QTab: Inter-Case-Aware Remaining Time Prediction via Queuing Networks and Tabular Machine Learning

Keyvan Amiri Elyasi[1][0009−0007−3016−2392], Andrej Tschalzev[1][0000−0002−0638−5744], 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
andrej.tschalzev@uni-mannheim.de
[2] Faculty of Computer Science, University of Vienna, Austria
han.van.der.aa@univie.ac.at

Abstract. Process-aware information systems support the execution and monitoring of business processes. The recorded event data can be leveraged to predict the remaining time of running cases, enabling organizations to meet service-level targets, and enhance operational performance. However, most existing approaches treat cases as being independent from each other, even though there often is contention between cases for shared resources. While some approaches account for inter-case dependencies, they overlook the queuing dynamics arising from work handovers between resources. Moreover, inter-case-aware approaches often use traditional tree-based models, which generally achieve lower accuracy than modern deep learning models. To overcome these limitations, we introduce QTab, a novel remaining time prediction approach powered by two components: (1) modeling of resource interactions and work handovers through a queuing network to derive informative inter-case features; (2) Modern tabular neural networks trained in sophisticated ensembling pipelines. Experimental results on eight real-world event logs show that QTab achieves competitive or superior predictive accuracy compared to existing tree-based and deep learning approaches in remaining time prediction. At the same time, it offers better scalability in feature extraction and improved efficiency in model training.

Keywords: Predictive Process Monitoring · Remaining Time Prediction · Queuing Networks · Machine Learning for Tabular Data.

1 Introduction

Business processes are typically supported by information systems that store data on process executions in event logs. Predictive process monitoring (PPM) uses the historical data from completed process instances (cases) to build models that forecast the future of ongoing cases. A key task in PPM is remaining

time prediction, which estimates how long a running case will take to complete. Accurate estimates support planning, scheduling, and resource allocation, while also supporting risk management through early detection of potential delays [8].

Existing machine learning approaches for remaining time prediction extract partial execution traces from event logs and encode them as feature vectors for supervised learning [28]. Most approaches rely solely on intra-case features that describe individual cases. They assume that—or act as if—cases are independent and that process behavior remains stationary over time [7, 22]. In practice, however, cases often compete for shared resources or are processed in batches, violating the independence assumption [7, 16, 18]. Moreover, process dynamics are typically non-stationary, for example due to fluctuating case arrival rates [17].

Recognizing that remaining time is influenced by factors external to individual cases, some approaches incorporate inter-case features like workload and the similarity of concurrent cases as contextual information to improve prediction accuracy [13, 16, 19, 21–23]. While these works demonstrate the promise of inter-case-aware modeling, they also face notable limitations. First, they primarily capture dependencies between concurrent cases at the same activity transitions [13, 16, 19, 21]. In reality, however, cases queue for resources, not activities [24]. Second, the practical applicability of several approaches [16, 19] is constrained by their reliance on domain expertise and user input to identify process segments (i.e., activity transitions) whose dynamics should be modeled. Finally, the prediction accuracy of existing approaches is limited either by the use of coarse-grained inter-case features [13, 22, 23] or by reliance on traditional tree-based machine learning models [16, 19, 21].

To address these limitations, this paper introduces QTab, a novel approach for remaining time prediction. QTab constructs a queuing network that explicitly models work handovers between organizational entities (e.g., resources or roles), rather than activity transitions, to capture inter-case dependencies. The network is automatically derived from event logs, without requiring domain knowledge. It represents organizational entities as queuing nodes and work handovers as edges, where queues naturally form. At any point in time, the network can be queried to extract fine-grained inter-case features such as workloads, queue lengths, and temporal dependencies among concurrent cases. QTab integrates these features with intra-case attributes into a tabular dataset and trains a deep neural network to estimate the remaining time of ongoing cases. The flexible node granularity of the queuing network improves scalability in feature extraction. It keeps the feature space tractable and enables the use of tabular neural networks. QTab leverages recent advances in tabular deep learning and employs a parameter-efficient ensemble of multilayer perceptrons [11] trained within a cross-validation ensemble pipeline [9].

We evaluate QTab against state-of-the-art tree-based and domain-specific deep learning models on eight publicly available real-world event logs. The results demonstrate that QTab consistently achieves strong predictive performance for the remaining time prediction task, while offering favorable scalability and efficiency characteristics compared to existing approaches.

The paper is organized as follows. Section 2 reviews existing inter-case-aware approaches. The QTab approach is introduced in Section 3 and is evaluated in Section 4. Finally, Section 5 concludes the paper and outlines future work.

2 Related Work and Open Challenges

This section reviews existing inter-case-aware approaches for remaining time prediction, followed by a discussion of the main challenges they leave unresolved.
Existing approaches. *Workload encoding* incorporates the number of concurrent cases as contextual information for remaining time prediction. Certain approaches rely on simple inter-case features, such as the overall process-level workload [2] or the number of concurrent cases per case type defined through domain knowledge [22]. However, these features fail to capture temporal dependencies among concurrently running cases. *Similarity-based encoding* considers the control-flow and temporal distance of a running case to its concurrent cases as inter-case features [23]. Nevertheless, these approaches [2, 22, 23] remain coarse-grained and overlook key queuing dynamics. As a result, predictive accuracy may degrade. For instance, high-level workload abstractions ignore how workload is distributed across different process steps, and temporal-distance features may introduce prediction latency: if congestion increases at a particular step, its effects will not be visible for cases that have not yet reached that step.

To incorporate more fine-grained inter-case features, some approaches utilize the *performance spectrum* [6], a visual process analysis tool that depicts each case's performance across process steps over time. It enables the identification of critical process segments and supports performance analysis within those segments [7]. However, this approach provides only aggregated insights—such as the likelihood of long waiting times—rather than predicting remaining time for individual cases. Subsequent studies extended performance spectrum analysis to remaining time prediction [16,19], but they still depend heavily on domain knowledge or visual analysis to determine relevant segments. Separately, Grinvald et al. [12] introduce a conceptual framework that derives inter-case features from temporal, organizational, and data perspectives by aggregating corresponding attributes across peer cases of a running instance. Yet, incorporating multiple process perspectives in this manner requires substantial manual feature engineering and often results in a high-dimensional feature space.

