PGTNet: A Process Graph Transformer Network for Remaining Time Prediction of Business Process Instances

Keyvan Amiri Elyasi[1][0009−0007−3016−2392], Han van der
Aa[2][0000−0002−4200−4937], and Heiner Stuckenschmidt[1][0000−0002−0209−3859]

[1] Data and Web Science Group, University of Mannheim, Germany
{keyvan,heiner}@informatik.uni-mannheim.de
[2] Faculty of Computer Science, University of Vienna, Austria
han.van.der.aa@univie.ac.at

Abstract. We present PGTNet, an approach that transforms event logs
into graph datasets and leverages graph-oriented data for training Pro-
cess Graph Transformer Networks to predict the remaining time of busi-
ness process instances. PGTNet consistently outperforms state-of-the-art
deep learning approaches across a diverse range of 20 publicly available
real-world event logs. Notably, our approach is most promising for highly
complex processes, where existing deep learning approaches encounter
difficulties stemming from their limited ability to learn control-flow rela-
tionships among process activities and capture long-range dependencies.
PGTNet addresses these challenges, while also being able to consider
multiple process perspectives during the learning process.

Keywords: Predictive process monitoring · Remaining time prediction
· Deep learning · Graph Transformers.

1 Introduction

Predictive process monitoring (PPM) aims to forecast the future behaviour of
running business process instances, thereby enabling organizations to optimize
their resource allocation and planning [17], as well as take corrective actions [7].
An important task in PPM is remaining time prediction, which strives to accu-
rately predict the time until an active process instance will be completed. Precise
estimations for remaining time are crucial for avoiding deadline violations, opti-
mizing operational efficiency, and providing estimates to customers [13, 17].

A variety of approaches have been developed to tackle remaining time predic-
tion, with recent works primarily being based on deep learning architectures. In
this regard, approaches using deep neural networks are among the most promi-
nent ones [15]. However, the predictive accuracy of these networks leaves consid-
erable room for improvement. In particular, they face challenges when it comes
to capturing long-range dependencies [2] and other control-flow relationships
(such as loops and parallelism) between process activities [22], whereas they

also struggle to harness information from additional process perspectives, such as case and event attributes [13]. Other architectures can overcome some of these individual challenges. For instance, the Transformer architecture can learn long-range dependencies [2], graph neural networks (GNNs) can explicitly incorporate control-flow structures into the learning process [22], and LSTM (long short-term memory) architectures can be used to incorporate (parts of) the data perspective [13]. However, so far, no deep learning approach can effectively deal with all of these challenges simultaneously.

Therefore, this paper introduces PGTNet, a novel approach for remaining time prediction that can tackle all these challenges at once. Specifically, our approach converts event data into a graph-oriented representation, which allows us to subsequently employ a neural network based on the general, powerful, scalable (GPS) Graph Transformer architecture [16] to make predictions. Graph Transformers (GTs) have shown impressive performance in various graph regression tasks [4, 10, 16] and their theoretical expressive power closely aligns with our objectives: they can deal with multi-perspective data (covering various process perspectives) and can effectively capture long-range dependencies and recognize local control-flow structures. GTs achieve these latter benefits through a combination of local message-passing neural networks (MPNNs) [6] and a global attention mechanism [18]. They employ sparse message-passing within their GNN blocks to learn local control-flow relationships among process activities, while their Transformer blocks attend to all events in the running process instance to capture the global context.

We evaluated the effectiveness of PGTNet for remaining time prediction using 20 publicly available real event logs. Our experiments show that our approach outperforms current state-of-the-art deep learning approaches in terms of accuracy and earliness of predictions. We also investigated the relationship between process complexity and the performance of PGTNet, which revealed that our approach particularly achieved superior predictive performance (compared to existing approaches) for highly complex, flexible processes.

The structure of the paper is outlined as follows: Section 2 covers background and related work, Section 3 presents preliminary concepts, Section 4 introduces our proposed approach for remaining time prediction, Section 5 discusses the experimental setup and key findings, and finally, Section 6 summarizes our contributions and suggests future research directions.

2   Background and Related work

This section briefly discusses related work on remaining time prediction and provides more details on Graph Transformers.

Remaining time prediction. Various approaches have been proposed for remaining time prediction, encompassing process-aware approaches relying on transition systems, stochastic Petri Nets, and queuing models, along with machine learning-based approaches [20]. In recent years, approaches based on deep learning have emerged as the foremost methods for predicting remaining time [15].

These approaches use different neural network architectures such as LSTMs [13, 17], Transformers [2], and GNNs [3].

Vector embedding and feature vectors, constituting the data inputs for Transformers and LSTMs, face a challenge in directly integrating control-flow relationships into the learning process. To overcome this constraint, event logs can be converted into graph data, which then acts as input for training a Graph Neural Network (GNN) [22]. GNNs effectively incorporate control-flow structures by aligning the computation graph with input data. Nevertheless, they suffer from over-smoothing and over-squashing problems [10], sharing similarities with LSTMs in struggling to learn long-range dependencies [16]. Moreover, existing graph-based predictive models face limitations due to the expressive capacity of their graph-oriented data inputs. Current graph representations of business process instances, either focus solely on the control-flow perspective [7, 19] or conceptualize events as nodes [3, 22], leading to a linear graph structure that adversely impacts the performance of a downstream GNN.

Graph Transformers. Inspired by the successful application of self-attention mechanism in natural language processing, two distinct solutions have emerged to address the limitations of GNNs. The first approach unifies GNN and Transformer modules in a single architecture, while the second compresses the graph structure into positional (PE) and structural (SE) embeddings. These embeddings are then added to the input before feeding it into the Transformer network [16]. Collectively known as Graph Transformers (GTs), both solutions aim to overcome the limitations of GNNs, by enabling information propagation across the graph through full connectivity [10, 16]. GTs also possess greater expressive power compared to conventional Transformers, as they can incorporate local context using sparse information obtained from the graph structure [4].

Building upon this theoretical foundation, we propose to convert event logs into graph datasets to enable remaining time prediction using a Process Graph Transformer Network (PGTNet), as discussed in the remainder.

3   Preliminaries

This section presents essential concepts that will be used in the remainder.

