Kirchdorfer *et al. Process Science* (2026) 3:9
https://doi.org/10.1007/s44311-026-00041-z

Process Science

RESEARCH Open Access

Arrival times in dynamic environments: modeling, evaluation, and benchmarking for business process simulation

Lukas Kirchdorfer[1,2*†], Konrad Özdemir[1*†], Han van der Aa[3] and Heiner Stuckenschmidt[1]

†Lukas Kirchdorfer and Konrad Özdemir contributed equally to this work.

*Correspondence:
Lukas Kirchdorfer
lukas.kirchdorfer@sap.com
Konrad Özdemir
konrad.oezdemir@uni-mannheim.de
[1]Data and Web Science Group, University of Mannheim, Mannheim, Germany
[2]SAP Signavio, Walldorf, Germany
[3]Faculty of Computer Science, University of Vienna, Vienna, Austria

Abstract

Business Process Simulation (BPS) plays a central role in analyzing and improving organizational processes by estimating the effects of potential changes. A key element of any BPS model is the case-arrival component, which determines when new cases enter the process. While accurate arrival time modeling is essential for producing reliable simulations, most existing approaches rely on static inter-arrival time distributions that overlook the temporal dynamics inherent in organizational environments, resulting in reduced accuracy and misleading insights. To address this, we propose *Auto Time Kernel Density Estimation* (AT-KDE), a scalable arrival time modeling approach that captures global trends, weekday effects, and intraday shifts. Across 20 diverse processes, our experiments show that AT-KDE produces more accurate and robust arrival times than existing case-arrival modeling approaches, while maintaining practical execution times. Moreover, we assess how different arrival modeling approaches affect overall simulation quality. Noting that existing BPS evaluation metrics may not explicitly account for process dynamics, we additionally showcase a novel utility-based evaluation framework, which we then use in our experiments.

Keywords Process simulation, Time series segmentation, Kernel density estimation

Introduction

Business Process Simulation (BPS) is a central instrument for supporting the redesign of organizational processes and their underlying information systems (van der Aalst 2015). By enabling the creation of *digital process twins* (Dumas 2021), BPS allows organizations to estimate the impact of proposed process changes, such as introducing a new information system or deploying a revised process version on key performance indicators (e.g., cycle time). Providing these estimates upfront can significantly enhance the efficiency of redesign initiatives and reduce associated risks for decision-makers (Dumas et al. 2013). The usefulness of BPS, however, critically depends on the availability of simulation models that faithfully reproduce real-world process behavior, as only accurate models can generate reliable insights into the consequences of redesign decisions. Because manually

Springer

constructing such models is both time-consuming and error-prone (van der Aalst 2015), a range of automated approaches has emerged to derive BPS models directly from historical execution data recorded in event logs (Rozinat et al. 2009; Camargo et al. 2020, 2022; Meneghello et al. 2023; Kirchdorfer et al. 2024). These data-driven approaches typically discover a process model and enrich it with simulation parameters, such as a case-arrival model and activity processing time distributions.

In this work, we argue that existing data-driven BPS approaches exhibit a significant limitation in how they model the arrival of new cases—a factor known to critically influence simulation outcomes (Martin et al. 2015). Most current approaches rely on static probability distributions to represent case arrivals. However, such static models are inherently time-invariant and thus unable to capture dynamic organizational environments in which arrival rates fluctuate over time. As a result, they fail to adapt to changes driven by global seasonal trends, local weekday effects, or intraday demand patterns. Consider, for example, patient arrivals in a hospital emergency department: volumes typically rise during flu season, decrease on weekends, and peak in the early morning compared to late at night. A case-arrival model that cannot reflect these variations will produce simulations that misrepresent the true workload of the system. This, in turn, can lead to unrealistic estimates of key performance indicators such as resource utilization, waiting times, and overall cycle times.

We therefore propose *Auto Time Kernel Density Estimation* (AT-KDE), an approach for modeling arrival times in BPS that explicitly accounts for temporal dynamics. AT-KDE combines a divide-and-conquer strategy with Kernel Density Estimation (KDE) (Parzen 1962) to iteratively segment the arrival time series and learn a non-parametric distribution for each segment, capturing variability at multiple temporal scales. This enables precise and scalable arrival-time modeling that reflects global dynamics (e.g., seasonalities and drifts), day-of-week effects, and intraday patterns—factors that most existing methods overlook. Using the standard log-to-log comparison framework adopted in our first evaluation on 20 real-life processes, we show that AT-KDE is considerably more accurate and robust than current arrival modeling approaches used in BPS and baselines from other domains.

Moreover, we assess how different arrival modeling approaches affect overall simulation quality. To do so, we adopt a utility-based evaluation framework (Özdemir et al. 2025) that evaluates simulated logs by their usefulness in downstream predictive process monitoring tasks, rather than by log-to-log similarity. This makes the framework particularly suitable for judging how well BPS models capture temporal dynamics. We first validate it on a synthetic loan-application process, showing that it penalizes misspecified models in the expected process perspectives, and then apply it to compare arrival modeling approaches on real event logs. Across these logs, AT-KDE consistently yields the highest downstream utility, demonstrating its practical value for simulation.

This paper presents an extended and revised version of our earlier work (Kirchdorfer et al. 2025), in which we introduced the first version of the AT-KDE approach. The present study advances that work in three key directions. First, we expand the arrival-generation mechanism so that AT-KDE can begin simulating arrivals from any point in time, whereas the initial version was restricted to do so on *future* arrivals only. Second, we conduct a comprehensive experiment by integrating AT-KDE as the arrival component within a full BPS model, enabling the simulation of complete event logs rather than

merely generating arrival timestamps. Third, to evaluate the quality of these simulations, we showcase and adopt a new utility-based evaluation framework that assesses the usefulness of simulated data through their performance in downstream predictive tasks.