Approaches capable of extracting fine-grained inter-case features without domain expertise or extensive manual feature engineering remain scarce. One such approach is *load state encoding*, which measures workload at critical activities that strongly influence process performance [13]. However, workload features overlook the role of arrival rates in shaping temporal dependencies among concurrent cases. Queuing theory, which considers both arrival rates and workloads, offers a promising solution. For example, *congestion graphs* [21] construct a network in which nodes represent activities and edges capture their transitions. This network is queried to extract inter-case features that reflect queuing dynamics at activity transitions. Yet, these features overlook resource interactions. Moreover,

generating features for all activities and their possible transitions quickly leads to high-dimensional feature spaces, particularly in complex processes with many distinct activities and a large number of possible transitions.

Open challenges. While the existing inter-case-aware approaches provide valuable insights, they still leave two important challenges unresolved.

Neglecting resource interactions. Most existing approaches capture inter-case dependencies primarily at the level of activity transitions. However, business process delays are often influenced by resource contention, where cases compete for shared resources and wait in queues before processing [1, 3, 18]. This motivates investigating whether modeling queuing dynamics at the resource level can provide additional predictive information beyond activity-level workload features. For example, consider a loan application process in which the financial risk assessment activity for high-value applications is executed by team A, and lower-value applications by team B. If team A experiences a temporary capacity drop, or if an interest-rate change triggers a surge in high-value applications, delays arise specifically for cases handled by team A, while cases assigned to team B remain unaffected. Approaches that capture workloads at the activity level [13, 16, 19, 21] fail to account for such resource-specific effects. To address this, we propose an approach that explicitly models the queuing dynamics arising from work handovers between resources.

Reliance on traditional machine learning. Incorporating fine-grained inter-case features into sequence encoding produces high-dimensional inputs. This increases computational cost and may degrade predictive performance due to the curse of dimensionality [4, 8]. To address this, most existing inter-case–aware approaches for remaining time prediction rely on tabular encoding and tree-based models, such as random forests [16, 22] and gradient boosting [16, 19, 21, 22]. In this setting, the predictive model operates on pre-aggregated features and inter-case features are typically extracted only for the most recent event. While this avoids excessive dimensionality, it limits the model's ability to capture temporal dependencies across earlier events. Consequently, tree-based approaches are more efficient and offer greater explainability, but they often yield suboptimal accuracy compared to deep learning models [28]. Recent advances in deep learning for tabular data [11, 14, 15] show that neural architectures can model complex interactions among heterogeneous features more effectively, achieving state-of-the-art performance on the most recent curated tabular benchmark TabArena [9]. Motivated by the need to better navigate the performance–efficiency trade-off, this paper is the first to bring these advances to predictive process monitoring.

3 QTab for Inter-Case-Aware Remaining Time Prediction

This section introduces QTab, our approach for predicting remaining time in business processes. As shown in Figure 1, it comprises training (blue) and inference (red) phases, with shared components in black.

During training, QTab processes the event log in three main steps: (1) it augments the log by identifying triggering activities and discovering resource

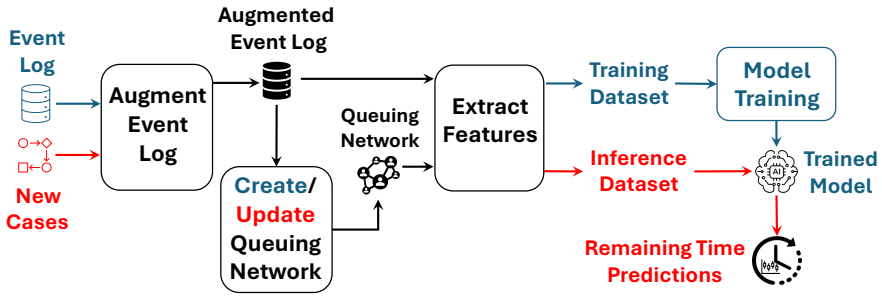


Fig. 1: Overview of the main steps of QTab.

roles; (2) it constructs a queuing network from the augmented log to model inter-case dependencies; and (3) it extracts intra- and inter-case features from both the augmented log and the queuing network to create a tabular dataset. Each row of this dataset represents an incomplete case, with its intra- and inter-case features serving as predictors and its remaining time as the target variable. A tabular machine learning model is then trained on this dataset. During inference, the queuing network is updated with new events, and the same feature extraction procedure is applied to ongoing cases to produce the runtime inference dataset, on which the trained model predicts remaining time. In the following, we describe the training procedure in detail and briefly outline the inference phase.

3.1 Event Log Augmentation

QTab takes an event log and augments each event with information required to construct a queuing network. The augmentation procedure involves three steps: (1) identifying triggering activities, (2) discovering roles, and (3) optionally estimating start timestamps. For all steps, QTab relies on established methods [5, 25, 29], which are briefly summarized below.

Event definitions. We assume an input event log L that consists of events represented as tuples $e = (c, a, t_s, t_c, r, D)$, where c denotes the case identifier, a the executed activity, t_s and t_c the start and completion timestamps, r the associated resource, and D a set of additional attribute–value pairs, covering both case-level attributes shared across all events of a case and event-level attributes specific to individual events. To construct an augmented event log L^*, this step extends each event with additional attributes, yielding a tuple $\epsilon = (c, a, a_e, t_s, t_c, t_e, r, r_e, \rho, \rho_e, D)$. Here, a_e is the triggering activity associated with the execution of a, identified as the most recent non-overlapping activity in the same case. The timestamp t_e denotes the completion time of a_e. The resources performing a and a_e are r and r_e, and their corresponding roles are given by ρ and ρ_e. We refer to attributes of ϵ using dot notation—for example, $\epsilon.t_c$ denotes the completion timestamp of the event ϵ. A trace is a sequence of events from the same case, ordered by completion time.

Deriving L^* from the event log L consists of the following steps:

Identifying the triggering activity. The queuing network captures resource interactions by modeling work handovers between resources, which must be inferred from the event log. Since activities may have nonzero duration, relying solely on completion timestamps imposes a strict total ordering that may obscure temporal overlap between activities [24]. QTab mitigates this issue by considering service-time intervals when identifying the *triggering activity* for each event. Following Wen et al. [29], for an event ϵ_j, we identify the triggering activity ϵ_i as the most recent event in the same case whose service-time interval $(\epsilon_i.t_s, \epsilon_i.t_c)$ does not overlap with that of ϵ_j. Once the triggering relationship is established (i.e., $\epsilon_j.a_e = \epsilon_i.a$), the triggering time is set as $\epsilon_j.t_e = \epsilon_i.t_c$, representing a work handoff between two resources: $\epsilon_j.r_e = \epsilon_i.r$.