Directed and attributed graphs. A directed graph $G = (V, \mathcal{E})$ is defined by a set of nodes V, and a set of ordered pairs of nodes $\mathcal{E} \subseteq V \times V$ called directed edges. For an edge $\epsilon_{ij} = (v_i, v_j) \in \mathcal{E}$ pointing from v_i to v_j, nodes $v_i \in V$ and $v_j \in V$ are called source and target nodes, respectively. Setting $n = |V|$, G's adjacency matrix A is a $n \times n$ matrix with $A_{ij} = 1$ if $\epsilon_{ij} \in \mathcal{E}$ and $A_{ij} = 0$ if $\epsilon_{ij} \notin \mathcal{E}$. An attributed graph is a graph that has node attributes in the form of a node feature matrix $X \in R^{n \times d}$, with $x_v \in R^d$ representing the feature vector of node v. It may also have edge attributes in the form of $\mathcal{Z} \in R^{m \times c}$, where $m = |\mathcal{E}|$ is the number of edges, and $z_{u,v} \in R^c$ denotes the feature vector of edge (u, v).

Events. Let Γ be the event universe (i.e., set of all possible event identifiers), $\mathcal{T}$ the time domain, $\mathcal{A}$ the finite set of process activities, $\mathcal{C}$ the set of all possible

case identifiers, and $V_1, V_2, ..., V_m$ sets of all possible values for data attributes $d_1, d_2, ..., d_m$. An event e is denoted by the tuple $e = (a, c, t, D)$. We assume that every event is characterized by mandatory properties, namely an activity identifier, a case identifier (i.e., the business process instance which the event belongs to), and a timestamp. Put it differently, there are $\Pi_A : \Gamma \to \mathcal{A}$, $\Pi_C : \Gamma \to \mathcal{C}$, and $\Pi_T : \Gamma \to \mathcal{T}$ functions that map an event e to an activity, a case, and a timestamp: $\Pi_A(e) = a$, $\Pi_C(e) = c$, $\Pi_T(e) = t$. All other attribute-value pairs (e.g., transition life-cycle, organizational attributes, etc.) which may be associated to the event are denoted by $D \equiv \{(d_1, v_1), (d_2, v_2), ..., (d_m, v_m)\}$ where $\forall v_i \in V_i$. Similar projection functions can be defined to extract values of specific attributes out of an event (e.g., $\Pi_{d_i}(e) = v_i$).

Traces, event log, event prefixes. Let Γ^* be the set of all possible sequences over Γ. A finite non-empty sequence of events $\sigma = < e_1, e_2, ..., e_n > \in \Gamma^*$, of length $|\sigma| = n$, is called a trace only and only if: 1) $\forall e_i, e_j \in \sigma : \pi_C(e_i) = \pi_C(e_j)$ (i.e., all events belong to the same case), 2) $\forall 1 \le i \le j \le n : \pi_T(e_i) \le \pi_T(e_j)$ (i.e., the events are ordered by their timestamps). Each trace thus represents the execution of one business process instance. An event log refers to a collection of traces and is denoted as $L \subseteq \Gamma^*$, where every event appears only once within the entirety of the log. For the remaining time prediction problem, we also need to define partial traces, which we do through event prefixes. An event prefix of length k consists of the first k events of a trace and is denoted by $hd^k(\sigma) = < e_1, ..., e_k >$, where $k \in [1, n-1]$ is a positive integer number.

Problem statement. Given an event log L of completed traces and an event prefix $hd^k(\sigma)$ of a trace $\sigma = < e_1, e_2, ..., e_n >$, where $\sigma \notin L$ is an unseen trace of the same process as the traces of L, then the remaining time prediction problem is defined as using the traces from L to learn a function θ_L that takes an unseen event prefix $hd^k(\sigma)$ and estimates the remaining time of the corresponding business process instance:

$$\theta_L(hd^k(\sigma)) \approx \pi_T(e_n) - \pi_T(e_k) \tag{1}$$

4   PGTNet for Remaining Time Prediction

To predict the remaining time of business process instances, we convert an event log into a graph dataset (see Section 4.1), and use it to train a predictive model (see Section 4.2). Once the model's parameters are learned, we can query the model to predict the remaining time of an active process instance based on its current partial trace.

4.1   Graph Representation of Event Prefixes

To train a predictive model, we first turn an event log L (consisting of traces of completed process instances) into a collection of event prefix-time tuples, with each tuple $(hd^k(\sigma), \pi_T(e_n) - \pi_T(e_k))$ capturing an event prefix of length k of a trace $\sigma \in L$, along with the time difference between σ's final timestamp and the

last timestamp of $hd^k(\sigma)$. Then, we establish a collection of directed attributed graphs $\mathcal{G}$, where each graph $G_{hd^k(\sigma)} \in \mathcal{G}$ encodes an event prefix $hd^k(\sigma)$, with its target attribute $\pi_T(e_n) - \pi_T(e_k)$. This transformation is illustrated in Figure 1, displaying a snapshot of an event log for case ID '27583' and the corresponding graph representation of the event prefix of a length of 6. The details of this transformation procedure are as follows:
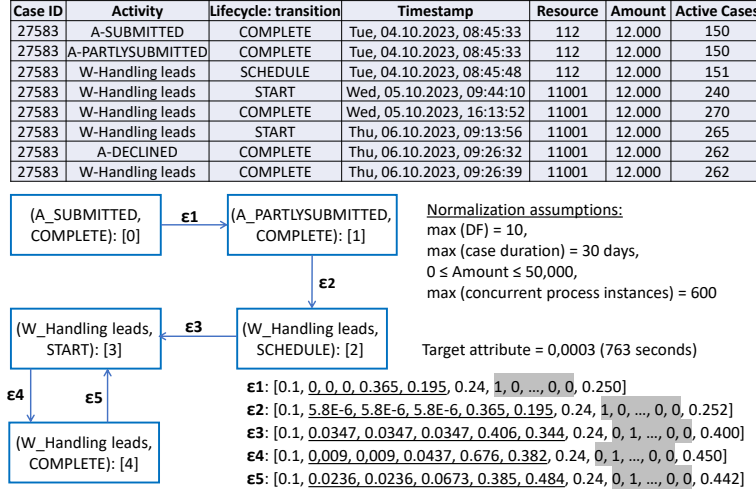
Case ID	Activity	Lifecycle: transition	Timestamp	Resource	Amount	Active Cases
27583	A-SUBMITTED	COMPLETE	Tue, 04.10.2023, 08:45:33	112	12.000	150
27583	A-PARTLYSUBMITTED	COMPLETE	Tue, 04.10.2023, 08:45:33	112	12.000	150
27583	W-Handling leads	SCHEDULE	Tue, 04.10.2023, 08:45:48	112	12.000	151
27583	W-Handling leads	START	Wed, 05.10.2023, 09:44:10	11001	12.000	240
27583	W-Handling leads	COMPLETE	Wed, 05.10.2023, 16:13:52	11001	12.000	270
27583	W-Handling leads	START	Thu, 06.10.2023, 09:13:56	11001	12.000	265
27583	A-DECLINED	COMPLETE	Thu, 06.10.2023, 09:26:32	11001	12.000	262
27583	W-Handling leads	COMPLETE	Thu, 06.10.2023, 09:26:39	11001	12.000	262

