

Enhancing Remaining Time Prediction in Business Processes through Graph Embedding

Thais Rodrigues Neubauer
University of Sao Paulo
Sao Paulo, Brazil
thais.neubauer@usp.br

Jari Peeperkorn
KU Leuven
Leuven, Belgium
jari.peeperkorn@kuleuven.be

Jochen De Weerd
KU Leuven
Leuven, Belgium
jochen.deweerd@kuleuven.be

Marcelo Fantinato
University of Sao Paulo
Sao Paulo, Brazil
m.fantinato@usp.br

Sarajane Marques Peres
University of Sao Paulo
Sao Paulo, Brazil
sarajane@usp.br

Abstract

Accurately predicting the remaining time of business processes is essential for operational efficiency but remains challenging due to the complex interdependencies among process activities. Traditional approaches often fail to capture these complexities effectively. This paper introduces an approach to improving remaining time prediction through the application of graph embedding to enrich the representation of process activities. The proposed approach enriches the data representation for model training that is agnostic to the prediction algorithm. We detail the graph design and explore embedding parameters, applying them to real-world event logs. Our experimental study demonstrates that our approach can reduce percentual prediction error rates by up to 35% compared to traditional methods, showing the effectiveness of graph embeddings in improving predictive accuracy in complex business environments.

Keywords: Business Process Management, Process Mining, Predictive process monitoring, Remaining Time Prediction, Graph Embedding

1. Introduction

Predictive process monitoring (PPM) enhances operational efficiency by using historical data from event logs to predict the remaining time of running process instances, helping prevent delays, meet schedules, and optimize resources (Dumas et al., 2013; Tax et al., 2016; Verenich et al., 2019). However, predictive models often struggle to consider complex relationships, affecting accuracy (Ceravolo et al., 2024).

Traditional PPM approaches do not fully capture this complexity (Ceravolo et al., 2024), limiting strategic

decision-making. More accurate representations of business process execution, accounting for the multiple interrelations among process activities, have the potential to boost predictions (Barbon Junior et al., 2021; De Koninck et al., 2018). Recent techniques have sought to overcome this challenge (Barbon Junior et al., 2021; De Koninck et al., 2018; Neubauer et al., 2023). Notably, graph embedding-based techniques leverage the inherent flexibility of graphs to manage relationships among different entities. By modeling key information about the execution of business processes within graphs and mapping them into dense vector spaces, these techniques successfully capture relevant interrelations without adding unnecessary complexity to the resulting representation (Neubauer et al., 2023).

We contend that graph embedding-based techniques can improve predicting the remaining time of ongoing process instances by supplying predictive models with relevant contextual interrelations within the underlying business processes. Indeed, despite advancements in remaining time prediction, the potential of these newer representation techniques remains largely untapped.

This paper introduces an innovative approach to improving remaining time prediction by applying a graph embedding-based technique to enrich the representation of process activities. Leveraging the graph structure's inherent ability to capture complex relationships, the resulting enriched activity representation is used to represent sequences of activities, thereby enhancing the accuracy of remaining time predictions. We further innovate by exploring the impact of different embedding dimensions. Applied to real-world event logs, our experimental study demonstrates that models using our approach can reduce percentual error rates by up to 35% compared to traditional activity representations.

Event id	Case id	Attributes	
		Activity	Timestamp
...
004	6	activity 2	2021-01-22 14:13
034	6	activity 5	2021-01-23 08:27
106	7	activity 1	2021-01-26 10:19
107	6	activity 10	2021-02-01 10:36
138	7	activity 10	2021-02-22 15:49
149	6	activity 10	2021-03-01 17:03
...

Figure 1: Fragment of an event log

This paper is organized as follows: Sections 2 and 3 present preliminaries and related work. Section 4 outlines the proposed approach. Section 5 details the experimental study. Section 6 covers final remarks.

2. Preliminaries

2.1. Process Mining

Process mining aims at extracting knowledge from *event logs*. An event log consists of *cases* and *cases* consist of *events* (van der Aalst, 2016). Each event refers to the execution of an *activity* (i.e., well-defined step in a process) within a business process. Additionally, an event is typically related to a process instance, also known as *case*. As shown in the example in Figure 1: each line corresponds to an event; each event has an identifier e (column “Event id”); the corresponding case identifier c related to each event is recorded (column “Case id”); additional information (*attributes*) related to events may be recorded, such as the timestamp of the event, included in the Figure, the cost or the person who executed the activity.

Cases can also have descriptive attributes. Each case has a special attribute named *trace* corresponding to its sequence of events. In addition, only a *prefix*, i.e. k first cases’ events, is often used in process mining tasks.

Sequences. Let S be the set of all possible sequences of elements, $\sigma = \langle s_1, s_2, \dots, s_n \rangle$ is a sequence of length n (σ has n elements); $\langle \rangle$ is the empty sequence; and $\sigma_1 \cdot \sigma_2$ is the concatenation of sequences σ_1 and σ_2 .

Prefixes. $head^k(\sigma) = \langle a_1, a_2, \dots, a_k \rangle$ is the prefix of length k ($0 < k < n$) of the sequence σ . For example, for the sequence $\sigma_1 = \langle a, b, c, d, e \rangle$, $head^2(\sigma_1) = \langle a, b \rangle$, $head^3(\sigma_1) = \langle a, b, c \rangle$, and so on.

Given an attribute, we often need to label an event using the values of part of its attributes. For example, one can apply a *function* $\pi_{\mathcal{A}}(e)$ for all events in a trace to obtain its sequence of events in terms of the activity names those events recorded. For example, applying $\pi_{\mathcal{A}}(e)$ to the events of case $c = 6$ from Figure 1, yields $\langle activity2, activity5, activity10, activity10 \rangle$.