Role discovery. QTab partitions resources by roles (i.e., resources with similar responsibilities) to achieve better scalability and computational efficiency. We augment each event by assigning a role ρ to its resource r, and a role ρ_e to the resource r_e that performed the triggering activity. A role represents a functional or organizational unit. When the event log does not contain role information, we apply a well-established algorithm [25] to group resources into roles based on their activity-execution profiles.

Estimating event start times. To separate waiting times from processing times and distinguish cases in service from those waiting in queues, QTab requires both start and completion timestamps. When start timestamps are unavailable, we estimate them using the approach proposed by Chapela-Campa and Dumas [5]. This approach accounts for process-level parallelism and incorporates resource availability to derive plausible start time estimates. The approach builds upon established heuristics for start-time estimation in process mining [10, 24, 30] and was specifically designed to improve the temporal realism of process execution data.

3.2 Queuing Network Creation

Using the augmented log L^*, we create a queuing network that models inter-case dependencies. We begin by defining a handoff graph, and then explain how it serves as the structural basis for construction of the queuing network.

Handoff graph. The handoff graph is a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where nodes $\mathcal{V}$ represent all unique resources in L^* and edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ capture transfers of work between them. An edge (r_s, r_t) exists when an activity performed by r_s is identified as the triggering activity of an activity performed by r_t, including self-handovers where $r_s = r_t$. Formally, $\mathcal{E} = \{(r_s, r_t) \mid r_s, r_t \in \mathcal{V}, \exists \epsilon_j \in L^* : \epsilon_j.r_e = r_s \wedge \epsilon_j.r = r_t\}$. To reduce the size of the handoff graph, resources can be grouped by role, so that nodes represent roles rather than individual resources and edges denote work handovers between roles. This flexible granularity reduces the feature space dimensionality and supports efficient training of tabular neural networks for remaining time prediction.

Queuing network. To build the queuing network $\mathcal{G}^*$, we replay L^* against its corresponding handoff graph $\mathcal{G}$, recording when each case arrives at and de-

parts from every node and edge. We use the term *queuing network* to denote a handoff graph augmented with queue-related state variables, rather than a classical analytical queuing network with probabilistic routing and service distributions. Consequently, nodes represent servers (individual resources or roles), while directed edges represent queues formed by work handovers between them. For an event ϵ_j processed at node r_t, the service time is $\epsilon_j.t_c - \epsilon_j.t_s$, while its waiting time on edge (r_s, r_t) is $\epsilon_j.t_s - \epsilon_j.t_e$. The queuing network $\mathcal{G}^*$ preserves the structure of $\mathcal{G}$ but is annotated with six time-dependent functions that are incrementally updated during the replay of L^*:

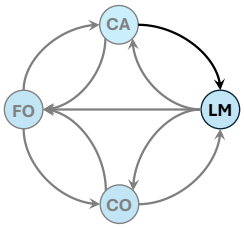
- *Workload Function*: $W(r, \mathcal{T})$ returns the number of active cases at node r at time $\mathcal{T}$, i.e., cases that have arrived but not yet departed.
- *Queue Function*: $Q((r_s, r_t), \mathcal{T})$ returns the number of cases waiting on edge (r_s, r_t) at time $\mathcal{T}$, i.e., those that have arrived but not yet departed.
- *Arrival Functions*: $\lambda_n(r, \mathcal{T}) = [\lambda_{n,f}(r, \mathcal{T})]_{f \in \{1,2,4,16,64,256,1024\}}$ computes moving averages of inter-arrival times over exponentially increasing window sizes f. For each f, $\lambda_{n,f}(r, \mathcal{T})$ is the mean time gap between the last f consecutive arrivals to node r occurring at or before time $\mathcal{T}$, characterizing the temporal intensity of arrivals at multiple granularities. The exponentially increasing windows capture both short-term fluctuations and longer-term temporal trends in congestion while keeping the number of features manageable. $\lambda_e((r_s, r_t), \mathcal{T})$ is defined analogously for edge (r_s, r_t).
- *Departure Functions*: Similarly, $\mu_n(r, \mathcal{T}) = [\mu_{n,f}(r, \mathcal{T})]_{f \in \{1,2,4,16,64,256,1024\}}$ computes moving averages of inter-departure times from node r up to $\mathcal{T}$, and $\mu_e((r_s, r_t), \mathcal{T})$ provides the corresponding measures for edge (r_s, r_t).

3.3 Feature Extraction

We generate multiple prefixes from each trace $\sigma =< \epsilon_1, \ldots, \epsilon_m >$ in the augmented log L^* to establish training samples. A prefix of length $k \in [1, m-1]$, denoted $\sigma^k = \langle \epsilon_1, \ldots, \epsilon_k \rangle$, represents the partial execution of σ up to its k-th event. Using L^* and its queuing network $\mathcal{G}^*$, we extract both intra- and inter-case features for these prefixes.

Inter-case features. For a prefix $\sigma^k = \langle \epsilon_1, \ldots, \epsilon_k \rangle$, inter-case features are extracted by querying the queuing network $\mathcal{G}^* = (\mathcal{V}, \mathcal{E})$ at the completion $(\epsilon_k.t_c)$, start $(\epsilon_k.t_s)$, and triggering $(\epsilon_k.t_e)$ times of its last event ϵ_k. At the completion timestamp $\epsilon_k.t_c$, these features quantify congestion across all nodes and edges. The *workload features* $W(\mathcal{V}, \epsilon_k.t_c) = \{W(r, \epsilon_k.t_c) \mid r \in \mathcal{V}\}$ capture the workload at each node, while the *queue features* $Q(\mathcal{E}, \epsilon_k.t_c) = \{Q((r_s, r_t), \epsilon_k.t_c) \mid (r_s, r_t) \in \mathcal{E}\}$ capture the number of cases waiting on each edge. We additionally incorporate the arrival and departure functions for the last visited node and edge. For the node r, the *arrival features* are $\lambda_n(r, \epsilon_k.t_s)$ and the *departure features* are $\mu_n(r, \epsilon_k.t_c)$, where $r = \epsilon_k.r$ or $\epsilon_k.\rho$, depending on whether the queuing network is defined over resources or roles. For the corresponding edge, the arrival and departure features are $\lambda_e((r_s, r_t), \epsilon_k.t_e)$ and $\mu_e((r_s, r_t), \epsilon_k.t_s)$, respectively, where $(r_s, r_t) = (\epsilon_k.r_e, \epsilon_k.r)$ or $(r_s, r_t) = (\epsilon_k.\rho_e, \epsilon_k.\rho)$ under the same convention.