(A_SUBMITTED, COMPLETE): [0] →ε1→ (A_PARTLYSUBMITTED, COMPLETE): [1]

Normalization assumptions:
max (DF) = 10,
max (case duration) = 30 days,
0 ≤ Amount ≤ 50,000,
max (concurrent process instances) = 600

ε2

(W_Handling leads, START): [3] ←ε3← (W_Handling leads, SCHEDULE): [2]

Target attribute = 0,0003 (763 seconds)

ε4   ε5

(W_Handling leads, COMPLETE): [4]

ε1: [0.1, 0, 0, 0, 0.365, 0.195, 0.24, 1, 0, …, 0, 0, 0.250]
ε2: [0.1, 5.8E-6, 5.8E-6, 5.8E-6, 0.365, 0.195, 0.24, 1, 0, …, 0, 0, 0.252]
ε3: [0.1, 0.0347, 0.0347, 0.0347, 0.406, 0.344, 0.24, 0, 1, …, 0, 0, 0.400]
ε4: [0.1, 0,009, 0,009, 0.0437, 0.676, 0.382, 0.24, 0, 1, …, 0, 0, 0.450]
ε5: [0.1, 0.0236, 0.0236, 0.0673, 0.385, 0.484, 0.24, 0, 1, …, 0, 0, 0.442]

Fig. 1. Graph representation of an event prefix of length of 6, Case ID= '27583'.

Nodes. We create a node in $G_{hd^k(\sigma)}$ for each event class, i.e., each unique combination of an activity and life-cycle attribute (if any) contained in prefix $hd^k(\sigma)$. Each node has a numeric identifier, such as ('A-SUBMITTED', 'COMPLETE'): [0] in Figure 1. Using event classes as graph nodes results in graphs with relatively few nodes, even for lengthy prefixes.

Edges and edge weights. We create an edge in $G_{hd^k(\sigma)}$ for each directly-follows (DF) relation in $hd^k(\sigma)$, while the edge's weight indicates the number of its occurrences within $hd^k(\sigma)$. These edge weights are normalized by the maximum number of occurrences of any directly-follows relation within the available training prefixes. In Figure 1, this normalization coefficient (`max(DF)`) is equal to 10, resulting in edge weights of 0.1 (the first feature in the edge's vector).

Edge features. To enhance the expressive capacity of the graph representation, we incorporate additional features into the edge feature vector:
- We use five different temporal features per edge. These include the total duration (t_1) and duration of the last occurrence (t_2) of the DF relation represented by the edge. Similar to other works [17], we also incorporate distances between the timestamp of the target node and the start of the

case (t_3), start of the day (t_4), and start of the week (t_5) for the latest occurrence of the DF relation. While t_1, t_2, and t_3 are normalized by the largest case duration in the training data, t_4 and t_5 are normalized by the duration of days and weeks, respectively. In Figure 1, the temporal features are underlined in the feature vectors of the edges.

- We encode case attributes and event attributes of the target node (for the last occurrence of the DF relation in $hd^k(\sigma)$). For numerical and categorical attributes, we use min-max normalization and one-hot encoding, respectively. In Figure 1, one-hot encoding of the categorical attribute 'Resource' is highlighted with a grey shadow, while the feature before this gray part (always 0.24 in the example) captures the case attribute 'Amount'.
- To account for the overall workload of the process at a given time, we capture the number of active cases at the timestamp of the target node (for the last occurrence of the DF relation). This feature is normalized by the maximum number of concurrent process instances observed in the training data.

Note that we encode this information as edge features, rather than on the nodes, in order to preserve the simplicity of the node semantics. In this way, PGTNet can also deal with event logs with a large number of event classes, achieved by employing an embedding layer.

Target attribute. For each graph $G_{hd^k(\sigma)} \in \mathcal{G}$, its target attribute $\pi_T(e_n) - \pi_T(e_k)$ is normalized by the longest case duration in the training data.

4.2   Training PGTNet to Predict Remaining Time

Once an event log is converted into a graph dataset, it can be used to train PGTNet to learn function θ_L in Equation 1 in an end-to-end manner. We specifically approach the remaining time prediction problem as a graph regression task, using L1-Loss (mean absolute error between predictions and ground truth remaining times). Model training employs the backpropagation algorithm to iteratively minimize the loss function. For this, we adopt the GPS Graph Transformer recipe [16] as the underlying architecture of PGTNet.

PGTNet architecture. PGTNet's architecture comprises embedding and processing modules, as shown in Figure 2.