Functions. Let \mathcal{E} be the set of all possible event identifiers, \mathcal{T} the time domain, and \mathcal{A} the finite set of process activities. The *timestamp* value is obtained through the function $\pi_{\mathcal{T}}(e) : \mathcal{E} \rightarrow \mathcal{T}$, and the *activity* value is obtained through the function $\pi_{\mathcal{A}}(e) : \mathcal{E} \rightarrow \mathcal{A}$.

2.2. Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM)

RNNs are a type of neural network where the connections between neurons form a directed cycle, specialized in processing sequential data, that is, they operate with a sequence of vectors $s_1; \dots; s_n$, where n is any arbitrary sequence length (Goodfellow et al., 2016). The ability to process sequence-like data makes this type of neural network very useful in PPM (Camargo et al., 2019; Evermann et al., 2017; Tax et al., 2016).

RNN poses an important issue in gradient propagation when applying backpropagation to update weights (Back Propagation Through Time (BPTT)): the gradients either vanish or explode when trying to learn long-term dependencies (Goodfellow et al., 2016).

LSTM was proposed as a solution to this problem. It is a special RNN architecture that creates paths through time that allow the gradients to flow deeper in the input prefix than in a vanilla RNN. Instead of using the previous state, h_{t-1} , LSTMs use a memory cell MC_t that has an internal recurrence and the usual recurrence of vanilla RNN. This internal recurrence is controlled by three gates, f_t , o_t , and i_t , to control the flow of information inside the cell. Information about a new input is accumulated in the memory cell if i_t is activated. The past memory cell status C_{t-1} can be “forgotten” if f_t is activated. The information of C_t will be propagated to the output h_t based on the activation of output gate o_t . The combination of the formulas that define an LSTM is as presented by the following equations:

- $h_t = o_t \circ \tanh(C_t)$,
- $f_t = \text{sigmoid}(b_f + U_f s_t + W_f h_{t-1})$,
- $i_t = \text{sigmoid}(b_i + U_i s_t + W_i h_{t-1})$,
- $o_t = \text{sigmoid}(b_o + U_o s_t + W_o h_{t-1})$,
- $\tilde{C}_t = \tanh(b_C + U_C s_t + W_C h_{t-1})$,
- $C_t = f_t \circ C_{t-1} + i_t \circ \tilde{C}_t$,

where b is the bias vector; U and W are trainable weights; \tilde{C}_t is the cell state for the timestep t ; and C_t is the combination of the past information of the cell with its current information. “ \circ ” denotes the Hadamard product between two matrices.

2.3. Graph Embedding

A graph $G(V, E)$ is defined as a collection of vertices $V = \{v_1, \dots, v_n\}$, also called “nodes”, and edges $E = \{e_{ij}\}_{i,j=1}^n$. The adjacency matrix S of graph G contains nonnegative weights associated with each edge: $s_{ij} \geq 0$. If v_i and v_j are not connected, then $s_{ij} = 0$. For undirected weighted graphs, $s_{ij} = s_{ji} \forall i, j \in [n]$.

Graph-based representation of entities’ relationships has enabled studies in various fields, such as linguistics, biology, and social sciences. Graph-to-vector transformations gained popularity for allowing effective data analysis in vector spaces while preserving graphs’ structural and relational properties. In this paper, we explore random walk-based methods, especially Node2vec. It generates vectors to represent the graph nodes, embedding the graph’s topology and the nodes’ neighborhood information into the resulting vectors, often referred to as “embeddings” (Grover and Leskovec, 2016).

The core of Node2vec lies in its utilization of random walks to obtain lists of nodes that sample their neighborhoods, closely mirroring the strategy used in natural language processing to capture context around words. The random walks of a given graph are controlled by two parameters: p and q . The return parameter p influences the likelihood of the path returning to the source node, thus affecting the walk’s scope between local and global structure. The in-out parameter q controls the exploration; with values less than 1 favoring nodes close to the preceding node, encouraging a breadth-first search (BFS) approach, and values greater than 1 favoring nodes further away, inducing a depth-first search (DFS). The choice of p and q enables the customization of the embedding to reflect various notions of node neighborhood, ranging from immediate peers to nodes reachable through specific path types (Grover and Leskovec, 2016).

The resulting lists of visited neighbors generated for each node are then used by the Skip-gram model to learn the node embeddings. This model, adapted from natural language processing, treats the lists of neighboring nodes as sentences, aiming to predict the neighboring nodes (sentence context) given a node (the current word) (Mikolov et al., 2013). Skip-gram maximizes the likelihood of predicting the neighborhood, ensuring that nodes frequently co-occurring within random walks are positioned closely in the resulting vector space. Through this, both the local neighborhood and topological similarities are considered to generate the resulting vectors (Grover and Leskovec, 2016).

Node2vec’s versatility comes from its ability to learn continuous vector representations for graphs’ nodes,

which can then be employed in a variety of downstream tasks such as clustering and classification.

3. Related Work

3.1. Remaining Time Prediction

Business process monitoring analyzes events produced during the execution of a business process to assess compliance requirements and performance objectives (Dumas et al., 2013). Monitoring can take place offline (e.g., based on periodical reports) or online, via dashboards displaying the performance of currently running process cases (Verenich et al., 2019). PPM refers to online process monitoring techniques based on training models using historical process executions to predict the future state or properties of ongoing executions of a process, e.g. the remaining cycle time of a process instance (Verenich et al., 2019). Such predictive models can be used to alert about problematic process instances or to support prescriptive models, e.g. allocating additional resources to at-risk instances.