The remainder of this manuscript is structured as follows. Section "Related work" discusses related work, and Sect. "Motivation" motivates our approach. Section "Approach" describes our approach for modeling and simulating arrival times, and Sect. "Experiments and Results" presents a thorough experimental evaluation. Finally, Sect. "Conclusion" concludes the paper.

Related work

This section reviews work on modeling arrivals in BPS and in other domains.

Approaches Applied in BPS Arrival modeling in BPS, so far, predominantly employs static approaches, with only one dynamic alternative.

Static Approaches Case arrivals are typically simulated by sampling from a fitted probability distribution of inter-arrival times and cumulatively adding those samples to attain the concrete case's arrival timestamps. The respective inter-arrival times are typically a priori assumed to follow an *exponential* distribution with constant rate (van der Aalst 2015), as seen in the simulation model of Rozinat et al. (2009). Similarly, Martin et al. (2015) adopt the gamma distribution—a generalization of the exponential. A more recent approach by Camargo et al. (2020) relaxes the a priori selection by allowing for the discovery of a best-fitting distribution from a predefined set of distributions. This method uses the earth mover's distance (Villani 2008) to identify the candidate distribution that best fits the empirical inter-arrival time data, potentially capturing more complex patterns than the exponential model. This has also been adopted by other BPS approaches (López-Pintado and Dumas 2022; Kirchdorfer et al. 2024).

Dynamic Approaches Differently to fitting a distribution on inter-arrival times, two BPS approaches (Camargo et al. 2022; Meneghello et al. 2023) use *Prophet* (Taylor and Letham 2018) to decompose time series into trend, seasonality, holidays, and error components. It is used to predict hourly arrival counts, which are then uniformly distributed within each hour to form timestamps. Although it improves upon static approaches, *Prophet* has notable limitations for process simulation, as we will discuss in Sect. 3.

Approaches Used Beyond BPS The simulation of process arrivals can also be viewed as a (dynamic) time series modeling problem, relevant in many scenarios beyond BPS, with approaches typically categorized into *Classical* and *ML Models*.

Classical Models Classical approaches provide fast and interpretable baselines for modeling aggregated arrival dynamics (e.g., daily counts or rates), and are therefore a natural reference point for our setting. A prominent example is the *Autoregressive Integrated Moving Average* (ARIMA) family, which forecasts future values from past observations and error terms (Hamilton 1994). While computationally efficient and widely used across domains (Kontopoulou et al. 2023), ARIMA typically requires diagnostics and specification choices (e.g., differencing and intervention handling) to remain robust under structural changes, which limits its out-of-the-box applicability across heterogeneous event logs. Closely related are exponential smoothing methods in state-space form (ETS). Since ETS and ARIMA overlap under additive-error state-space formulations, we treat ARIMA as a representative model in this class. Moreover, multiplicative ETS variants can be ill-suited in the presence of zero-arrival periods, which are common

in process event logs. A second baseline class comprises nonhomogenous Poisson point-process models (NPP), which typically describe arrivals through an explicit time-varying intensity $\lambda(t)$. We consider the *Hawkes process* as a widely used self-exciting representative that captures temporal clustering and is common in queuing systems (Daw and Pender 2018). Notably, though, standard Hawkes specifications rely on a fixed parametric kernel and time-invariant parameters unless explicitly extended.

ML Models In recent years, machine learning models have emerged as a powerful complement to classical approaches by directly modeling complex relationships in temporal data. The *Long Short-Term Memory* (LSTM) network, a recurrent neural network variant, is widely used to capture long-term dependencies and forecast events in environments with complex temporal dynamics (Lindemann et al. 2021). Ensemble methods such as XGBoost have also been applied effectively to arrival time forecasting, offering competitive performance by exploiting non-linear interactions in the data (Porto and Fogliatto 2024). More recently, frameworks like Chronos, which build upon advances in foundation time series modeling, have been proposed to leverage large-scale data and pre-trained representations (Ansari et al. 2024).

Our experiments will show that static methods ignore pattern variability and dynamic methods lack robustness and incur high computational costs; in contrast, AT-KDE captures process dynamics both efficiently and robustly.

Motivation

In this section, we highlight the critical impact of case arrivals on the accuracy of BPS outcomes. Using a loan application process as an example, we examine how different types of dynamics influence arrival patterns and complicate the extraction of a reliable arrival model from process data. We then show that existing arrival modeling approaches used in BPS fail to account for these dynamics, leading to inaccurate arrival estimates and, thus, poor simulation accuracy.

Understanding Arrival Behavior To illustrate the intricate complexities within arrival data, we consider a loan application process (Dumas et al. 2013, Chap. 10.8) during a period of changing interest rates (data available in Kirchdorfer et al. (2025)) and focus on arrivals of new applications, visualized in Fig. 1. The process comprises 12 activities, beginning with *Check application form completeness*. Its control-flow structure includes a loop, a parallel branch involving three activities, three exclusive gateways, and three possible end points: *Approve application*, *Reject application*, and *Cancel application*. In total, the process is carried out by 19 distinct resources. Figure 2 provides a