Embedding modules have two main functionalities:
- They map node and edge features into continuous spaces. To ensure that similar event classes are closer in the embedding space, an embedding layer is used to map integer node features into a continuous space. We use fully-connected layer(s) to compress edge features into the same hidden dimension and address the challenges arising from high-dimensional data attributes.
- They compress the graph structure into multiple positional and structural encodings (PE/SE), and seamlessly incorporate these PE/SEs into node and edge features [16]. This integration is achieved through diverse PE/SE initialization strategies and the utilization of several learnable modules, including MLPs (multi-layer perceptron) and batch normalization layers, as illustrated in Figure 2.
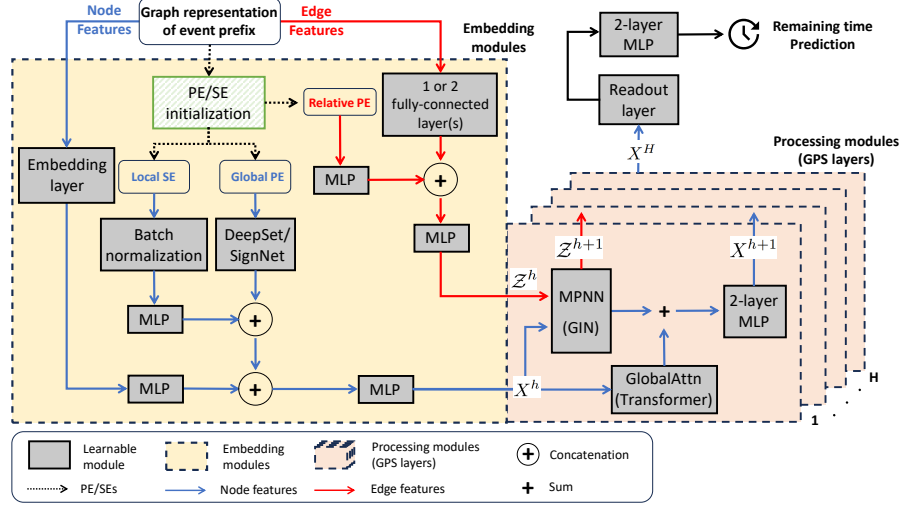
Fig. 2. PGTNet architecture: based on the GPS Graph Transformer recipe [16]. Paths to process node and edge features are specified by blue and red colors, respectively.

Assuming a graph $G = (V, E)$ with adjacency matrix A, node feature matrix $\tilde{X}^0$, and edge feature matrix $\tilde{\mathcal{Z}}^0$, embedding modules are described by Equation 2.

$$X^0, \mathcal{Z}^0 = \mathbf{f}_{PE/SE}(\tilde{X}^0, \tilde{\mathcal{Z}}^0, A) \tag{2}$$

Graph Transformers vary in their choice of function $\mathbf{f}_{PE/SE}$, which is often a neural network with learnable parameters. Further insights into the PE/SEs is provided in the subsequent discussion of the design space for PGTNet.

Processing modules consist of hybrid layers combining MPNN and Transformer blocks. Each of the H layers, also referred to as GPS layers, computes a hidden representation of nodes and edges (X^{h+1}, $\mathcal{Z}^{h+1}$) based on the node and edge embedding from the previous layer (X^h, $\mathcal{Z}^h$), and the adjacency matrix A, as summarized in Equations 3a, 3b, 3c, and 3d below. Parallel computations in MPNN and Transformer blocks aim to strike a balance between local message passing and global attention mechanisms and resolve over-smoothing and over-squashing problems [16].

$$X^{h+1}, \mathcal{Z}^{h+1} = GPS^h(X^h, \mathcal{Z}^h, A) \quad \forall h \in \{0, 1, ..., H-1\} \tag{3a}$$

$$X_M^{h+1}, \mathcal{Z}^{h+1} = MPNN^h(X^h, \mathcal{Z}^h, A) \quad \forall h \in \{0, 1, ..., H-1\} \tag{3b}$$

$$X_T^{h+1} = GlobalAttn^h(X^h) \quad \forall h \in \{0, 1, ..., H-1\} \tag{3c}$$

$$X^{h+1} = MLP^h(X_M^{h+1} + X_T^{h+1}) \quad \forall h \in \{0, 1, ..., H-1\} \tag{3d}$$

After obtaining node embeddings X^H from the last GPS layer, they are aggregated in the readout layer to derive a graph-level representation that is subsequently fed into a 2-layer MLP to predict the remaining time (see Equation 4). Since a simple permutation invariant function (e.g., mean or sum) is used for the readout function in Equation 4, the readout layer accommodates varying node counts. Therefore, PGTNet avoids the need for zero-padding [2,13] and its computational overhead.

$$y_G = MLP(readout(X^H)) \qquad (4)$$

Note that edge features are solely processed by MPNN blocks and are not utilized by Transformer blocks or in obtaining the graph-level representation.

Design space for PGTNet. The modular design of the GPS Graph Transformer recipe offers flexibility in choosing various types of positional/structural encodings (PE/SEs) and MPNN/Transformer blocks.

PE/SEs aim to enhance positional encoding for Transformer blocks [10], and enable GNN blocks to be more expressive [4]. The compression of graph structure into PE/SEs can be achieved through the utilization of various initialization strategies (PE/SE initialization in Figure 2). Notably, Laplacian eigenvector encodings (LapPE) [10] furnishes node embedding with information about the overall position of the event class within the event prefix (global PE), while it enhances edge embedding with information on distance and directional relationships between nodes (relative PE). Random-walk structural encoding (RWSE) [4] incorporates local SE into node features, facilitating the recognition of cyclic control-flow patterns among event classes. Graphormer employs a combination of centrality encoding (local SE) and edge encoding (relative PE) to enhance both node and edge features [24].

Additionally, a range of learnable modules for processing PE/SEs can be integrated into PGTNet as highlighted in [16]. These design options include simple MLPs as well as more advanced networks such as DeepSet [25] and SignNet [11]. Lastly, while it is possible to use various MPNN and global attention blocks within each GPS layer [16], we exclusively used the graph isomorphism network (GIN) [9] and conventional transformer architecture [18]. Further details regarding our policy for designing PGTNet are elaborated upon in Section 5.1.

5 Evaluation

This section presents the experiments used to evaluate the performance of PGT-Net for remaining time prediction. Table 1 summarizes the characteristics of the 20 publicly available event logs used as a basis for this. In the remainder, Section 5.1 describes the experimental setup, followed by the results in Section 5.2. Our employed implementation and additional results are available in our project's public repository.[3]

---

[3] https://github.com/keyvan-amiri/PGTNet

Table 1. Characteristics of the employed event logs (time-related attributes in days).