In this paper, we focus on predicting time-related properties of ongoing process cases. Given an initial partial trace of a process case (a *prefix*), our goal is to predict a process performance measure in the future, specifically the remaining time until case completion.

A diverse array of machine learning and deep learning techniques can be found in the literature proposing and/or comparing approaches to remaining time prediction. Tax et al. (2016) underscore the LSTM networks in capturing long-term dependencies within event sequences, a critical component often missed by other machine learning techniques. The authors demonstrated LSTM’s superior ability to predict not just the sequence but also the timestamp of future activities. Mehdiyev et al. (2017) presented a multistage methodology employing n-gram representations stacked with autoencoders and deep feedforward neural network classifiers for next activity prediction. Combining ideas from various works, Camargo et al. (2019) applied separately trained embeddings of categorical variables with activity timestamps to predict future events and their corresponding timestamps.

Besides RNNs, other architectures have been proposed for different predictive tasks: such as Convolutional Neural Networks (CNNs) (Pasquadisceglie et al., 2019; Weytjens and De Weerd, 2020), Generative Adversarial Networks (GANs) (Taymouri et al., 2020), and transformer models (Bukhsh et al., 2021). Most recently, PPM for object-centric event data has garnered attention, for

example, Adams et al. (2023) used object information to construct more informative process graphs, capitalizing on the inherent parallelism of objects. The recent works by Venugopal et al. (2021) and Chiorrini et al. (2023) have proposed approaches based on Graph Neural Networks (GNNs) for the next activity prediction task.

Verenich et al. (2019) offered a benchmark of traditional algorithms and reported the robustness of LSTM in diverse data sets, highlighting the importance of feature engineering in improving prediction accuracy. Rama-Maneiro et al. (2023), provided a benchmark of deep learning approaches in PPM, where recurrent neural networks, especially LSTMs, stand out for their ability to capture the sequential nature of event logs.

Our research distinguishes itself in the context of remaining time prediction by providing an enriched graph embedding-based knowledge representation to enhance the reasoning capabilities of predictive models. Specifically, we aim to enhance LSTM performance, leveraging its established success with sequential data in event logs. We hypothesize that LSTM robustness can assess the benefits of the proposed enriched representation for remaining time prediction. Moreover, our approach differs from GNN-based methods by creating task-independent embeddings. Even though task-specific embeddings can improve model performance for a particular task, they reduce the potential for transferring representation knowledge to other tasks and require new embeddings for each task. This limits the versatility and generalizability of the learned representations. By developing a more general representation, our approach seeks to maintain versatility and broader applicability.

3.2. Vector Representation in Process Mining

Adequately represented input data is essential for any problem-solving strategy. In process mining literature, a variety of works emphasize the relevance of an adequate representation in reaching a more efficient and effective solution (Luna et al., 2021; Tavares et al., 2023). Since multiple machine learning-based approaches require a vector representation as input, the different concepts related to the business process execution, such as activities, cases, traces, attributes, etc. have to be mapped to vector spaces. This is a key challenge for properly aligning machine learning and process mining applications (Ceravolo et al., 2024).

Although various mapping schemes exist in process mining, few studies explore their impact on final results (Barbon Junior et al., 2021; Ceravolo et al., 2024; Tavares et al., 2023). The schemes in this context can be organized into three main categories (Barbon

Junior et al., 2021): *trace replay and alignment*, where the resulting vectors' dimensions are conformance measures, such as missing and remaining activities (Barbon Junior et al., 2021); *word embedding*, grounded in natural language processing, which can be subdivided into one-hot encoding (Nolle et al., 2018), count-based (Appice and Malerba, 2016; Song et al., 2009) and neural network-based (De Koninck et al., 2018); *graph embedding*, in which complex relationships, such as long-term relations, can be modeled in graphs and, thus, be mapped to vectors that embed graph neighborhood and structure roles (Barbon Junior et al., 2021; Neubauer et al., 2023; Tavares et al., 2023).

While many schemes exist for representing event log information in vector spaces, they may have limitations that negatively affect results. For example, information loss in the sequence of performed activities when applying trace replay and alignment or count-based; loss of connection between distant events may occur if the defined length is narrow when applying one-hot encoding (Nolle et al., 2018); sparsity in one-hot encoding and count-based approaches due to the relation of the number of dimensions to the number of different values registered for the attributes.

Graph embedding-based approaches address the limitations of other representational schemes by leveraging their flexibility in capturing relevant process characteristics and relationships through nodes and edges. Barbon Junior et al. (2021), Tavares et al. (2023) pioneered the application of graph embedding in process mining, reporting its lower computational costs than word embedding and demonstrating its potential in anomaly detection. By using synthetic event logs, these studies left the application of graph embeddings to real-world processes untapped. Neubauer et al. (2023) applied graph embeddings to real-world event logs for trace clustering, reporting it can generate vector spaces following specific task and context-related constraints. Yet, these prior studies did not investigate the effects of key embedding parameters like dimensionality nor explored how the resulting embedding for activity representation can improve prediction tasks' results. We address these gaps by varying embedding dimension values, applying graph embeddings to real event logs, and using the resulting activity embedding representation to improve remaining time prediction.

4. Graph Embedding-Based Activity Representation for Remaining Time Prediction

The proposed approach for constructing vector representations of activities is based on graph

embedding to capture intrinsic relationships within business processes, by transforming process-related information into a graph structure. The graph is built exclusively considering the events of a pre-defined training set, ensuring the principle in predictive modeling of separating training data from test data.