We illustrate the extraction of inter-case features using a simplified loan application process. The process involves four roles: Front Office Officer (FO), Credit Analyst (CA), Compliance Officer (CO), and Loan Manager (LM), responsible for application registration, credit assessment, compliance checks, and final approval, respectively. Figure 2 shows a running example for a prefix of length four. Up to this point, the application has been registered, assessed by the Credit Analyst and Compliance Officer, and reviewed by the Loan Manager; subsequent events illustrate how the case may undergo an additional credit assessment before approval. The queuing network is queried at the completion, start, and triggering timestamps of the last event to obtain the workload of every role, the queue length of every handover, and the arrival and departure features for the last visited role (LM) and the triggering handover ($CA \rightarrow LM$).



k	a	ρ_e	ρ	t_e	t_s	t_c
1	Register Application	$\perp$	FO	2026-05-18 10:15	2026-05-18 10:15	2026-05-18 10:24
2	Evaluate Financial Risk	FO	CA	2026-05-18 10:24	2026-05-26 11:31	2026-06-01 16:42
3	Compliance Control	FO	CO	2026-05-18 10:24	2026-05-27 15:06	2026-05-29 09:17
4	**Review Assessment**	**CA**	**LM**	**2026-06-01 16:42**	**2026-06-02 14:19**	**2026-06-02 14:27**
5	Additional Credit Assessment	LM	CA	2026-06-02 14:27	2026-06-04 10:07	2026-06-04 13:21
6	Review Assessment	CA	LM	2026-06-04 13:21	2026-06-04 16:43	2026-06-04 17:08
7	Approve Application	LM	LM	2026-06-04 17:08	2026-06-04 17:08	2026-06-04 17:11
	Send Offer	LM	FO	2026-06-04 17:11	2026-06-05 09:06	2026-06-05 09:07

$\mu_n(LM, 2026\text{-}06\text{-}02\ 14:27) = \{16.8, 15.9, 14.7, 12.1, 9.6, 7.8, 6.9\}$

$W(\mathcal{V}, 2026\text{-}06\text{-}02\ 14:27) = \{FO: 4, CA: 6, CO: 8, LM: 2\}$

$\lambda_n(LM, 2026\text{-}06\text{-}02\ 14:19) = \{3.8, 4.5, 5.6, 8.9, 11.7, 14.2, 15.6\}$

$Q(\mathcal{E}, 2026\text{-}06\text{-}02\ 14:27) = \{FO \rightarrow CA: 5, CA \rightarrow FO: 0, FO \rightarrow CO: 4, CO \rightarrow FO: 0, CA \rightarrow LM: 2, LM \rightarrow CA: 0, CO \rightarrow LM: 2, LM \rightarrow CO: 1, LM: FO: 3\}$

$\mu_e(CA \rightarrow LM, 2026\text{-}06\text{-}02\ 14:19) = \{14.9, 14.2, 13.5, 11.4, 9.1, 7.5, 6.7\}$

$\lambda_e(CA \rightarrow LM, 2026\text{-}06\text{-}01\ 16:42) = \{5.2, 6.0, 7.4, 10.8, 13.9, 16.5, 18.1\}$

Fig. 2: Loan application example illustrating inter-case features for a prefix of length four. Arrival and departure features are reported in hours.

Intra-case features. For a prefix $\sigma^k = \langle \epsilon_1, \ldots, \epsilon_k \rangle$, intra-case features are extracted from its last event ϵ_k and the preceding execution history. To capture the *control-flow* perspective, we use the last five executed activities and the triggering activity $\epsilon_k.a_e$. Using a bounded history length provides a compact control-flow representation while avoiding excessive growth in feature dimensionality. From the *temporal* perspective, we include the last activity's processing time ($\epsilon_k.t_c - \epsilon_k.t_s$) and waiting time ($\epsilon_k.t_s - \epsilon_k.t_e$), cumulative processing and waiting times, elapsed time since case start, and the weekday and hour of $\epsilon_k.t_e$, $\epsilon_k.t_s$, and $\epsilon_k.t_c$. For the *resource* perspective, we include the resources ($\epsilon_k.r$, $\epsilon_k.r_e$) and roles ($\epsilon_k.\rho$, $\epsilon_k.\rho_e$) associated with the last activity and its triggering activity. Finally, all additional case- and event-level attributes from the last event $\epsilon_k.D$ capture the *data* perspective.

Tabular dataset creation. We concatenate the intra- and inter-case features to form a tabular training dataset, where each row represents a prefix encoded by d heterogeneous categorical and numerical features, with the remaining time as the target value. The dataset is denoted by a feature matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ and a target vector $\mathbf{y} \in \mathbb{R}^n$, where n is the number of prefixes.

3.4 Tabular Model: Training and Inference

Formulating remaining time prediction as a tabular modeling problem offers several advantages over deep neural networks for sequential data. First, tabular models are computationally and memory efficient, avoiding the overhead of recurrent or attention mechanisms through compact, pre-aggregated feature representations. Second, each feature corresponds to an interpretable property of the running case or the queuing dynamics, enabling explainability. Finally, tabular data supports a wide range of predictive models, from tree-based approaches to neural networks. Most existing inter-case-aware remaining time prediction methods rely on tree-based models [16, 19, 21, 22]. However, these models often achieve lower predictive accuracy than neural networks for sequential data [28], in part because pre-aggregated features can overlook temporal dependencies between past events.