Event log	Cases	Events	Event Classes	Variants	Case length		Case duration	
					Avg.	Max	Avg.	Max
Env.permit	1434	8577	27	116	5.98	25	5.4	275.8
Helpdesk	4580	21 348	14	226	4.66	15	40.9	60.0
BPIC12	13 087	262 200	36	4366	20.04	175	8.6	137.2
BPIC12W	9658	170 107	19	2643	17.61	156	11.7	137.2
BPIC12C	13 087	164 506	23	4336	12.57	96	8.6	91.5
BPIC12CW	9658	72 413	6	2263	7.50	74	11.4	91.0
BPIC12O	5015	31 244	7	168	6.23	30	17.2	89.6
BPIC12A	13 087	60 849	10	17	4.65	8	8.1	91.5
BPIC20I	6449	72 151	34	753	11.19	27	86.5	742.0
BPIC20D	10 500	56 437	17	99	5.37	24	11.5	469.2
Sepsis	1050	15 214	16	846	14.49	185	28.5	422.3
Hospital	100 000	451 359	18	1020	4.51	217	127.2	1035.4
BPIC15-1	1199	52 217	398	1170	43.55	101	95.9	1486.0
BPIC15-2	832	44 354	410	828	53.31	131	160.3	1326.0
BPIC15-3	1409	59 681	383	1349	42.36	123	62.2	1512.0
BPIC15-4	1053	47 293	356	1049	44.91	115	116.9	927.0
BPIC15-5	1156	59 083	389	1153	51.11	153	98.0	1344.0
BPIC13I	7554	65 533	13	2278	8.68	123	12.1	771.4
BPIC13C	1487	6660	7	327	4.48	35	178.9	2254.0
Traffic fines	150 370	561 470	11	231	3.73	20	341.6	4372.0

5.1   Experimental Setup

Data preprocessing. We filter out traces with fewer than 3 events, since our approach requires an event prefix of at least length 2 to make a prediction. Aside from that, we do not apply any preprocessing to the available event logs.

Benchmark approaches. We compare our approach against four others:
- *DUMMY*: A simple baseline that predicts the average remaining time of all training prefixes with the same length k as a given prefix.
- *DALSTM* [13]: An LSTM-based approach that was recently shown to have superior results among LSTMs used for remaining time prediction [15].
- *ProcessTransformer* [2]: A transformer-based approach designed to overcome LSTM's limitations in capturing long-range dependencies.
- *GGNN* [3]: An approach that utilizes gated graph neural networks to incorporate control-flow relationships into the learning process. It employs gated recurrent unit (GRU) within its MPNN layers, enabling the learning of the sequential nature of graph nodes.

Data split. There is no consensus on the data split for predictive process monitoring. Some papers employed a chronological holdout split [2, 17], while others opted for cross-validation [5, 14, 15]. The holdout split maintains chronological order but may introduce instability due to end-of-dataset bias. To avoid such instability, additional data preprocessing, as suggested by [23], is necessary.

As our benchmark approaches were not trained with such preprocessing, we avoided additional steps and chose a 5-fold cross-validation data split (CV=5) to enhance model's robustness against the end-of-dataset bias. We randomly partition the dataset into 5 folds where each fold serves as a test set once, and the remaining four folds are used as training and validation sets. For completeness, we also report results obtained using holdout data splits in our supplementary GitHub repository.

Prefix establishment. We turn the three sets of traces (training, validation, and test) into three sets of event prefixes by taking the event prefixes for each length $1 < k < |\sigma|$ per trace. In contrast to [2, 13], we excluded event prefixes of length 1 because a minimum of two events is required to form the graph representation of an event prefix. Moreover, similar to [13], we excluded complete prefixes (i.e., $k = |\sigma|$) because predicting the remaining time for such prefixes often lacks practical value.

Training setup and configuration choices. We used the AdamW optimizer [12] and cosine with warm-up learning rate scheduling [8] to train our model. Training spanned 600 epochs, including 50 warm-up epochs, with a base learning rate of 0.001 and weight decay of e^{-5}. We used a batch size of 128, with occasional adjustments for specific event logs. Similarly, the number of training epochs is adjusted to account for variations in validation loss behaviours across different event logs. The key configuration choices in our experiments include:

- *PE/SE modules:* LapPE+RWSE [4, 10] serves as the default module, occasionally substituted by Graphormer [24]. By default, DeepSet [25] processes PE/SEs, with SignNet [11] replacing LapPE+RWSE and DeepSet in some experiments. PE/SEs are tested in two sizes: 8 and 16.
- *Embedding modules:* Nodes and edges use an embedding size of 64. Edge features are compressed using two fully-connected layers, though in some experiments we opt for a single layer.
- *Processing modules:* comprising 5 GPS layers (GIN + Transformer) [16] with 8 heads, utilizing a dropout of 0.0 for MPNN blocks and 0.5 for Transformer blocks. In some experiments, we used 10 GPS layers with 4 heads instead, while in others we applied a dropout of 0.2 for MPNN blocks.
- *Readout layer:* Mean pooling is the default configuration, occasionally replaced by sum pooling.

Our focus is on demonstrating PGTNet's applicability for remaining time prediction rather than an exhaustive hyperparameter search. Therefore, we evaluated a limited set of configurations per log, selecting the best based on validation loss.

Evaluation metrics. We used Mean Absolute Error (MAE) to measure prediction accuracy. Since we are interested in models that not only have smaller MAE but also can make accurate predictions earlier, allowing more time for corrective actions, we used the method proposed in [17], which evaluates MAE across different event prefix lengths. This approach provides insights into the predictive performance of the model as more events arrive.

5.2   Results

Overall results. Table 2 summarizes the experimental results for the 20 event logs, providing the average and standard deviations of the MAEs obtained over experiments with three distinct, random seeds for training and evaluation. The table shows that our approach consistently outperforms the benchmark approaches across the 20 event logs, yielding an average MAE of 12.92, compared to 24.63 for the next best approach (GGNN).

Table 2. Mean Absolute Error for remaining time prediction (MAE: in days).