To enable the use of graph embeddings, the business process context needs to be mapped to a graph structure. Specifically aimed at addressing the remaining time prediction task, we propose a mapping based on the representation of events, activities, and cases. These elements must be represented by nodes, organized into their respective sets of nodes GN_{event} , GN_{ac} , and GN_{case} . Each node in a set is connected to nodes in other sets, allowing for the representation of the characteristics of the cases and the events that constitute the process dynamics. Edges relate the event nodes ($en \in GN_{event}$) to their corresponding activity (one node of GN_{ac}) and case (one node of GN_{case}). Figure 2 exemplifies how graphs are constructed based on our choices of nodes and edge sets.

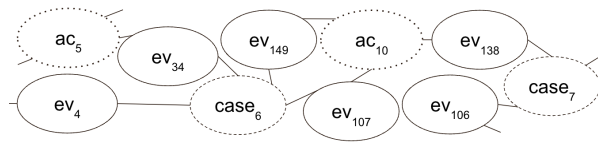


Figure 2: Fragment of the graph resulting from mapping the event log fragment of Figure 1 following the proposed approach, showing that, for example, event 34 (node ev_{34}) refers to the execution of “activity 5” (ac_5) in the case 6 ($case_6$).

After a graph is created from an event log, Node2vec generates embedding vectors for each of its nodes. Although embeddings corresponding to event nodes are included in these results, we focused on the activity embeddings. This ensures consistency in applying predictive models online by enabling the replication of the input generation process used during training with historical data. If event embeddings were used, replicating the input generation process for new events received online would be impossible since these new events were not part of the graph embedding training and thus would not have corresponding embeddings.

Since the resulting embedding vectors are based on random walks, every node in the graph may be involved in each resulting activity embedding vector, transferring neighborhood information into the embedding space. For example, a random walk containing the following trajectory – ac_{10} , ev_{107} , $case_6$, ev_{149} , $case_6$, ev_{34} , ac_5 – will embed the information that “activity 10” was executed at least twice in “case 6” (events 107 and

149) and that the “activity 5” is also present in that case (event 34). The representation of “activity 5” will contain similar information that will contribute to a similar vector representation for these two activities due to their occurrences in the same case.

Considering the relevance of the order of events in the execution of business processes, predictors of the remaining time that take sequences as input have presented higher accuracy (Tax et al., 2016). To apply the activity representation obtained through the graph embedding-based procedure proposed herein to the input of such predictors, each trace σ recorded in the log is segmented into prefixes $head^k$ based on their number of events ($|\sigma|$). Each event $e_i \in head^k(\sigma)$ is assigned to a feature vector \vec{v} . Following the procedures reported in (Tax et al., 2016), the event feature vector is formed by the concatenation of: (i) the vector representation $\vec{ac} \in AC$ of the activity from the set of possible activities AC corresponding to the activity performed in e_i ; (ii) a set $AF = \{a\vec{f}^1, \dots, a\vec{f}^m\}$ of additional m vector features that further characterize the event execution, such as vectors representing resources, attributes or time-related characteristics.

The first prefix contains only one event as input, thus, lacking sufficient data for making predictions. The last event marks the conclusion of the trace, leaving no remaining time to predict if it is included in a prefix. Therefore, k ranges from 2 to $|\sigma| - 1$, where $|\sigma|$ denotes the total number of events in a trace. Given a S set of prefixes generated from the traces recorded in an event log, following the reports in (Tax et al., 2016), the final input for a sequence-based predictor was defined by employing a size-fixed zero-padded prefix vector representation based on the biggest k in S , i.e. each $head^k \in S$ is represented by a fixed amount of events, defined by Max_k , the maximum value of k in S ; prefixes for which k is smaller than Max_k , zeroes are used to fill the $Max_k - k$ events at beginning of the prefix representation. Figure 3 illustrates the proposed sequence representation based on activity representation obtained through graph embedding.

5. Experimental evaluation

To evaluate the impact of graph embedding-based activity representation on the remaining time prediction task, we employ the proposed approach considering the context of two event logs: the *helpdesk* event log, previously explored by Tax et al. (2016), and the *incidents* event log, investigated by Neubauer et.al (2022) and Neubauer et. al (2023). We compared predictive models generated using activity representations obtained as described in Section 4

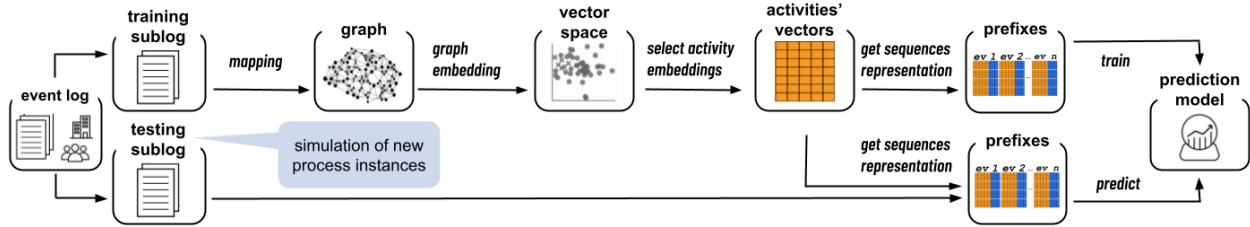


Figure 3: Overview of the proposed approach to applying graph embedding-based activity vector representation to represent event sequences (prefixes) originally registered in an event log. The context of the business process dynamics is embedded into the activity representation and the ordering aspect of this dynamic is considered to build the prefixes

against those produced using conventional one-hot encoding, commonly used in process mining. The following sections detail: the business process contexts and corresponding event logs; the experiments design; and the analysis of the obtained results.

5.1. Event Logs Description

The event logs “Helpdesk” and “Incidents” were used in the experimental evaluation.