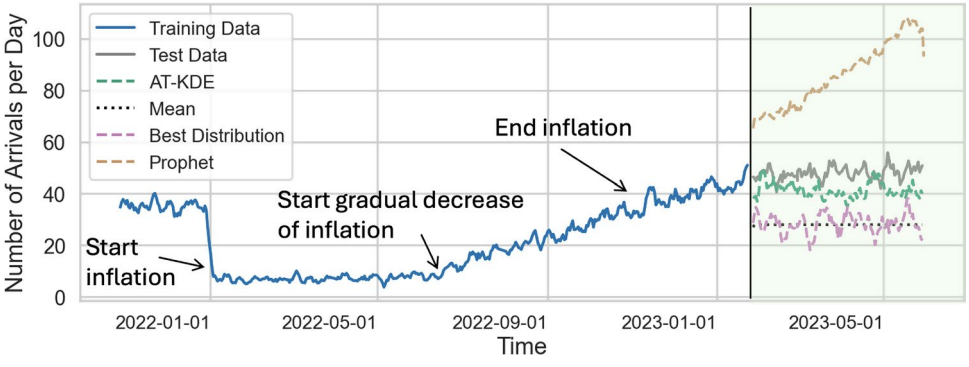


Fig. 1 7-Day rolling average of arrival count in the loan application process

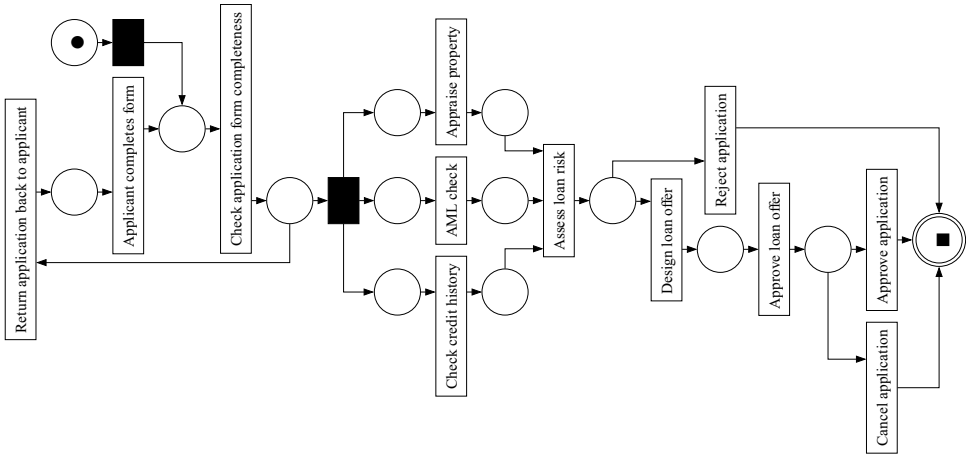


Fig. 2 Petri net describing the loan application process

visual representation of the control-flow. The underlying data was synthetically generated.[1] Within this scenario, we can expect various types of dynamics leading to variations in arrival rates over time:

- *Intraday Dynamics.* The arrival pattern can change over the course of a single day. For instance, customers may be more likely to submit loan applications in the morning, with fewer arrivals in the evening or overnight.
- *Weekday Dynamics.* The arrival pattern can vary across the days of the week. For instance, loan applications may peak on Mondays as customers submit documents prepared over the weekend, decline gradually during the week, and drop on weekends—especially if applications require in-person visits.
- *Global Dynamics.* Broader trends and external factors can alter arrival rates over extended periods. In our scenario, rising interest rates driven by inflation due to a market crash may lead to a sudden decline in loan applications, followed by a gradual recovery as economic conditions stabilize, as reflected by the daily arrival counts in Fig. 1.

Note that we define global dynamics to include all variations beyond weekly patterns, such as seasonal trends (e.g., peaks at the beginning of the month), long-term drifts (e.g., evolving financial policies), and one-off events (e.g., economic shocks). While some of these changes are predictable, others are sudden and disruptive. By explicitly modeling even one-off events, we separate their impact from recurring patterns, leading to more accurate future arrival estimates without distortions by past anomalies.

Furthermore, note that we explicitly distinguish these three types of dynamics (also as part of our approach in Sect. 4) because they reflect different and largely independent drivers of arrival behavior in organizational processes. Intraday and weekday dynamics arise from recurring operational routines and calendars that are present across domains

[1] The running example is based on a synthetically generated event log obtained from a discrete-event simulation of the loan application process. Arrival behavior was constructed by combining intraday, weekday, and global dynamics. Intraday dynamics were introduced by partitioning each day into fixed time windows (between 8am and 8pm) with different arrival intensities; within each window, inter-arrival times were sampled from a uniform distribution to generate arrivals at a constant rate. Weekday dynamics were realized by modulating these intraday profiles across days of the week, while global dynamics were imposed by varying the overall arrival intensity over time. As a result, inter-arrival times are non-stationary across multiple temporal dimensions, and the marginal distribution over the full log corresponds to a mixture induced by these dynamics.

and can be reliably identified from event logs, while global dynamics capture all remaining variations beyond weekly regularities. This includes longer-term seasonal effects (e.g., monthly or yearly patterns), gradual trends, and one-off external events. While such effects can therefore be modeled within our approach, we do not explicitly enforce additional temporal segmentation levels. Instead, we deliberately restrict explicit segmentation to dynamics that are both commonly observed and sufficiently supported by data, avoiding over-segmentation and unstable arrival estimates. The approach remains extensible, but focuses on a minimal set of dynamics that can be robustly integrated into business process simulation.

Limitations of Existing Approaches in BPS Given the above illustration, we postulate that an effective approach for deriving an arrival model from data should take these three dynamics into account, i.e., it needs to be able to consider potentially varying arrival patterns on the intraday, weekday, and global level.