Event log	DUMMY	DALSTM	Process Transformer	GGNN	PGTNet
Env.permit	5.21	3.36 ± 0.04	4.26 ± 0.04	3.52 ± 0.02	$\mathbf{2.72 \pm 0.08}$
Helpdesk	9.15	8.22 ± 0.23	6.33 ± 0.01	6.21 ± 0.04	$\mathbf{4.11 \pm 0.04}$
BPIC12	9.03	9.34 ± 0.41	7.11 ± 0.02	4.78 ± 0.01	$\mathbf{2.31 \pm 0.19}$
BPIC12W	9.16	8.22 ± 0.06	7.40 ± 0.01	5.12 ± 0.02	$\mathbf{2.70 \pm 0.01}$
BPIC12C	8.92	8.21 ± 0.27	6.86 ± 0.01	5.32 ± 0.01	$\mathbf{2.77 \pm 0.02}$
BPIC12CW	9.17	8.04 ± 0.09	7.46 ± 0.01	6.99 ± 0.01	$\mathbf{5.07 \pm 0.03}$
BPIC12O	8.39	8.21 ± 0.09	7.29 ± 0.01	6.93 ± 0.04	$\mathbf{5.57 \pm 0.01}$
BPIC12A	8.17	7.62 ± 0.03	7.79 ± 0.01	7.48 ± 0.01	$\mathbf{7.38 \pm 0.01}$
BPIC20I	27.20	20.43 ± 0.39	17.06 ± 0.11	15.67 ± 0.04	$\mathbf{7.67 \pm 0.19}$
BPIC20D	4.33	4.15 ± 0.12	3.65 ± 0.01	3.25 ± 0.01	$\mathbf{3.10 \pm 0.01}$
Sepsis	41.12	25.21 ± 0.66	34.77 ± 0.18	19.44 ± 0.05	$\mathbf{16.48 \pm 0.19}$
Hospital	59.41	43.66 ± 0.10	47.00 ± 0.07	41.84 ± 0.06	$\mathbf{35.68 \pm 0.03}$
BPIC15-1	50.22	36.48 ± 2.69	31.01 ± 0.36	16.77 ± 0.01	$\mathbf{1.76 \pm 0.06}$
BPIC15-2	83.11	63.66 ± 2.36	44.04 ± 0.48	20.76 ± 0.05	$\mathbf{3.02 \pm 0.07}$
BPIC15-3	28.76	17.69 ± 1.16	15.23 ± 0.23	7.06 ± 0.03	$\mathbf{1.54 \pm 0.23}$
BPIC15-4	56.75	53.33 ± 2.63	34.40 ± 0.42	17.97 ± 0.03	$\mathbf{1.65 \pm 0.06}$
BPIC15-5	45.97	42.89 ± 3.08	27.76 ± 0.28	13.61 ± 0.08	$\mathbf{1.61 \pm 0.01}$
BPIC13I	16.18	7.60 ± 0.45	13.54 ± 0.04	11.99 ± 0.04	$\mathbf{2.23 \pm 0.05}$
BPIC13C	152.93	91.82 ± 1.48	127.01 ± 0.85	123.28 ± 0.53	$\mathbf{37.44 \pm 1.49}$
Traffic fines	196.26	187.41 ± 0.53	187.08 ± 0.11	154.56 ± 0.19	$\mathbf{113.53 \pm 0.12}$
Average	41.47	32.78 ± 0.84	31.85 ± 0.16	24.63 ± 0.06	$\mathbf{12.92 \pm 0.14}$

Next to these absolute MAE scores, we also computed the relative MAE (i.e., the MAE divided by the average case duration per log) to account for differences in the cycle times across event logs. Using these relative scores, we can visualize the accuracy improvements across different logs, as done in Figure 3 (for clarity, we aggregated the results of logs that stem from the same BPI collection, e.g., averaging the results of all BPIC15 logs).

PGTNet also excels in earliness of predictions, achieving lower MAE for most prefix lengths across all event logs. We illustrate some of these results in Figure 4, which depicts MAE trends for BPIC15-1, BPIC12, BPIC13I, Sepsis, Helpdesk, and BPIC12A logs at various prefix lengths. These six logs are representative of major MAE trends that we observed across the 20 event logs (the remaining
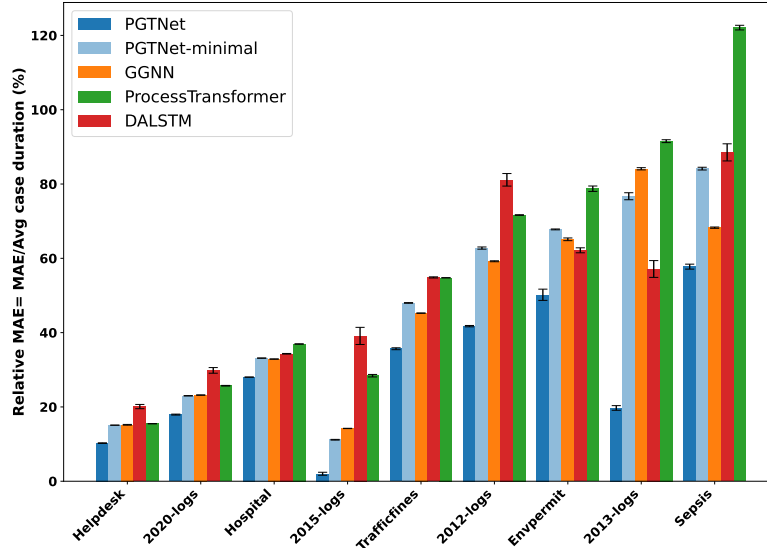
Fig. 3. Remaining time prediction accuracy in terms of relative MAE (in percentage).

figures are available in our repository). To improve understandability, we exclude uncommonly long prefixes from the visualization, so that Figure 4 focuses on prefixes up to a length that corresponds to 90% of all prefixes in each dataset.