Helpdesk Contains events from a ticketing management process of the helpdesk of a software company¹. The underlying business process is composed of nine activities, and it contains 3,804 cases and 13,710 events. All cases start with the insertion of a new ticket into the ticketing management system. Each case ends when the issue is resolved and the ticket is closed.

Incidents Contains 141,712 events referring to 24,918 cases of a real-world incident management process in an IT company (Amaral et al., 2019). Each event is described by 29 attributes, including the life-cycle attribute “*incident_state*” (referring to the state in the incident management standard process) and other attributes such as *priority* and *reassignment_count*².

The attributes show sequential repetitions within a case, as the system records all values whenever any change occurs. Following the procedure described in Neubauer et al. (2023), an activity notion was defined based on the value changes in each pair of subsequent events for three selected attributes - *incident_state*, *category*, and *priority*. These attributes were indicated by experts as the most relevant to the remaining time prediction task (Neubauer et al., 2022). If a change was identified in the value of ‘*incident_state*’, the value presented in the latest event is registered as an activity. If changes were identified in the values of “*category*” or “*priority*”, the activities “Change in category” or

ID	timestamp	incident_state	category	priority
1	2016-03-01 10:03	$v_1 = \text{"New"}$	$v_1 = \text{"Internet"}$	$v_1 = 1$
2	2016-03-01 11:24	$v_2 = \text{"Active"}$	$v_1 = \text{"Internet"}$	$v_1 = 1$
3	2016-03-01 10:05	$v_1 = \text{"New"}$	$v_2 = \text{"SAP"}$	$v_2 = 2$
4	2016-03-01 10:16	$v_1 = \text{"New"}$	$v_3 = \text{"Telephony"}$	$v_2 = 2$
5	2016-03-01 11:24	$v_1 = \text{"New"}$	$v_4 = \text{"Network"}$	$v_2 = 2$
6	2016-03-02 09:32	$v_1 = \text{"New"}$	$v_4 = \text{"Network"}$	$v_3 = 4$

Original event log format

ID	Activity	timestamp
1	New	2016-03-01 10:03
2	Change in state	2016-03-01 11:24
3	First state	2016-03-01 10:05
4	New	2016-03-01 10:16
5	First state	2016-03-01 11:24
6	Change in priority	2016-03-02 09:32

Resulting activities

Figure 4: Examples of the activity notion definition for the *Incidents event log*.

“Change in priority” are registered, respectively. If no changes were identified between the pair of events for this set of attributes, the activity “Change in another attribute” was registered. The timestamp assigned to the registered activities is the same as the latest event in the analyzed pair. The definition of this activity notion, illustrated in Figure 4, results in the identification of 11 distinct activities within this log.

5.2. Experimental Design

This section details the experimental evaluation.

Node2vec algorithm Through a value inspection procedure for the algorithm’s hyperparameters, we fixed the length of the random walk (*walk_length* = 30), the size of the context window (*window_size* = 10), and the return parameter (*p* = 3). Embedding dimensions 3, 5, 8, 10, 12, 15 were tested, using default values for the remaining parameters³. The graphs used as input for Node2vec were constructed using only the cases included in the training set. To bolster the robustness of the findings resulting from representation comparison, the Node2vec algorithm was executed three times per parameter combination. This triadic approach ensures

¹ Available at <https://data.mendeley.com/datasets/39bp3vv62t/1>

² Attributes’ description at <https://archive.ics.uci.edu/dataset/498>

³ “node2vec” package, version 0.4.6, available at <https://pypi.org/project/node2vec/>.

that the resulting activity representations are not the product of a single execution’s randomness.

LSTM architecture The LSTM architecture used was based on the results reported in Tax et al. (2016) and the results of preliminary experiments carried out in the context of this study. The hyperparameters were left at their default values, except for the number of layers and associated number of neurons, which were set to one dense layer with 100 neurons. The weights of the LSTM were optimized using the Adam learning algorithm to minimize the mean absolute error between the ground truth and the predicted values.

Feature space The feature vector \vec{v} representing an event e_i is obtained following the proposed approach as described in Section 4. For this experimental evaluation, there are tests in which graph embedding-based \vec{ac} is substituted by a one-hot encoding activity representation (baseline). The additional features AF are af_1 , the position i of the event in the trace σ , and four time-based features ($af_2(e_i)$, $af_3(e_i)$, $af_4(e_i)$, $af_5(e_i)$).

$af_2(e_i)$: elapsed time since the beginning of the trace (timestamp of the first event in the trace) normalized by the average elapsed time across events in the training set.

$af_3(e_i)$: elapsed time between the previous event e_{i-1} in the prefix $head^k(\sigma)$ and e_i . $af_3(e_i) = 0$ if $i = 1$ and $af_3(e_i) = \pi_{\mathcal{T}}(e_i) - \pi_{\mathcal{T}}(e_{i-1})$ otherwise. $af_3(e_i)$ is normalized by the average elapsed time across events in the training set. This feature allows LSTM to learn that time differences may depend on the trace position.

$af_4(e_i)$: elapsed time within the day (seconds since midnight), normalized by total seconds in a day (86,400). This feature distinguishes events executed during office hours from those in another period.

$af_5(e_i)$: index of the weekday, normalized by the maximum possible value (7). This feature distinguishes events regarding their execution being during office hours or not, and identifying events executed close to the end of the week.