To address this trade-off between efficiency and accuracy, we utilize a tabular neural network that leverages the expressive power of deep models while remaining efficient and scalable for large event streams. Specifically, we adapt the recently proposed deep tabular model, TabM [11], to remaining time prediction. The model is based on multilayer perceptrons (MLPs), where each row $x \in \mathbf{X}$ is processed through successive layers of linear operations (matrix multiplications) and nonlinear activation functions. TabM enhances MLPs through an efficient ensembling mechanism that uses a shared MLP backbone with multiple lightweight adapters and output heads. Each input is simultaneously processed by $\boldsymbol{\kappa}$ adapters, each generating a distinct encoding of the input to promote diversity. These representations are passed through the shared backbone, which improves regularization and mitigates overfitting. The resulting outputs are then mapped through $\boldsymbol{\kappa}$ parallel output heads to produce $\boldsymbol{\kappa}$ predictions, which are averaged at inference time. This design improves regularization, reduces overfitting, and lowers prediction variance.

To further reduce prediction variance, we train TabM in an 8-fold cross-validation pipeline. Essentially, **8** instances of the same model class are trained on subsets of $\frac{\mathbf{7}}{\mathbf{8}}$ of the data, while the remaining $\frac{\mathbf{1}}{\mathbf{8}}$ are used for early stopping. At inference time, the predictions of the **8** models are averaged. This procedure is frequently used to improve predictive performance in Kaggle competitions [26] and has been shown to be important to make neural networks competitive on tabular data [9, 27]. The choice of **8** folds reflects a trade-off between predictive performance and computational cost, as larger ensembles generally improve robustness at the expense of increased training and inference time.

Inference with QTab. As we incrementally construct the queuing network by replaying the augmented event log, its functions (workload, queue, arrival, and departure) are continuously updated during inference as new events occur. This allows QTab to capture the current system state through inter-case features using only the information available up to the latest executed event. The resulting intra- and inter-case features are then provided to the trained TabM model to predict the remaining time of each running case.

4 Evaluation

This section presents the experiments conducted to evaluate the performance of QTab, including the experimental setup, and results.

4.1 Experimental Setup

We describe the event logs and experimental setup used to evaluate QTab. The event logs, complete implementation of QTab (including event-log augmentation, preprocessing, feature extraction, model training and evaluation, configurations, and baseline implementations), are publicly available in the project repository.[1]

Event logs. Table 1 presents the eight real-world event logs used in our evaluation. For each log, cases are sorted chronologically by their start time. The first 80% are used for training and the remaining 20% for test. Within the training set, 20% of cases serve as a validation set.

Table 1: Characteristics of the event logs used for evaluation. Average lead time (ALT) is reported in days.

Event log	Cases	Events	Activities	Resources	Variants	ALT	Domain
P2P	608	9119	21	27	70	21.5	Procurement
BPIC17W	30276	240854	8	148	6356	12.7	Finance
BPIC20ID	6449	72151	34	8	753	86.5	Education
BPIC20DD	10500	56437	17	7	99	11.5	Education
BPIC20PTC	2099	18246	29	8	202	36.8	Education
BPIC15-1	1199	52217	398	23	1170	95.9	Government
BPIC15-5	1156	59083	389	22	1153	98.0	Government
Helpdesk	4580	21348	14	22	226	40.9	IT Services

Benchmark approaches. We compare our approach against three others:

- *PGTNet* [2]: A deep learning approach that transforms event logs into graph datasets and integrates graph neural networks with transformers to model control-flow relations and temporal dependencies within each case. Its use of inter-case features, is limited to the overall system workload, making it a strong deep learning baseline with only coarse-grained inter-case information.
- *LS-ICE* [13]: A deep learning approach that employs load state encoding and long short-term memory (LSTM) neural networks. Compared to PGTNet, it incorporates more fine-grained inter-case features, capturing workloads at performance-critical activities. Since the performance difference across its variants is minor, we use the system-based variant.
- *Congestion graphs* [21]: An inter-case-aware approach that extracts activity-level workloads and inter-arrival times to create a tabular dataset for training an XGBoost model. Compared to LS-ICE and PGTNet, it leverages more granular inter-case features, and relies on tree-based machine learning.

[1] https://github.com/keyvan-amiri/QTab

To ensure a fair comparison, QTab and all three baselines use the same intra-case features; thus, performance differences depend solely on how inter-case information is modeled and on the choice of predictive model. All models—except PGTNet, which only supports prefixes of length two or more—are trained on prefixes of length one to $\boldsymbol{m-1}$, where $\boldsymbol{m}$ is the case length. For prefixes of length 1, we use a simple heuristic for PGTNet, predicting the median remaining time of training prefixes of the same length. For all three baselines, we adopt the configurations recommended in the original papers [2, 13, 21].

QTab configuration. To reduce the number of nodes in the queuing network, roles were used as nodes for the BPIC17W log, which contains substantially more resources than the remaining event logs, while resources were used for all other logs. In the BPIC15 logs, infrequent resources (involved in less than 1% of activities) were grouped under "Others". The preprocessing of tabular features follows the standard pipeline implemented in TabArena [9] with some additions. Numerical features were normalized with scikit-learn's quantile transformer, with missing values imputed by the mean. Categorical features were one-hot encoded, with missing values treated as a separate category. The target variable (remaining time) was log-transformed and standardized to zero mean and unit variance during training and rescaled to its original scale for evaluation. For both, TabM and XGBoost models, we use the default hyperparameters in TabArena [9].

Training setup. TabM is trained using Mean Squared Error and the Adam optimizer for up to 200 epochs, with early stopping patience of 5 epochs. The validation set is used exclusively for early stopping during training. Due to temporal and concept shifts in the data, hyperparameter tuning carries a high risk of overfitting. Therefore, we do not conduct hyperparameter optimization. Instead, as described in Section 3, we rely on heavy ensembling of one configuration to improve performance.

Evaluation metrics. We use Mean Absolute Error (MAE) to evaluate prediction error due to its robustness against outliers [28]. In addition, we report the normalized MAE (nMAE) [20], which contextualizes performance relative to a simple baseline. nMAE is defined as

$$\text{nMAE} = \frac{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}_{\text{test}}} |\boldsymbol{y_i} - \boldsymbol{\hat{y}_i}|}{\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}_{\text{test}}} |\boldsymbol{y_i} - \boldsymbol{\tilde{y}_{train}}|}, \tag{1}$$

where $\boldsymbol{y_i}$ and $\boldsymbol{\hat{y}_i}$ denote the true and predicted remaining time of prefix $\boldsymbol{i}$, respectively, and $\tilde{\boldsymbol{y}}_{\text{train}}$ is the median remaining time in the training set. An nMAE of 1 corresponds to the baseline that always predicts $\bar{\boldsymbol{y}}_{\text{train}}$. Values below 1 indicate that the model outperforms this baseline, while values above 1 indicate worse predictive performance and limited generalization [20].