Summarizing the overall results in terms of MAE, relative MAE, and prediction earliness, we obtain the following main insights:

– In 11 out of 20 logs, PGTNet achieves an MAE improvement of over 50% compared to the best baseline approach. For five BPIC15 and two BPIC13 logs, MAE trends closely mirror those of BPIC15-1 and BPIC13I as depicted in Figure 4. In BPIC12, BPIC12W, BPIC12C, and BPIC20I, the MAE trends closely follow those observed in BPIC12. In BPIC15 and BPIC12 logs, ProcessTransformer is the best baseline approach for short prefixes, whereas GGNN exhibits competitive performance only after execution of a substantial number of events. Notably, in the BPICS13 event logs, DALSTM outperforms other baseline approaches.
– In another 7 logs, PGTNet exhibits considerable improvement in MAE (15% to 35%), with Traffic fines, Env.permit and BPIC12O showing similar MAE trends to Helpdesk log in Figure 4. MAE trends for BPIC12CW resemble those of other BPIC12 logs in the first group. Notably, the Sepsis log shows a distinct MAE trend, where GGNN achieves comparable results to PGTNet for most prefix lengths.
– For BPIC20D and BPIC12A, the improvement is more modest, with the latter case yielding nearly identical MAEs for all prefix lengths (see Figure 4).
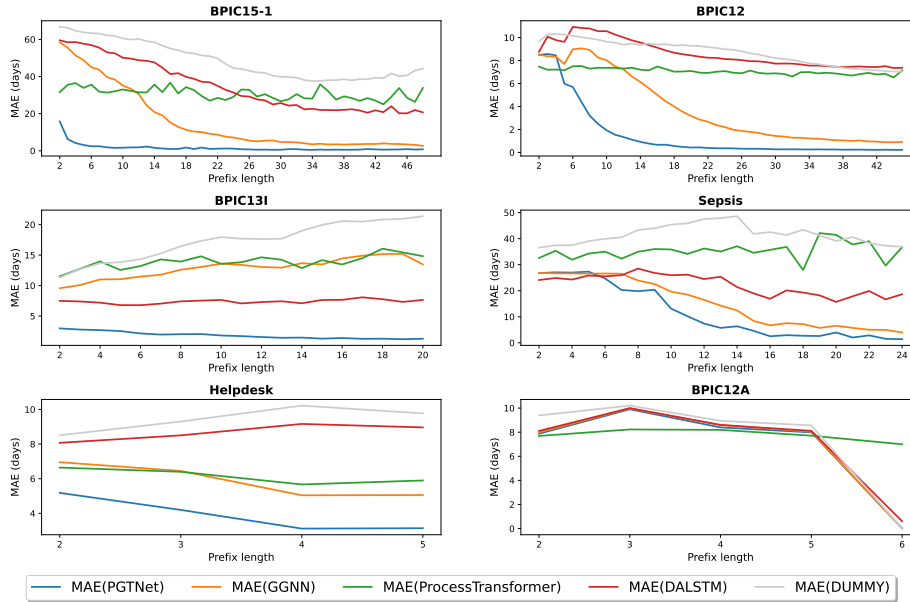
Fig. 4. MAE over different prefix lengths (selected event logs).

Ablation study. The remarkable performance of PGTNet can be attributed to a synergy between the expressive capacity of the employed architecture and the incorporation of diverse process perspectives into the graph representation of event prefixes. To distinguish the impacts of these factors, we conducted an ablation study for which we trained a minimal PGTNet model, relying solely on edge weights (i.e., control-flow) and temporal features, thus omitting data attributes from consideration. We used identical hyperparameters and configurations as was done for the complete PGTNet model. Hence, the ablation study establishes a lower boundary for contribution of the PGTNet's architecture.

Our experiments reveal that the minimal PGTNet model consistently outperforms ProcessTransformer in terms of MAE (see Figure 3). This underscores PGTNet's capabilities in capturing both local and global contexts, which is advantageous for predicting remaining time. However, the predictive performance of the minimal PGTNet is comparable to that of the GGNN approach, suggesting that learning from local control-flow structures in MPNN blocks (done by GGNN) holds greater significance than capturing long-range dependencies (as done by ProcessTransformer). This observation is further supported by the overall results presented in Table 2, where GGNN outperforms DALSTM and ProcessTransformer in all event logs, except for two BPIC13 logs. Additionally, the contribution of the architecture and the incorporation of extra data attributes varies across different event logs. While the PGTNet's architecture plays a decisive role for logs such as BPIC15 and BPIC20I, the improvements in MAE for logs such as BPICS12 and BPICS13 is primarily due to the incorpora-

tion of additional features. Further details regarding our ablation study can be found in our repository.

Impact of process complexity. The MAE improvement achieved by PGT-Net varies significantly across different event logs. Investigating these variations, we correlated process complexity metrics from [1, 21] with MAE improvements achieved by PGTNet. Notably, our approach outperforms alternatives when the number of process variants increases rapidly with respect to the number of cases. This trend extends to other *variation metrics*, including *'structure'* (average distinct activities per case) and *'level of detail'* (number of acyclic paths in the transition matrix) [1]. In terms of *size metrics*, PGTNet exhibits superior performance with increasing average trace length and/or number of distinct activities. The most significant positive correlation is observed for *'normalized sequence entropy'*, a graph entropy metric adept at capturing both variation and size complexity [1].

This reveals that PGTNet excels in highly flexible and complex processes, where benchmark approaches may overlook sparse but meaningful directly-follows relations among activities. The graph representation, detailed in Section 4.1, converts different process variants into non-isomorphic graphs with varying nodes, connectivity structures, and edge weights. Graph isomorphism network (GIN) modules, renowned for distinguishing non-isomorphic graphs [8], process this graph-oriented data. Simultaneously, Transformer blocks capture long-range dependencies. In complex processes like BPIC15, GIN blocks benefit from diverse set of non-isomorphic graphs available for learning, while Transformer blocks leverage insightful PE/SEs, thus providing a synergy that results in a remarkable reduction in MAE.

Training, and inference time. We conducted experiments using an Nvidia RTX A6000 GPU, with training and inference times detailed in Table 3. For training time, we computed the sum of training time for all cross-validation data splits and then averaged these times across all 20 event logs. Regarding average inference time, we compute the time to infer the remaining time per event prefix and report the average inference time across all event logs.

In terms of training time, DALSTM and ProcessTransformer, which use shallow neural networks (either with 2 LSTM layers or 1 self-attention layer), can be trained an order of magnitude faster than the graph-based approaches, though PGTNet is still trained faster than GGNN (12.93 vs. 18.56 hours). We see a similar trend in terms of inference times, though essentially all approaches are reasonably fast here, with PGTNet being the slowest with an average time just below 3 miliseconds.