Training/testing strategy Adhering to (Tax et al., 2016), this study implements a holdout-based strategy to split event logs into training and testing sets, using the first chronological 2/3 of cases for training and the remaining 1/3 for model evaluation. Traces from these case sets generate prefixes through the established prefix creation methodology, ensuring each process instance is fully represented and maintaining execution coherence at the case level. The target output o^k of the time step k is the expected remaining time of the corresponding prefix $head^k(\sigma)$. The remaining time o^k is the time difference between the last event of the trace σ and the

timestamp of the last event e_k of the prefix $head^k(\sigma)$, that is, $o^k(head^k(\sigma)) = \pi_{\mathcal{T}}(e_{|\sigma|}) - \pi_{\mathcal{T}}(e_k)$. To ensure the robustness of the analysis, each sequence representation obtained for each log is subjected to three LSTM model training. This triple execution prevents attributing findings to the randomness of a single run, reinforcing the validity of the results.

Evaluation strategy The quality of the predictions was evaluated using the well-known error indicator for regression: Mean Absolute Percentage Error (MAPE). The choice of this metric is based on three aspects.

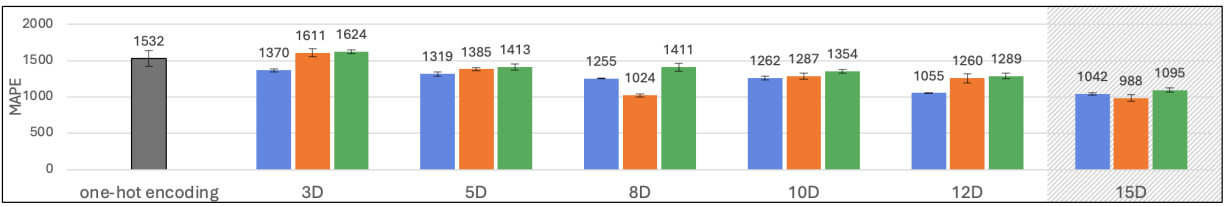
Firstly, MAPE is proportional to the expected value’s magnitude. For example, if 100 seconds is expected and the prediction is 500, the MAPE is calculated as $abs(500 - 100)/100 = 4$. This means the prediction error is four times the expected value. Even though 400 seconds can be considered negligible, the fact that the error was four times the expected value can be relevant, especially considering online PPM. Secondly, in incident management processes with high variability, MAPE is more insightful than position-based normalized metrics, which tend to nullify differences. MAPE highlights overestimation errors, showing whether a predictor is more suitable for certain process management contexts. For example, in reducing operational costs with resource allocation, overestimating predictors pose a greater risk and are less favorable than those that underestimate. Lastly, MAPE allows for fair and straightforward comparisons across event logs, as it is independent of the cases’ duration distribution, unlike absolute error.

Even though MAPE yields high values when predicted values are high or expected values are very low, this issue is consistent across all compared models⁴.

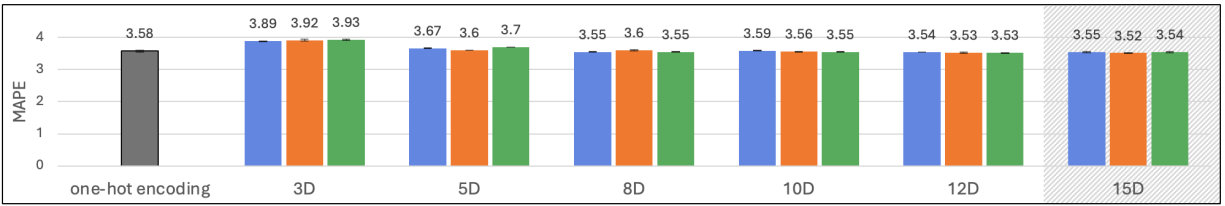
Computational testing environment This experimental study was carried out through Python scripts using Keras recurrent neural network library (Chollet, 2015)⁵. The graph embedding models were trained on an Intel Xeon Gold 642R CPU, taking from 173 to 177 seconds for the *Helpdesk* event log, and from 260 to 285 minutes for the *Incidents* event log. LSTM models were trained on Google Colab Machines with TPU, taking from 312 to 456 seconds for the *Helpdesk* event log and from 506 to 733 seconds for the *Incidents* event log. The execution time for obtaining a prediction is in the order of milliseconds both for the graph embedding and the LSTM models.

⁴Out of the 26,2793 prefixes in the test set, 24 have the remaining time $o^k = 0$ and, thus, had their absolute errors not considered in MAPE calculations to avoid divide-by-zero errors.

⁵Data and code available at anonymous.4open.science/r/node2vec-and-lstm4remaining-time-pred-E4E2



(a) Helpdesk event log



(b) Incidents event log

Figure 5: The bar graphs show MAPE across embedding dimension values for each event log. The gray bar represents the baseline (one-hot encoded activity representation). Each dimension value has three colored bars, each corresponding to one of the three Node2vec runs, showing the average MAPE (with standard deviation) from each run’s three resulting LSTM models. Most significant improvements are highlighted with hatched areas.

5.3. Results and Discussion

The results show graph embeddings improved remaining time prediction accuracy. Figure 5 shows, for both event logs, the MAPE obtained using one-hot encoded activity representation and graph embedding-based representation.

In the Helpdesk event log, case duration leans towards shorter intervals. This skews MAPE, since most absolute errors are high-valued compared to small expected values. Although interpreting these MAPE values can be challenging, the 15-dimensional (15D) activity embeddings show improved prediction accuracy. The difference of 544 from the baseline (1,532) to an average of 988 for the second 15D embedding run shows that graph embedding-based representation decreased MAPE by 35.5%.