Moreover, from this perspective, we argue that existing approaches used in BPS fail to do so, leading to poor arrival estimates as shown in the green-shaded area of Fig. 1. Static approaches ignore all three dynamics by assuming a single distribution over inter-arrival times. For instance, *Best Distribution* (Camargo et al. 2020), which fits a parameterized distribution to the inter-arrival times just fluctuates around the naive *Mean* baseline. *Prophet*, a dynamic approach, appears to capture weekday and global variations to some extent but neglects intraday patterns, as its common implementation in BPS approaches (Camargo et al. 2022; Meneghello et al. 2023) estimates arrivals per time unit and distributes them randomly, lacking a direct mechanism for precise timestamp estimation. In our loan process, *Prophet* greatly exaggerates the gradual trend, which we suspect to be a general problem due to a lack of autoregressive components (Taylor and Letham 2018). In contrast, the green line representing our AT-KDE approach (detailed in the next section) shows that more accurate arrivals can be generated by appropriately taking dynamics at different levels into account.

Impact on BPS Accuracy We compare arrival modeling approaches in the loan application process to demonstrate how poor arrival estimates—e.g., those that ignore temporal dynamics—can degrade simulation accuracy. We employ *AgentSimulator* (Kirchdorfer et al. 2024) to discover a BPS model and simulate process executions, evaluating results against a held-out test set using the current state-of-the-art BPS evaluation framework (Chapela-Campa et al. 2023). Focusing on time and congestion perspectives, we report distribution distances for cycle time (CTDD), absolute event time (AEDD), circadian event time (CEDD), and relative event time (REDD).

Table 1 suggests that (i) simulation quality varies greatly depending on the chosen arrival approach, and (ii) AT-KDE consistently achieves the most accurate results. For instance, AT-KDE yields a CTDD of 54.08, caused by an average simulated cycle time of 119 hours, compared to 143 hours in the ground truth, 29 hours with *Best Distribution*, and 640 hours with *Prophet*. These deviations reflect arrival estimation errors (cf. Fig. 1):

Table 1 Simulation accuracy for different arrival models (lower is better)

Arrival approach	CTDD	AEDD	CEDD	REDD
Mean	114.53	146.86	0.20	118.23
Best Distribution	112.67	145.47	0.24	115.97
Prophet	504.65	605.26	0.83	433.67
AT-KDE	**54.08**	**85.59**	**0.12**	**56.18**

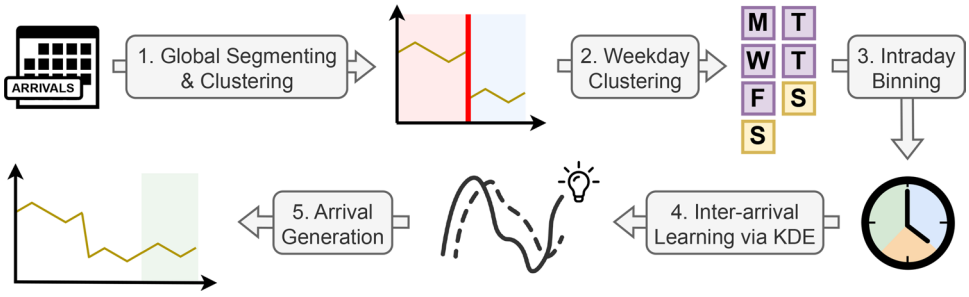


Fig. 3 Workflow of the modeling methodology of our AT-KDE approach

Best Distribution underestimates arrivals, shortening cycle times, while *Prophet* overestimates them, inflating delays. Now, consider a simulation used to assess whether hiring an additional employee would reduce application-to-decision times. If the arrival model significantly misrepresents the actual workload—as with *Best Distribution* or *Prophet*—the simulation's outcome may be misleading, potentially causing costly or ineffective staffing decisions. These results highlight the critical need for accurate arrival modeling to ensure simulation outcomes are realistic.

Approach

This section presents our AT-KDE approach for modeling and simulating arrival times in BPS. Its input, output, and main steps are as follows:

Input Our approach takes as input an *arrival dataset*, denoted by $\mathcal{D} = (\mathbf{t}^i)_{i=1}^N$. This dataset is a sequence of N sequences, where each $\mathbf{t}^i$ represents the sequence of arrival timestamps on a day $i \in \{1, \ldots, N\}$. Note that the length of $\mathbf{t}^i$ varies. Moreover, the simulation domain is provided. For example, a range of $\tilde{N}$ days *after*$\mathbf{t}^N$.

Output Similar to its input, our approach generates as output another sequence of sequences of arrival timestamps $\mathcal{D}\prime = (\tilde{\mathbf{t}}^i)_{i=1}^{\tilde{N}}$ over $\tilde{N}$ days, corresponding to the requested simulation domain.

Approach Steps The overarching goal is to simulate realistic arrival times by discovering a case-arrival model that explicitly captures the dynamic nature of organizational environments. To achieve this, our *divide-and-conquer* approach, AT-KDE, consists of five main steps, as visualized in Fig. 3. The first three steps make up the *divide* phase, where the arrival dataset is partitioned into subsets to address the different types of dynamics outlined in Sect. 3. In the *conquer* phase, steps 4 and 5 focus on learning a separate KDE model for each subset and combining these models to generate new arrivals.

1. *Global Segmenting and Clustering.* Given the arrival dataset $\mathcal{D}$, we account for *global dynamics* by identifying change points that signal potential drifts or seasonalities in the data. These change points divide the dataset into segments, capturing periods of consistent arrival patterns. To group similar patterns, the segments are clustered into a set of $J \geq 1$ segment clusters $\{D_1, \ldots, D_J\}$, where each cluster contains one or more segments from $\mathcal{D}$.
2. *Weekday Clustering.* Given a segment cluster D_j, we account for *weekday dynamics* by grouping weekdays with similar arrival patterns. Thus, we divide the global cluster D_j into weekday clusters $\{W_1^j, \ldots, W_{K_j}^j\}$. If no clusters are found we obtain the upper bound (number of days in a week): $K_j = 7$.