4.2 Experimental Results

Table 2 reports the mean and standard deviation of the MAE for each remaining time prediction approach across the eight event logs, along with the corresponding average nMAE values computed over five runs with different random seeds.

Table 2: Average MAE in days, reported as mean with standard deviation, and nMAE of remaining time prediction approaches, computed over five runs with different random seeds. Congestion graph is abbreviated as CG.

Event log	MAE				nMAE			
	PGTNet	LS-ICE	CG	QTab	PGTNet	LS-ICE	CG	QTab
P2P	14.4 (0.5)	23.1 (0.4)	**13.8 (0.0)**	14.1 (0.0)	0.68	1.09	**0.65**	0.66
BPIC17W	**5.8 (0.0)**	7.0 (0.2)	8.0 (0.4)	**5.8 (0.0)**	**0.86**	1.04	1.18	**0.86**
BPIC20ID	14.7 (0.7)	22.8 (0.4)	13.6 (0.1)	**12.6 (0.0)**	0.75	1.17	0.69	**0.63**
BPIC20DD	4.1 (0.0)	6.1 (0.0)	4.4 (0.0)	**4.0 (0.0)**	0.84	1.25	0.90	**0.83**
BPIC20PTC	9.1 (0.6)	11.9 (0.3)	8.8 (0.0)	**8.1 (0.0)**	0.92	1.21	0.89	**0.81**
BPIC15-1	22.0 (1.4)	32.8 (2.1)	22.4 (0.9)	**20.0 (0.1)**	0.74	1.10	0.75	**0.68**
BPIC15-5	38.3 (8.5)	36.2 (7.8)	30.6 (1.2)	**25.4 (0.3)**	0.96	0.91	0.77	**0.64**
Helpdesk	**5.7 (0.5)**	18.7 (0.3)	9.9 (0.3)	6.8 (0.1)	**0.48**	1.59	0.84	0.57
Win Rate	2/8	0/8	1/8	**6/8**	2/8	0/8	1/8	**6/8**
Average					0.78	1.17	0.83	**0.71**

QTab achieves the highest predictive accuracy on six of the eight logs. On average, QTab attains the lowest nMAE of 0.71, compared to 0.78 for the next-best approach, PGTNet.

Statistical significance. To evaluate whether the observed differences in predictive performance between models are statistically significant, we compare all pairs of models using the Wilcoxon signed-rank test on nMAE values (Table 3). The superior performance of QTab is significant for the comparison to congestion graphs and LS-ICE. The better average performance compared to PGTNet is not statistically significant at the conventional 95% confidence level ($\boldsymbol{p = 0.11}$). Since the comparison is based on eight event logs, statistical significance tests have limited power. We therefore additionally report effect sizes and median performance differences. As shown in Table 3, both measures consistently favor QTab, indicating that its empirical improvements over PGTNet remain practically meaningful despite the absence of statistical significance at the 95% level.

Table 3: Wilcoxon signed-rank test results for pairwise model comparisons on nMAE. Columns report the test statistic (W), p-value, median difference in nMAE, and effect size (rank-biserial correlation).

Comparison	Test Statistic (W)	p-value	Median Difference	Effect Size
QTab > LS-ICE	0.0	0.008	0.421	1.00
QTab > Congestion graph	1.0	0.016	0.075	0.75
QTab > PGTNet	6.0	0.109	0.041	0.50
Congestion graph > PGTNet	16.0	0.844	-0.007	0.00
Congestion graph > LS-ICE	2.0	0.023	0.348	0.75
PGTNet > LS-ICE	1.0	0.016	0.385	0.75

Contributions of inter-case features and TabM model. We conduct an ablation study to disentangle the contributions of inter-case features and model

Table 4: nMAE for remaining time prediction: average and standard deviation across five random seeds. Congestion graph is abbreviated as CG.

Event log	XGBoost			TabM		
	CG	QTab	Combined	CG	QTab	Combined
P2P	0.65 (0.00)	**0.63 (0.00)**	0.65 (0.00)	**0.62 (0.00)**	0.66 (0.00)	0.66 (0.00)
BPIC17W	1.18 (0.06)	**0.87 (0.00)**	0.89 (0.01)	0.87 (0.00)	**0.86 (0.00)**	**0.86 (0.00)**
BPIC20ID	0.69 (0.01)	0.65 (0.00)	**0.64 (0.00)**	0.60 (0.00)	0.63 (0.00)	**0.59 (0.00)**
BPIC20DD	0.90 (0.00)	0.84 (0.00)	**0.83 (0.00)**	0.83 (0.00)	0.83 (0.00)	**0.81 (0.00)**
BPIC20PTC	0.89 (0.00)	0.86 (0.00)	**0.83 (0.00)**	0.81 (0.00)	0.81 (0.00)	**0.80 (0.00)**
BPIC15-1	0.75 (0.03)	0.79 (0.03)	**0.74 (0.01)**	0.69 (0.00)	**0.68 (0.00)**	**0.68 (0.00)**
BPIC15-5	0.77 (0.03)	**0.62 (0.01)**	0.73 (0.02)	**0.57 (0.01)**	0.64 (0.01)	**0.57 (0.01)**
Helpdesk	0.84 (0.03)	0.64 (0.02)	**0.59 (0.01)**	**0.53 (0.00)**	0.57 (0.01)	0.56 (0.01)
Average	0.83 (0.02)	0.74 (0.01)	0.74 (0.01)	**0.69 (0.00)**	0.71 (0.00)	**0.69 (0.00)**

capacity to QTab's strong performance. We train both XGBoost and TabM models using inter-case features derived from QTab, congestion graphs, or their combination (Table 4). Since all feature sets share identical intra-case features, differences in performance directly reflect the impact of inter-case features and the expressive power of the predictive model.