The distribution of MAPE values obtained for this log shows that individual percentage absolute errors greater than 10 are observed when the expected remaining time is around 30 minutes. Considering that MAPE is more influenced by these higher magnitudes of individual percentage errors, the 544 difference in MAPE can be interpreted as predictions approximately 11 days more accurate ($544 \times 0.5 \text{ hour} = 272 \text{ hours} \approx 11 \text{ days}$). To put it into perspective considering the average case duration in this log (8.8 days), this difference of 11 days would represent an error reduction of roughly 125% of the average cases’ duration.

For the Incidents event log, although improvements were less pronounced, embeddings still refined

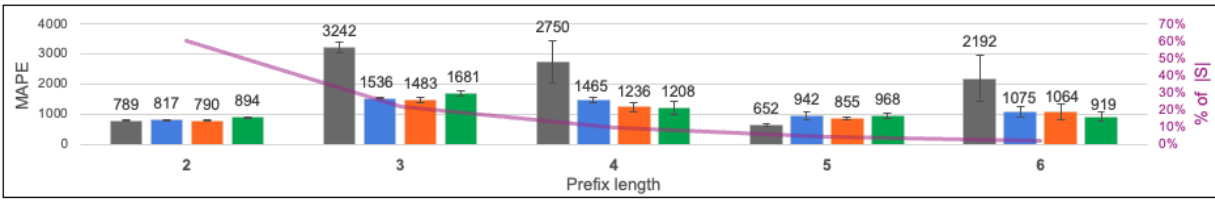
prediction accuracy⁶. The second run of 15D proves that graph embedding-based activity representation can decrease MAPE by 0.06 (from 3.58 baseline to 3.52 on average for this run). Considering the average expected remaining time in this log of 268 hours, the 0.06 difference can be interpreted as 16 hours more accurate predictions. In perspective to the average case duration in this log (10.3 days), this would represent an error reduction of approximately 6.5%.

Figure 6 offers additional insights into the performance of graph embedding through a more granular analysis across varying prefix lengths k for the 15D embeddings. As depicted in Figure 6a, improvements are more pronounced for specific prefix lengths in the Helpdesk event log. The most significant improvement was observed for prefixes of lengths 3 and 4, where the 15D embeddings achieved a remarkable reduction in MAPE compared to the baseline.

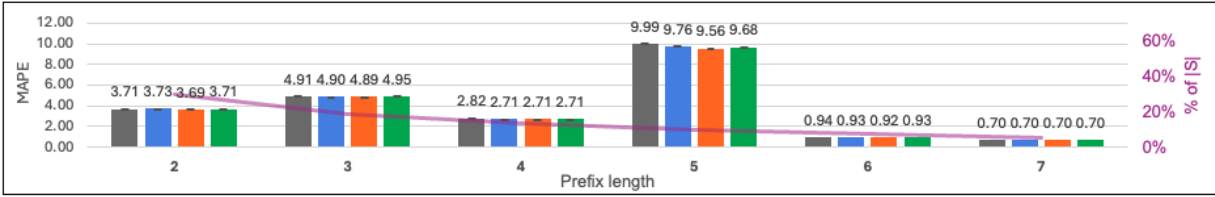
For the Incidents event, as depicted in Figure 6b, the greatest improvements were observed as k increases until prefix length $k = 5$ ⁷. However, corroborating with the analysis across embedding dimensions’ values,

⁶For each log, predictions from the 54 models using activity embeddings were compared to predictions from the three corresponding baselines using Wilcoxon tests ($p < 0.05$). Only 12 “Helpdesk” and eight “Incidents” models did not differ statistically from the baselines. Among the nine models for each log with 15D embeddings (three LSTMs for each of the three Node2vec runs) in Figure 6), those with the lowest MAPE differed from the baselines.

⁷The spike in MAPE for prefixes of length 5 in the Incidents event log is due to a characteristic of this specific event log, where there is a higher concentration of predictions with expected values close to zero at this prefix length compared to others. These small denominators inflate MAPE, contributing to the observed spike.



(a) Helpdesk dataset



(b) Incidents dataset

Figure 6: MAPE by prefix length for each event log. Similar to Figure 5, each bar color represents one run of Node2vec and the baseline is shown as a gray bar. However, here only the MAPE for the runs with 15 dimensions as parameter are shown across different prefix length. The secondary axis shows the percentage of S within each prefix length. Prefixes under 2% for *Helpdesk* are excluded, while for *Incidents*, displayed lengths account for over 80% of S .

the observed differences are moderate. Moreover, since the number of predictions decreases as k increases, the greatest differences affect fewer predictions.

The differences across prefix lengths suggest that graph embedding is more effective at certain stages of the process in improving prediction accuracy, highlighting the potential for tailored approaches. In this kind of approach, the choice of model could be strategically aligned with the specific requirements of the process execution phase. Moreover, the variance in the effectiveness of graph embedding between event logs underscores the influence of log-specific characteristics and the complexity of the underlying business processes on the performance of predictive models, corroborating with the findings in Rama-Maneiro et al. (2023).

6. Conclusion

In this paper, we apply graph embedding-based activity representation combined with LSTM to enhance the accuracy of remaining time prediction in business processes. We also explored the impact of the key embedding parameter of dimensionality for the vector embeddings obtained via graph embedding. With the application to real-world event logs, we demonstrated that the proposed approach can reduce prediction error and that the right choice of parameters has a central role. The most notable improvement was observed for the *Helpdesk* event log, with a 35.5% reduction in MAPE values. This underscores the potential of

graph embeddings to capture the complexities intrinsic to existing relationships within business processes.

Our research contributes to ongoing efforts to improve PPM, offering a novel approach that encodes relevant information through graph structures. The variable performance across the event logs underscores the need to identify optimal graph embedding configurations and assess their generalizability across different business process scenarios. Future work will also explore the further enrichment of the resulting representation via adding other process-related information into the graph structure, such as resource allocation patterns and case attributes. Additionally, Graph Neural Networks (GNNs) have shown promise in tasks such as next activity prediction. A potential research direction is to compare vector embeddings from our approach with those generated by GNNs for remaining time prediction, investigating any biases introduced by task-dependent embedding generation.