Algorithm 1 ClusterGlobalSegments

Input: Arrival dataset $\mathcal{D}$, window size ω, max clusters $k_{\max}$, sensitivity range Z
Output: Set of global segment clusters $\mathcal{G} = \{D_1, \ldots, D_J\}$, cluster labels $\mathcal{L}$

▷ **Data Transformation**
$M \leftarrow \text{GetArrivalCountSequence}(\mathcal{D})$
$M_A \leftarrow \text{ComputeMovingAverages}(M, \omega)$
$\Lambda \leftarrow \text{ComputeSlidingWindowDifferences}(M_A, \omega)$
▷ **Segmentation and Clustering**
for $z \in Z$ **do**
  $\mathcal{C} \leftarrow \text{DetectChangePoints}(\Lambda, z)$
  $\mathcal{S} \leftarrow \text{GetSegments}(\mathcal{C}, \mathcal{D})$
  $\mathcal{G}, \mathcal{L} \leftarrow \text{ClusterSegments}(\mathcal{S})$
  $len_s \leftarrow \text{GetSegmentLengthsInDays}(\mathcal{S})$
  ▷ **Solution Validation**
  if $\min(len_s) < \omega$ **or** $|\mathcal{S}| < 2$ **or** $|\text{unique}(\mathcal{L})| \geq k_{\max}$ **then**
    if $z == \text{FinalElementOf}(Z)$ **then**
      $\mathcal{G} \leftarrow \{\mathcal{D}\}; \mathcal{L} \leftarrow \langle 1 \rangle$
  else
    break
return $\mathcal{G}, \mathcal{L}$

3. *Intraday Binning.* Given a weekday cluster W_k^j, we account for *intraday dynamics* by dividing a day $\mathbf{t}^i \in W_k^j$ into $L \in \mathbb{N}$ equally long bins.
4. *Inter-arrival Learning via KDE.* The arrival dataset $\mathcal{D}$ is now divided into multiple disjoint subsets. Each of those contains sequences of arrival timestamps that exhibit similar characteristics. Next, for each subset, a separate KDE-model is fitted to the inter-arrival times, resulting in an ensemble of KDE-models that collectively capture the diverse dynamics of the data.
5. *Arrival Generation.* Finally, we iteratively sample from the KDE-model ensemble to generate new arrival timestamps for the simulation period, resulting in the output arrival dataset $\mathcal{D}\prime$.

Running example (uni library WiFi) To provide an intuition with respect to each step of our approach, we present the following running example: consider a university library that logs timestamps of devices that (re-)connect with its WiFi network. We treat each connection as an arrival event. Thus, for each day i, the sequence $\mathbf{t}^i$ contains the within-day timestamps at which devices connect, and $|\mathbf{t}^i|$ is the daily arrival count.

Step 1: global segmenting and clustering

The first step focuses on identifying global dynamics in the arrival dataset $\mathcal{D}$, such as seasonalities or concept drifts. As outlined in Algorithm 1, this is achieved by three key phases: *Data Transformation* transforms the arrival dataset into a higher-level representation, enabling the detection of meaningful change points and clustering of the resulting segments based on statistical similarity in the *Segmentation and Clustering* phase. Finally, the *Solution Validation* phase evaluates the admissibility of the identified segment clusters by checking if they meet a set of predefined requirements. Conceptually, we note that this step could also be performed via sufficiently interpretable time series approaches which are capable of extracting pattern-related characteristics (e.g., seasonal

ARIMA). However, our design is primarily motivated by the underlying nature of many event logs-one that often times does not exhibit clear patterns like seasonalities.

Data Transformation First, we aggregate the arrival dataset $\mathcal{D}$ into a sequence of daily arrival counts M, where each $M_i = |\mathbf{t}^i|$ represents the number of arrivals on day i, for $i = 1, \ldots, N$ (Line 2). This aggregation forms the basis for detecting global dynamics. Next, we compute the moving averages M_A based on M using a window size $\omega \in \mathbb{N}$ (Line 3). This smoothing reduces noise and highlights trends in the arrival counts. The moving average at position i is calculated as $M_{A,i} = \omega^{-1} \sum_{k=i}^{i+\omega-1} M_k$, $i = 1, \ldots, N - \omega + 1$. We then calculate the sequence of sliding window differences Λ based on M_A (Line 4), which measures the average change between two consecutive moving windows: $\Lambda_i = M_{A,i+\omega} - M_{A,i}$, for $i = 1, \ldots, N - 2\omega + 1$. This sequence captures the rate of change in M_A, where larger absolute values indicate more significant shifts in arrival patterns.

Segmentation and Clustering Given the sequence of sliding window differences Λ, we proceed with identifying change points and clustering the resulting segments. The segment clustering serves two main purposes: it mitigates variance inflation due to data scarcity by combining data from similar segments, and it reveals global patterns given by the ordering of segment clusters, which is essential for simulating new arrival data $\mathcal{D}\prime$ in later steps.