Table 3. Average training and inference time for remaining time prediction.

Time	DALSTM	Process Transformer	GGNN	PGTNet
Training time (hours)	1.68	1.64	18.56	12.93
Inference time (miliseconds)	0.51	0.14	2.14	2.96

6   Conclusion and Future Work

This paper introduces a novel approach employing Process Graph Transformer Networks (PGTNet) to predict the remaining time of running business process instances. Our approach consists of a data transformation from an event log to a graph dataset, and training a neural network based on the GPS Graph Transformer recipe [16]. Our graph representation of event prefixes incorporates multiple process perspectives and also enables integration of control-flow relationships among activities into the learning process. This graph-oriented data input is subsequently processed by PGTNet, which strikes a balance between learning from local contexts and long-range dependencies.

Through experiments conducted on 20 real-world datasets, our results demonstrate the superior accuracy and earliness of predictions achieved by PGTNet compared to the existing deep learning approaches. Notably, our approach exhibits exceptional performance for highly flexible and complex processes, where the performance of LSTM, Transformer and GGNN architectures falls short.

While originally designed for predicting remaining times, our approach has the potential to learn high-level event prefix representations, rendering it applicable to other tasks, including next activity prediction and process outcome prediction. In future research, we therefore aim to apply PGTNet for these tasks, whereas we also aim to improve the predictive accuracy of our approach by investigating the potential of multi-task learning and exploring different positional and structural embeddings, as well as varying graph representations. Finally, we aim to extend our approach to also be applicable to object-centric event logs.

Reproducibility: Our source code and all evaluation results are accessible in our repository: https://github.com/keyvan-amiri/PGTNet

References

1. Augusto, A., Mendling, J., Vidgof, M., Wurm, B.: The connection between process complexity of event sequences and models discovered by process mining. Information Sciences **598**, 196–215 (2022)
2. Bukhsh, Z.A., Saeed, A., Dijkman, R.M.: ProcessTransformer: Predictive Business Process Monitoring with Transformer Network (2021), arXiv:2104.00721 [cs]
3. Duong, L.T., Travé-Massuyès, L., Subias, A., Merle, C.: Remaining cycle time prediction with Graph Neural Networks for Predictive Process Monitoring. In: International Conference on Machine Learning Technologies (ICMLT). ACM (2023)
4. Dwivedi, V.P., Luu, A.T., Laurent, T., Bengio, Y., Bresson, X.: Graph Neural Networks with Learnable Structural and Positional Representations (2022), arXiv:2110.07875 [cs]
5. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017)
6. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 1263–1272. PMLR (2017)

7. Harl, M., Weinzierl, S., Stierle, M., Matzner, M.: Explainable predictive business process monitoring using gated graph neural networks. Taylor and Francis (2020)
8. He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., Li, M.: Bag of tricks for image classification with convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
9. Hu, W., Liu, B., Gomes, J., Zitnik, M., Liang, P., Pande, V., Leskovec, J.: Strategies for Pre-training Graph Neural Networks (2020), arXiv:1905.12265 [cs, stat]
10. Kreuzer, D., Beaini, D., Hamilton, W.L., Létourneau, V., Tossou, P.: Rethinking Graph Transformers with Spectral Attention (2021), arXiv:2106.03893 [cs]
11. Lim, D., Robinson, J., Zhao, L., Smidt, T., Sra, S., Maron, H., Jegelka, S.: Sign and Basis Invariant Networks for Spectral Graph Representation Learning (2022), arXiv:2202.13013 [cs, stat]
12. Loshchilov, I., Hutter, F.: Decoupled Weight Decay Regularization (2019), arXiv:1711.05101 [cs, math]
13. Navarin, N., Vincenzi, B., Polato, M., Sperduti, A.: LSTM networks for data-aware remaining time prediction of business process instances. In: 2017 IEEE Symposium series on Computational Intelligence (SSCI). pp. 1–7 (2017)
14. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Data-aware remaining time prediction of business process instances. In: 2014 International Joint Conference on Neural Networks (IJCNN). pp. 816–823 (2014)
15. Rama-Maneiro, E., Vidal, J.C., Lama, M.: Deep Learning for Predictive Business Process Monitoring: Review and Benchmark. IEEE Transactions on Services Computing **16**(1), 739–756 (2023)
16. Rampášek, L., Galkin, M., Dwivedi, V.P., Luu, A.T., Wolf, G., Beaini, D.: Recipe for a General, Powerful, Scalable Graph Transformer. NeurIPS **35**, 14501–14515 (2022)
17. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive Business Process Monitoring with LSTM Neural Networks. In: Advanced Information Systems Engineering. pp. 477–492. Springer International Publishing, Cham (2017)
18. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is All you Need. In: NeurIPS. vol. 30. Curran Associates, Inc. (2017)
19. Venugopal, I., Töllich, J., Fairbank, M., Scherp, A.: A Comparison of Deep-Learning Methods for Analysing and Predicting Business Processes. In: 2021 International Joint Conference on Neural Networks (IJCNN). pp. 1–8 (2021)
20. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Teinemaa, I.: Survey and Cross-benchmark Comparison of Remaining Time Prediction Methods in Business Process Monitoring. ACM Trans. on Intelligent Systems and Technology **10**(4), 34:1–34:34 (2019)
21. Vidgof, M., Wurm, B., Mendling, J.: The Impact of Process Complexity on Process Performance: A Study using Event Log Data (2023), arXiv:2307.06106 [cs]
22. Weinzierl, S.: Exploring Gated Graph Sequence Neural Networks for Predicting Next Process Activities. In: BPM Workshops. pp. 30–42. Springer (2022)
23. Weytjens, H., De Weerdt, J.: Creating Unbiased Public Benchmark Datasets with Data Leakage Prevention for Predictive Process Monitoring. In: BPM Workshops. pp. 18–29. Springer International Publishing, Cham (2022)
24. Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., Liu, T.Y.: Do Transformers Really Perform Badly for Graph Representation? In: NeurIPS. vol. 34, pp. 28877–28888. Curran Associates, Inc. (2021)
25. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep Sets. In: NeurIPS. vol. 30. Curran Associates, Inc. (2017)