References

- Adams, J. N., Park, G., & van der Aalst, W. M. (2023). Preserving complex object-centric graph structures to improve machine learning tasks in process mining. *Eng. Appl. Artif. Intell.*, 125.
- Amaral, C. A. L., Fantinato, M., Reijers, H. A., & Peres, S. M. (2019). Enhancing completion time prediction through attribute selection. *Lec. Notes Bus. Inf. Process.*, 346.

- Appice, A., & Malerba, D. (2016). A co-training strategy for multiple view clustering in process mining. *IEEE Trans. Serv. Comput.*, 9.
- Barbon Junior, S., Ceravolo, P., Damiani, E., & Tavares, G. M. (2021). Evaluating trace encoding methods in process mining. *From Data to Models and Back*, 12611, 174–189.
- Bukhsh, Z. A., Saeed, A., & Dijkman, R. M. (2021). Processtransformer: Predictive business process monitoring with transformer network.
- Camargo, M., Dumas, M., & González-Rojas, O. (2019). Learning accurate lstm models of business processes. *Int. Conf. Bus. Process Manage.*, 286–302.
- Ceravolo, P., Junior, S. B., Damiani, E., & Van Der Aalst, W. (2024). Tuning machine learning to address process mining requirements. *IEEE Access*, 12, 24583–24595.
- Chiorrini, A., Diamantini, C., Genga, L., & Potena, D. (2023). Multi-perspective enriched instance graphs for next activity prediction through graph neural network. *J Intell Inf Syst*, 61.
- Chollet, F. (2015). Keras [https://github.com/fchollet/keras].
- De Koninck, P., vanden Broucke, S., & De Weerd, J. (2018). Act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes. *Bus. Process Manage.*, 11080.
- Dumas, M., Rosa, M., Mendling, J., & Reijers, H. (2013). *Fundamentals of business process management* (1st). Springer.
- Evermann, J., Rehse, J.-R., & Fettke, P. (2017). Predicting process behaviour using deep learning. *Decision Support Systems*, 100.
- Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
- Grover, A., & Leskovec, J. (2016). Node2vec: Scalable feature learning for networks. *ACM Int. Conf. Knowl. Discovery Data Min.*, 8.
- Luna, M. A. S., Lima, A. P., Neubauer, T. R., Fantinato, M., & Peres, S. M. (2021). Vector space models for trace clustering: A comparative study. *Anais do XVIII Encontro Nacional de Inteligência Artificial e Computacional*.
- Mehdiyev, N., Evermann, J., & Fettke, P. (2017). A multi-stage deep learning approach for business process event prediction. *IEEE Conf. Bus. Inf.*, 1.
- Mikolov, T., Chen, K., Corrado, G. S., & Dean, J. (2013). Efficient estimation of word representations in vector space. *Int. Conf. Learn. Representations*.
- Neubauer, T. R., Fernandes, A. G. L., Fantinato, M., & Peres, S. M. (2022). Interactive trace clustering to enhance incident completion time prediction in process mining. *Int. Workshop on Comput. Intell. for Process Min.*, 1–12.
- Neubauer, T. R., Peeperkorn, J., Peres, S. M., De Weerd, J., & Fantinato, M. (2023). Vector representation for business process: Graph embedding for domain knowledge integration. *Int. Conf. Mach. Learn.*
- Nolle, T., Luetggen, S., A., S., & Mühlhäuser, M. (2018). Analyzing business process anomalies using autoencoders. *Mach Learn*, 107.
- Pasquadibisceglie, V., Appice, A., Castellano, G., & Malerba, D. (2019). Using convolutional neural networks for predictive process analytics. *Int. Conf. Process Min.*
- Rama-Maneiro, E., Vidal, J. C., & Lama, M. (2023). Deep learning for predictive business process monitoring: Review and benchmark. *IEEE Trans. Serv. Comput.*, 16(1), 739–756.
- Song, M., Günther, C., & van der Aalst, W. (2009). Trace clustering in process mining. *Bus. Process Manage. Workshops*, 17, 109–120.
- Tavares, G. M., Oyamada, R. S., Barbon Junior, S., & Ceravolo, P. (2023). Trace encoding in process mining: A survey and benchmarking. *Eng. Appl. Artif. Intell.*, 126, 107028.
- Tax, N., Verenich, I., Rosa, M. L., & Dumas, M. (2016). Predictive business process monitoring with lstm neural networks. *Int. Conf. Adv. Inf. Syst. Eng.*
- Taymouri, F., La Rosa, M., Erfani, S., Bozorgi, Z. D., & Verenich, I. (2020). Predictive business process monitoring via generative adversarial nets: The case of next event prediction. *Bus. Process Manage.*, 237–256.
- van der Aalst, W. M. P. (2016). *Process mining: Data science in action* (2nd ed.). Springer.
- Venugopal, I., Töllich, J., Fairbank, M., & Scherp, A. (2021). A comparison of deep-learning methods for analysing and predicting business processes. *Int. Joint Conf. Neural Networks*.
- Verenich, I., Dumas, M., Rosa, M. L., Maggi, F. M., & Teinemaa, I. (2019). Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. *ACM Trans. Intell. Syst. Technol.*, 10(4).
- Weytjens, H., & De Weerd, J. (2020). Process outcome prediction: Cnn vs. lstm (with attention). *Bus. Process Manage. Workshops*, 321–333.