DetectChangePoints: We detect change points in the manner of outlier detection: We begin with some sensitivity parameter value $z \in Z \subseteq (0, 1]$, controlling the strictness of the outlier detection, and compute a change factor $\mathrm{CF} = 1.5 \times \mathrm{IQR} \times z$, where $\mathrm{IQR} = Q_3 - Q_1$ is the interquartile range given by the first (Q_1) and third (Q_3) quartiles of Λ. The factor of 1.5 is a standard rule-of-thumb from box plot construction that defines the whiskers outside of the IQR to flag potential outliers, balancing sensitivity to extremes with robustness. Then, indices where Λ_i falls outside the range $R = [Q_1 - \mathrm{CF}, Q_3 + \mathrm{CF}]$ are identified as candidate change points: $\tilde{\mathcal{C}} = \{i \mid \Lambda_i \notin R\}$. To assign only one change point to each drift, we constrain $\tilde{\mathcal{C}}$: For each set of consecutive outlier-indices $G \subseteq \tilde{\mathcal{C}}$, we select the index $i \in G$ with the maximum absolute value of Λ_i. The set of change points is then defined as $\mathcal{C} = \{i \in G \mid i = \arg\max_{j \in G} |\Lambda_j|\}$. Thus, note that AT-KDE does not remove or filter outliers; instead, they are interpreted as indicators of potential global changes in arrival intensity at an aggregated level. By collapsing consecutive outlier indices into a single representative change point, the procedure ensures that short bursts of extreme deviations give rise to at most one candidate change point, thereby preventing over-segmentation.

GetSegments: The identified change points $\mathcal{C}$ partition $\mathcal{D}$ into a collection of disjoint segments $\mathcal{S} = \{\tilde{D}_1, \tilde{D}_2, \ldots\}$ (Line 8). Each $\tilde{D}_s$ corresponds to a period between two change points and reflects a consistent arrival pattern. For instance, in our motivating loan application process in Fig. 1, three change points are detected during the training period: *Start inflation, Start gradual decrease of inflation*, and *End inflation*, resulting in four adjacent segments.

ClusterSegments: We next group segments with similar arrival patterns through clustering (Line 9). To do this, we characterize each segment via six statistical features: the average number of arrivals per day, the 25th and 75th percentiles of daily arrivals, the standard deviation of inter-arrival times, and the 25th and 75th percentiles of inter-arrival times. After standardizing these features (removing the mean and scaling to unit

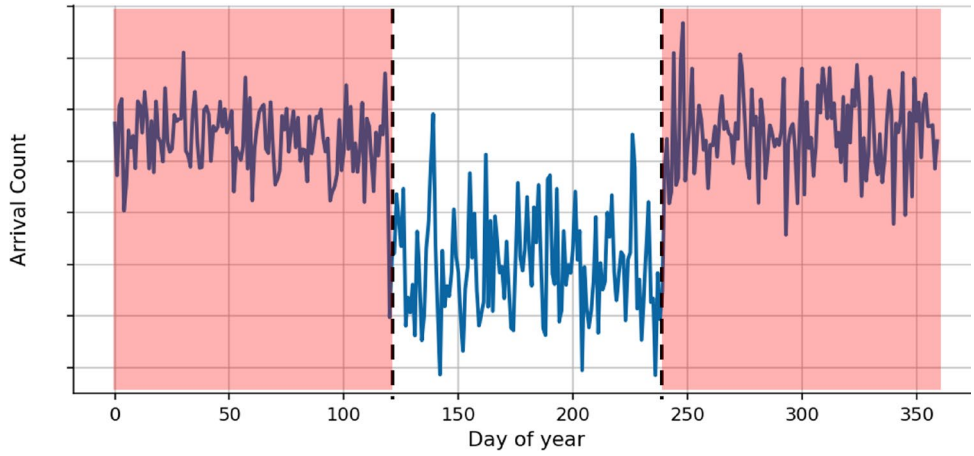


Fig. 4 University library illustration: schematic clustering of two global segments via AT-KDE; term time and summer break

variance), and to avoid assuming the number of clusters apriori, we apply DBSCAN (Ester et al. 1996) to label segments. Each unique label in $\mathcal{L}$ corresponds to a cluster $D_j \in \mathcal{G},\ j \in \{1, \ldots, J\},\ J \leq |\mathcal{S}|$, denoting the disjoint union of all segments whose arrival sequences exhibit similar patterns.

Solution Validation Finally, we assess the admissibility of the segmentation and clustering results (Lines 10–17). Each segment must span at least ω days to capture meaningful global dynamics—shorter patterns are addressed in subsequent steps—ensuring segments are of sufficient length. Additionally, there must be at least two segments ($|\mathcal{S}| \geq 2$) to identify data changes, and fewer than $k_{\max}$ clusters to ensure each provides enough data for robust model fitting following subsequent steps. If these conditions are not met for the current sensitivity parameter z, we proceed to the next value in Z. If no value in Z yields an admissible solution, we conclude that meaningful global change points are absent and set $\mathcal{G} = \{\mathcal{D}\}$ with a single cluster label $\mathcal{L} = \langle 1 \rangle$ (Line 14).

As a result, this step divides the entire dataset into statistically more coherent and disjoint subsets: $\mathcal{D} = \biguplus_{j=1}^{J} D_j$.

In the library setting, a natural global regime change is the transition between term time and summer break. During term time, daily connection counts are consistently higher; during summer, they drop to a lower level. Step 1 detects such level shifts in the aggregated count series and partitions the year into corresponding segments; across multiple years, segments with similar intensity are then clustered (e.g., all "term" segments vs. all "summer" segments), as illustrated in Figure 4.

