Conforti, M. D. Leoni, M. L. Rosa, and W. M. P. V. D. Aalst, "Supporting risk-informed decisions during business process execution," in *Proc. 25th Int. Conf. Adv. Inf. Syst. Eng. (CAiSE)*, 2013, pp. 116–132.

[18] A. Leontjeva, R. Conforti, C. D. Francescomarino, M. Dumas, and F. M. Maggi, "Complex symbolic sequence encodings for predictive monitoring of business processes," in *Proc. Int. Conf. Bus. Process Manage.* Cham, Switzerland: Springer, 2016, pp. 297–313.

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[20] E. Rama-Maneiro, J. C. Vidal, and M. Lama, "Deep learning for predictive business process monitoring: Review and benchmark," *IEEE Trans. Services Comput.*, vol. 16, no. 1, pp. 739–756, Jan. 2023.

[21] J. Evermann, J. R. Rehse, and P. Fettke, "A deep learning approach for predicting process behaviour at runtime," in *Proc. Int. Conf. Bus. Process Manage.*, 2016, pp. 327–338.

[22] A. Shewalkar, D. Nyavanandi, and S. A. Ludwig, "Performance evaluation of deep neural networks applied to speech recognition: RNN, LSTM and GRU," *J. Artif. Intell. Soft Comput. Res.*, vol. 9, no. 4, pp. 235–245, Oct. 2019.

[23] I. Teinemaa, M. Dumas, A. Leontjeva, and F. M. Maggi, "Temporal stability in predictive process monitoring," *Data Mining Knowl. Discovery*, vol. 32, no. 5, pp. 1306–1338, Sep. 2018.

[24] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate LSTM models of business processes," in *Proc. Int. Conf. Bus. Process Manage.* Cham, Switzerland: Springer, 2019, pp. 286–302.

[25] N. Navarin, B. Vincenzi, M. Polato, and A. Sperduti, "LSTM networks for data-aware remaining time prediction of business process instances," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Nov. 2017, pp. 1–7.

[26] C. D. Francescomarino, C. Ghidini, F. M. Maggi, G. Petrucci, and A. Yeshchenko, "An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring," in *Proc. Int. Conf. Bus. Process Manage.* Cham, Switzerland: Springer, 2017, pp. 252–268.

[27] M. Hinkka, T. Lehto, and K. Heljanko, "Exploiting event log event attributes in RNN based prediction," in *Data-Driven Process Discovery and Analysis*. Cham, Switzerland: Springer, 2018, pp. 67–85.

[28] B. Wickramanayake, Z. He, C. Ouyang, C. Moreira, Y. Xu, and R. Sindhgatta, "Building interpretable models for business process prediction using shared and specialised attention mechanisms," *Knowl.-Based Syst.*, vol. 248, Jul. 2022, Art. no. 108773.

[29] N. D. Mauro, A. Appice, and T. Basile, "Activity prediction of business process instances with inception CNN models," in *Proc. Int. Conf. Italian Assoc. Artif. Intell.* Cham, Switzerland: Springer, 2019, pp. 348–361.

[30] V. Pasquadibisceglie, A. Appice, G. Castellano, and D. Malerba, "Using convolutional neural networks for predictive process analytics," in *Proc. Int. Conf. Process Mining (ICPM)*, Jun. 2019, pp. 129–136.

[31] A. Khan, H. Le, and K. Do, "DeepProcess: Supporting business process execution using a MANN-based recommender system," in *Proc. Int. Conf. Service-Oriented Comput.* Cham, Switzerland: Springer, 2021, pp. 19–33.

[32] J. Theis and H. Darabi, "Decay replay mining to predict next process events," *IEEE Access*, vol. 7, pp. 119787–119803, 2019.

[33] H. Chen, X. Fang, and H. Fang, "Multi-task prediction method of business process based on BERT and transfer learning," *Knowl.-Based Syst.*, vol. 254, Oct. 2022, Art. no. 109603.

[34] S. Ruder, "An overview of multi-task learning in deep neural networks," 2017, *arXiv:1706.05098*.

[35] D. Harris and S. Harris, *Digital Design and Computer Architecture*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, Aug. 2012.

[36] M. Lin, Q. Chen, and S. Yan, "Network in network," 2013, *arXiv:1312.4400*.

[37] H. Alaeddine and M. Jihene, "Deep residual network in network," *Comput. Intell. Neurosci.*, vol. 2021, pp. 1–9, Feb. 2021.

[38] R. L. Kumar, J. Kakarla, B. V. Isunuri, and M. Singh, "Multi-class brain tumor classification using residual network and global average pooling," *Multimedia Tools Appl.*, vol. 80, no. 9, pp. 13429–13438, Apr. 2021.

[39] J. Lei Ba, J. Ryan Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[40] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Netw.*, vol. 2, no. 6, pp. 459–473, Jan. 1989.

[41] D. H. Wolpert, "Stacked generalization," *Neural Netw.*, vol. 5, no. 2, pp. 241–259, Jan. 1992.

[42] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.

[43] S. Park and J. C. Ye, "Multi-task distributed learning using vision transformer with random patch permutation," *IEEE Trans. Med. Imag.*, vol. 42, no. 7, pp. 2091–2105, Jul. 2022.

[44] G. Rotman and R. Reichart, "Multi-task active learning for pre-trained transformer-based models," *Trans. Assoc. Comput. Linguistics*, vol. 10, pp. 1209–1228, Nov. 2022.

**JIAOJIAO WANG** received the Ph.D. degree in computer science from Hangzhou Dianzi University, Hangzhou, China, in 2019. She is currently a Research Assistant with the Institute of Intelligent Media Technology, Communication University of Zhejiang, China. Her current research interests include process data management and mining, predictive process monitoring, and conformance checking.

**JIAWEI HUANG** was born in Jingdezhen, Jiangxi, China, in 2001. He received the B.S. degree in software engineering from the Communication University of Zhejiang, China, in 2022. His current research interests include process data mining, predictive process monitoring, and machine learning.

**XIAOYU MA** received the Ph.D. degree in computer science from the Communication University of China, Beijing, China, in 2019. He is currently a Research Assistant with the Institute of Intelligent Media Technology, Communication University of Zhejiang, China. He has published seven articles and filed one patent. His research interests include image quality assessment, video compression, and medical image processing.

**YAQI WANG** received the Ph.D. degree from Hangzhou Dianzi University, China, in 2020. From 2019 to 2020, she was a Visiting Scholar with the Queen Mary University of London. She is currently a Lecturer with the Communication University of Zhejiang, China. She is dedicated to deep learning methods research in medical imaging and big data machine learning analysis. Her research interests include multimodal data fusion, bioinformatics AI processing, and machine learning methods in medical images.

**ZHONGJIN LI** received the Ph.D. degree from the Software Institute, Nanjing University, Nanjing, China, in 2017. He is currently a Lecturer with the School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. His research interests include cloud computing, workflow scheduling, and mobile edge computing.

**DINGGUO YU** was born in 1976. He received the Ph.D. degree in computer application technology from Tongji University, China, in 2011. He is currently a Professor and the Director of the Key Laboratory of Film and TV Media Technology of Zhejiang Province, China. His research interests include media fusion technology, big data, and artificial intelligence for media.