Except for BPIC15-1, XGBoost achieves higher accuracy with QTab features than with congestion graphs, with the largest gains on BPIC17W, Helpdesk, and BPIC15-5. A Wilcoxon signed-rank test on nMAE confirms that these improvements are statistically significant ($\boldsymbol{p = 0.05}$). This observation is consistent with the limited representational capacity of traditional tree-based models such as XGBoost, whose performance is constrained by parameters such as maximum tree depth, making them more sensitive to the quality of inter-case features. In contrast, for TabM, the performance gap between the two feature sets is much smaller, and no statistically significant difference is observed between them. The higher expressive capacity of TabM enables it to learn more complex feature interactions, making its predictive performance less sensitive to the choice of inter-case feature representation. Nevertheless, QTab's resource-centric queuing features remain competitive with congestion graph features under TabM, demonstrating that they continue to provide useful predictive information even when paired with highly expressive tabular neural networks.

Combining both feature sets does not always improve accuracy, but for some event logs (e.g., BPIC20ID, BPIC20DD, and BPIC20PTC), it yields lower MAE, indicating that workloads and inter-arrival patterns inferred from activity transitions and resource handovers can be complementary.

Efficiency and performance trade-off. We evaluate QTab's scalability in feature extraction and its efficiency during training and inference by comparing its runtime to baseline approaches. Training time encompasses both constructing the training dataset from the event log and fitting the model, while inference time includes feature extraction and predicting the remaining time for each prefix. All experiments were executed on identical hardware (NVIDIA RTX A6000 GPU) to ensure fairness in runtime comparisons.

Figure 3 illustrates the trade-off between predictive performance (y-axis, measured by nMAE) and computational cost on a log-scale (x-axis), shown separately for training time and inference time.

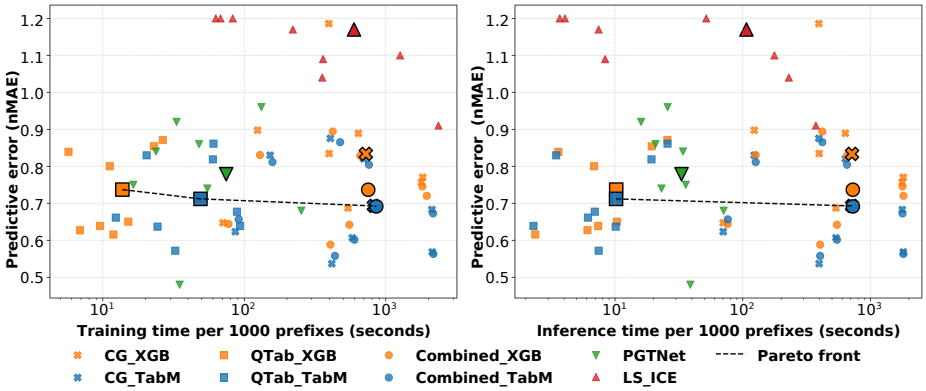


Fig. 3: Trade-off between performance (nMAE) and runtime for training (left) and inference (right). Each point shows a model's performance on a single event log. The larger points represent the average across all logs, normalized to 1000 samples prior to averaging. Points closer to the lower-left region of each subplot represent models that achieve both low error and low runtime; the dashed line highlights the Pareto front.

Notably, all Pareto-dominant points are approaches utilizing a tabular representation demonstrating the strong performance vs. efficiency trade-off compared to deep learning approaches for sequential data. XGBoost using our queue features is the most training time efficient approach. Using TabM further improves the performance at the cost of an increased training time. While better performance can be achieved using CG features or a combination of both feature sets, this comes with a severe increase in training and particularly inference time. This overhead stems primarily from the expensive feature extraction, which generates large congestion graphs for processes with many activities, such as a graph with 398 nodes and 4,807 edges in the BPIC15-1 log. Extracting multiple features for every node and edge is therefore computationally intensive. QTab avoids this complexity through flexible aggregation of organizational entities (e.g., collapsing activities to roles), enabling a much more efficient feature extraction process. Our results demonstrate that both of our proposed innovations, using queuing features as well as a tabular neural network, are crucial for practical applicability. The resource-centric structure of the underlying queuing network greatly improves the efficiency compared to previous approaches, while using a tabular neural network substantially improves predictive performance.

Limitations. While QTab enables the extraction of fine-grained yet scalable inter-case features, it still relies on several design choices that introduce limitations. First, although batch processing effects are implicitly reflected in arrival and departure features, QTab currently offers no explicit feature engineering support for such mechanisms. For example, in a logistic process where dispatching depends on both resource availability and product readiness, the manufacturer may delay dispatching until a minimum batch size is reached. Second, our approach relies on reconstructed start timestamps when event logs contain only completion times. Although we employ a state-of-the-art repair approach [5], inaccuracies in reconstructed timestamps may still influence workload and queue-related features. Evaluating the sensitivity of QTab to reconstructed start timestamps remains a direction for future work. Finally, our empirical findings are based on eight event logs; while they provide diverse evidence and consistently favor QTab in average performance, broader generalization would benefit from experiments on additional datasets and from further investigating the trade-offs between resource- and role-based queuing networks.

5 Conclusion

By integrating queuing theory, process mining, and modern tabular machine learning into a unified framework, this work advances the broader goal of efficient and accurate predictive process monitoring. To this end, we introduced QTab, a novel approach for inter-case-aware remaining time prediction in business processes. QTab models resource interactions using a queuing network, from which it derives fine-grained inter-case features. By explicitly capturing congestion at resource-to-resource handovers, it addresses a key limitation of existing approaches—such as congestion graphs, load state encoding, and performance spectrum—that largely neglect the role of resource interactions. QTab utilizes a neural network ensemble for remaining time prediction, eliminating reliance on traditional tree-based models while retaining the scalability and efficiency of tabular encoding. Experiments on eight real-world event logs show that QTab achieves competitive-to-superior predictive accuracy compared to state-of-the-art tree-based and deep learning approaches, while offering improved scalability and efficiency.

This work opens several avenues for future research. Because the queuing network is updated continuously as events occur, QTab naturally supports predictive analytics over event streams and may help mitigate the impact of concept drift. The diversity among QTab's neural network ensemble members could also be leveraged for uncertainty-aware remaining time prediction, yielding prediction intervals rather than point predictions. Furthermore, its explicit modeling of congestion at resource handovers provides a foundation for explainable remaining time prediction by exposing operational bottlenecks and resource interactions. Finally, incorporating alternative tabular neural networks, such as TabPFN [14] or RealMLP [15], offers a promising direction for assessing how different models affect the accuracy of remaining time prediction.