Step 2: Weekday clustering

Step 2 addresses *weekday dynamics* within each global segment cluster identified in the first step. Specifically, for each global cluster D_j, we group weekdays with similar arrival patterns into weekday clusters $\{W_1^j, \ldots, W_{K_j}^j\}$, where $K_j \leq 7$, $j = 1, \ldots, J$. This allows us to model day-of-week variations effectively. Algorithm 2 outlines this process, which is divided into two phases:

Weekday Data Extraction For each weekday $w \in \{1, 2, \ldots, 7\}$, we extract all arrival times $\mathbf{t}^i$ that occur on that weekday within the global cluster D_j (Line 4): $\mathcal{T}_w^j = \{\mathbf{t}^i \mid \mathbf{t}^i \in D_j,\ \text{weekday}(\mathbf{t}^i) = w\}$.

Algorithm 2 ClusterWeekdays

Input: Global clusters $\mathcal{G} = \{D_1, \ldots, D_J\}$
Output: Set of weekday clusters $\mathcal{W}$

1: **for** each global cluster D_j **do**
2: ▷ **Weekday Data Extraction**
3: **for** each weekday $w \in \{1, 2, \ldots, 7\}$ **do**
4: $\mathcal{T}_w^j \leftarrow \{\mathbf{t}^i \in D_j \mid \text{weekday}(\mathbf{t}^i) = w\}$
5: ▷ **Feature Computation and Clustering**
6: **if** $\mathcal{T}_w^j \neq \emptyset$ **then**
7: $\mathbf{f}_w^j \leftarrow \text{ComputeStatistics}(\mathcal{T}_w^j)$
8: $\mathbf{F}^j \leftarrow \text{StandardizeFeatures}(\{\mathbf{f}_w^j \mid \mathcal{T}_w^j \neq \emptyset\})$
9: $\text{WTC}_j \leftarrow \text{WardClustering}(\mathbf{F}^j)$ ▷ Mapping from weekdays to cluster labels
10: **for** each cluster label k in WTC_j **do**
11: $W_k^j \leftarrow \{\mathbf{t}^i \in D_j \mid \text{WTC}_j(\text{weekday}(\mathbf{t}^i)) = k\}$
12: **for** each weekday $w \in \{1, 2, \ldots, 7\}$ **do**
13: **if** $\mathcal{T}_w^j = \emptyset$ **then**
14: Assign w to the special cluster W_{NoData}^j
15: **return** $\mathcal{W} = \bigcup_{j=1}^{J} \{W_1^j, \ldots, W_{K_j}^j\}$

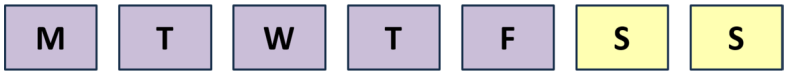


Fig. 5 University library illustration: schematic clustering of weekdays within a global cluster; e.g., term time

Feature Computation and Clustering For each weekday w with arrivals (i.e., $\mathcal{T}_w^j \neq \emptyset$), we compute a feature vector $\mathbf{f}_w^j$ characterizing its arrival patterns, such as the average number of arrivals per day (Lines 7–8). We then cluster[2] the feature matrix $\mathbf{F}^j$ to group similar weekdays, resulting in a mapping WTC_j from weekdays to cluster labels (Line 9). Subsequently, the arrival times of the current global segment cluster $\mathbf{t}^i \in D_j$ are assigned to weekday clusters W_k^j based on their weekdays' cluster labels (Line 11): $W_k^j = \{\mathbf{t}^i \in D_j \mid \text{WTC}_j(\text{weekday}(\mathbf{t}^i)) = k\}$. Weekdays without arrivals (i.e., $\mathcal{T}_w^j = \emptyset$) are grouped into a separate cluster W_{NoData}^j (Line 14). The total number of weekday clusters K_j equals the number of unique cluster labels in WTC_j, incremented by one if empty weekdays are present. Finally, the set of weekday clusters over all global clusters $\mathcal{W} = \bigcup_{j=1}^{J} \{W_1^j, \ldots, W_{K_j}^j\}$ represents the output of Step 2, with each global cluster D_j divided into disjoint subsets such that $D_j = \biguplus_{k=1}^{K_j} W_k^j$, with $W_k^j \in \mathcal{W}$. Since each subset W_k^j is created by performing clustering within its global cluster D_j, the elements within a subset are more similar to each other than to those in any other subset, reflecting improved statistical coherence.

In our university library example, Step 2 is applied within each global cluster. For the term time cluster, it is plausible that Mon–Fri exhibit similar attendance, while Sat/Sun follow a different pattern due to opening hours and leisure behavior. Step 2 groups such weekdays into distinct weekday clusters (e.g., {Mon–Fri} vs. {Sat,Sun}), as shown in Figure 5.

[2] We use Ward's method (Ward Jr. 1963), but other clustering algorithms are also applicable.

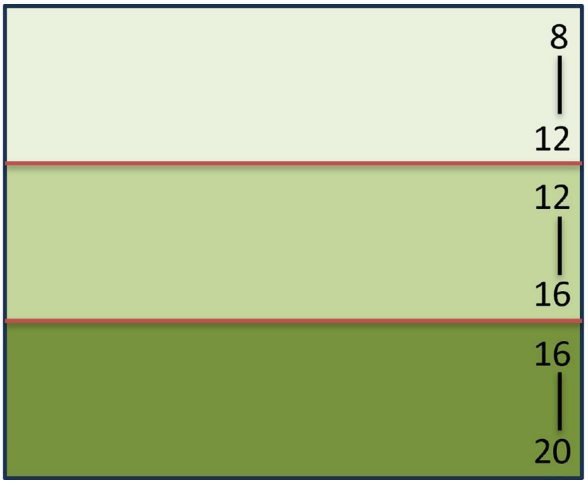


Fig. 6 University library illustration: schematic clustering of a day within a (global, weekday) cluster; e.g., a weekday within term time (start/end: 8am–8pm)