References

1. Ali, M.A., Milani, F., Dumas, M.: Data-driven identification and analysis of waiting times in business processes: Ma ali et al. Business & Information Systems Engineering **67**(2), 191–208 (2025)
2. Amiri Elyasi, K., van der Aa, H., Stuckenschmidt, H.: PGTNet: A process graph transformer network for remaining time prediction of business process instances. In: CAISE. pp. 124–140. Springer (2024)
3. Amiri Elyasi, K., van der Aa, H., Stuckenschmidt, H.: A simple and calibrated approach for uncertainty-aware remaining time prediction. In: International Conference on Business Process Management. pp. 217–234. Springer (2025)
4. Ceravolo, P., Comuzzi, M., De Weerdt, J., Di Francescomarino, C., Maggi, F.M.: Predictive process monitoring: concepts, challenges, and future research directions. Process Science **1**(1), 2 (2024)
5. Chapela-Campa, D., Dumas, M.: Repairing activity start times to improve business process simulation. arXiv preprint arXiv:2208.12224 (2022)
6. Denisov, V., Fahland, D., van der Aalst, W.M.: Unbiased, fine-grained description of processes performance from event data. In: International Conference on Business Process Management. pp. 139–157. Springer (2018)
7. Denisov, V., Fahland, D., van der Aalst, W.M.: Predictive performance monitoring of material handling systems using the performance spectrum. In: ICPM. pp. 137–144 (2019)
8. Di Francescomarino, C., Ghidini, C.: Predictive process monitoring. In: Process mining handbook, pp. 320–346. Springer (2022)
9. Erickson, N., Purucker, L., Tschalzev, A., Holzmüller, D., Desai, P., Salinas, D., Hutter, F.: Tabarena: A living benchmark for machine learning on tabular data. Advances in Neural Information Processing Systems **38** (2026)
10. Fracca, C., de Leoni, M., Asnicar, F., Turco, A.: Estimating activity start timestamps in the presence of waiting times via process simulation. In: International Conference on Advanced Information Systems Engineering. pp. 287–303. Springer (2022)
11. Gorishniy, Y., Kotelnikov, A., Babenko, A.: Tabm: Advancing tabular deep learning with parameter-efficient ensembling. In: International Conference on Learning Representations. vol. 2025, pp. 77899–77935 (2025)
12. Grinvald, A., Soffer, P., Mokryn, O.: Inter-case properties and process variant considerations in time prediction: A conceptual framework. In: BPMDS. pp. 96–111. Springer (2021)
13. Gunnarsson, B.R., vanden Broucke, S., De Weerdt, J.: LS-ICE: A load state inter-case encoding framework for improved predictive monitoring of business processes. Information Systems **125**, 102432 (2024)
14. Hollmann, N., Müller, S., Purucker, L., Krishnakumar, A., Körfer, M., Hoo, S.B., Schirrmeister, R.T., Hutter, F.: Accurate predictions on small data with a tabular foundation model. Nature **637**(8045), 319–326 (2025)
15. Holzmüller, D., Grinsztajn, L., Steinwart, I.: Better by default: Strong pre-tuned mlps and boosted trees on tabular data. Advances in Neural Information Processing Systems **37**, 26577–26658 (2024)
16. Klijn, E.L., Fahland, D.: Identifying and reducing errors in remaining time prediction due to inter-case dynamics. In: ICPM. pp. 25–32 (2020)
17. Kraus, A., Amiri Elyasi, K., van der Aa, H.: On the use of steady-state detection for process mining: Achieving more accurate insights. In: CAISE. Springer (2025)

18. Lashkevich, K., Milani, F., Chapela-Campa, D., Suvorau, I., Dumas, M.: Unveiling the causes of waiting time in business processes from event logs. Information Systems **126**, 102434 (2024)
19. Pourbafrani, M., Kar, S., Kaiser, S., van der Aalst, W.M.P.: Remaining time prediction for processes with inter-case dynamics. In: Process Mining Workshops. pp. 140–153. Springer (2022)
20. Roider, J., Nguyen, A., Zanca, D., Eskofier, B.M.: Assessing the performance of remaining time prediction methods for business processes. IEEE Access (2024)
21. Senderovich, A., Beck, J.C., Gal, A., Weidlich, M.: Congestion graphs for automated time predictions. AAAI **33**(1), 4854–4861 (2019)
22. Senderovich, A., Di Francescomarino, C., et al.: Intra and inter-case features in predictive process monitoring: A tale of two dimensions. In: Business Process Management. pp. 306–323. Springer (2017)
23. Senderovich, A., Francescomarino, C.D., Maggi, F.M.: From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring. Information Systems **84**, 255–264 (2019)
24. Senderovich, A., Leemans, S.J., Harel, S., Gal, A., Mandelbaum, A., van der Aalst, W.M.: Discovering queues from event logs with varying levels of information. In: International Conference on Business Process Management. pp. 154–166. Springer (2016)
25. Song, M., van der Aalst, W.M.P.: Towards comprehensive support for organizational mining. Decision Support Systems **46**(1), 300–317 (2008)
26. Tschalzev, A., Marton, S., Lüdtke, S., Bartelt, C., Stuckenschmidt, H.: A data-centric perspective on evaluating machine learning models for tabular data. Advances in Neural Information Processing Systems **37**, 95896–95930 (2024)
27. Tschalzev, A., Purucker, L., Lüdtke, S., et al.: Unreflected use of tabular data repositories can undermine research quality. In: The Future of Machine Learning Data Practices and Repositories at ICLR 2025 (2025)
28. Verenich, I., Dumas, M., et al.: Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. ACM Trans. Intell. Syst. Technol. **10**(4), 34:1–34:34 (2019)
29. Wen, L., Wang, J., van der Aalst, W.M., Huang, B., Sun, J.: A novel approach for process mining based on event types. Journal of Intelligent Information Systems **32**(2), 163–190 (2009)
30. Wombacher, A., lacob, M.E.: Start time and duration distribution estimation in semi-structured processes. In: Proceedings of the 28th annual ACM symposium on applied computing. pp. 1403–1409 (2013)