Step 3: intraday binning

This step addresses *intraday dynamics* within each weekday cluster $W_k^j \in \mathcal{W}$ identified in Sect. 4.2. To capture variations in arrival patterns throughout the day, we divide each day $\mathrm{t}^i \in W_k^j$ into $L \in \mathbb{N}$ equally long time bins. Specifically, we partition the time domain of a day into L consecutive intervals, where each bin spans an equal duration (e.g., 3 hours). For each day $\mathrm{t}^i_{(j,k)}$ in W_k^j, we segregate the arrival times into these bins, resulting in subsets $\{\mathbf{t}^i_{(j,k,1)}, \ldots, \mathbf{t}^i_{(j,k,L)}\}$, where: $\mathbf{t}^i_{(j,k,l)}$ represents all timestamps that occurred during the l-th bin on day $\mathrm{t}^i_{(j,k)}$. Also, we obtain that each weekday cluster W_k^j can now be represented via a disjoint union of the aforementioned bins: $W_k^j = \biguplus_{l=1}^{L} \mathbf{t}^i_{(j,k,l)}$.

In our university library example, within a given (global, weekday) cluster, Step 3 discretizes the day into L equal-duration bins to capture systematic within-day variation. For instance with $L = 3$, a term time weekday may show a morning ramp-up, a midday plateau, and an afternoon/evening decline toward closing time. Binning separates these regimes so that each bin can be modeled with its own inter-arrival distribution (cf. Figure 6).

Step 4: inter-arrival learning via KDE

Having completed the *divide* phase of our approach, we are able to fully partition the arrival dataset: $\mathcal{D} = \biguplus_{j=1}^{J} \biguplus_{k=1}^{K_j} \biguplus_{l=1}^{L} \mathbf{t}_{(j,k,l)}$. Concretely, each sequence of arrival times $\mathbf{t}_{(j,k,l)}$ corresponds to a global cluster D_j, a weekday cluster W_k^j, and an intraday bin l, and will be associated to a separate arrival time model. By dividing the dataset along these temporal dimensions, we expect to mitigate much of the underlying dynamic variability, thereby enabling an effective application of static modeling methods.

However, even after isolating arrivals into temporally homogeneous subsets, empirical inter-arrival time distributions may remain skewed, heavy-tailed, or multi-modal, and thus are not well captured by a single parametric family. Committing to a fixed distributional form would therefore introduce an additional modeling assumption that is difficult to validate across datasets and segments. Therefore, we adopt Kernel Density Estimation (KDE) as a flexible, non-parametric alternative that allows each subset's

Algorithm 3 GenerateArrivals

Input: Num. of days $\tilde{N}$, Labels $\mathcal{L}$, Segments $\mathcal{S}$, Dataset $\mathcal{D}$, Bins L, KDE ensemble $\mathcal{E}$
Output: Generated arrival dataset $\mathcal{D}'$

1: ▷ **Time Frame Initialization**
2: *estim_segments_per_day* ← EstimateSegmentCluster($\tilde{N}, \mathcal{L}, \mathcal{S}$)
3: *lower_time*, *upper_time* ← DetermineBounds($\mathcal{D}$)
4: *time_bins* ← CreateTimeBins(*lower_time*, *upper_time*, L)
5: ▷ **Arrival Data Generation**
6: $\mathcal{D}' \leftarrow \langle\rangle$ ▷ Initialize sequence of arrival sequences
7: **for** $i \in \{1, \ldots, \tilde{N}\}$ **do**
8: *current_date* ← GetDate(i)
9: *weekday_cluster* ← GetWeekdayCluster(*current_date*)
10: *estim_segment* ← *estim_segments_per_day*[i]
11: *seq* ← $\langle\rangle$ ▷ Initialize arrivals for one day
12: **for** each bin in *time_bins* **do**
13: *interarrivals* ← SampleInterarrivals($\mathcal{E}$, *bin*, *weekday_cluster*, *estim_segment*)
14: *arrivals* ← GenerateArrivals(*interarrivals*, *bin.start_time*, *bin.end_time*)
15: *seq*.append(*arrivals*)
16: $\mathcal{D}'$.append(*seq*)
17: **return** $\mathcal{D}'$

empirical distribution to be represented directly, without imposing restrictive structural assumptions. Briefly, the KDE for a probability density function f, given i.i.d. realizations $(X_i)_{i=1}^n$, is defined as $\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x-X_i}{h}\right)$. Here, h is termed *bandwidth* and $K(\cdot)$ the *kernel function*. The quality of the estimated density $\hat{f}$ depends on both K and h (Parzen 1962).

Modeling each subset's inter-arrival times with a separate KDE forms an *ensemble of models*$\mathcal{E}$, which we use in the final step to generate new arrivals.

In our university library example, after Steps 1–3, the data is partitioned by global regime, weekday cluster, and intraday bin; concretely, two global regimes (term time vs. summer break), two weekday clusters (weekday vs. weekend), and $L = 3$ bins. This yields $2 \times 2 \times 3 = 12$ subsets. We fit one KDE per subset, allowing each context (e.g., "summer–weekend–morning" vs. "term–weekday–morning") to retain its own empirical inter-arrival profile.

Step 5: arrival generation

The final step of our AT-KDE approach generates new arrival data for the simulation period. Algorithm 3 outlines this step, which consists of two main phases:

Time Frame Initialization For the chosen simulation horizon, each to-be-simulated day $i \in \{1, \ldots, \tilde{N}\}$ must be assigned a global segment cluster label (Line 2). If said day lies within the training domain $\mathcal{D}$, its assignment is immediate: since the global segments were learned from $\mathcal{D}$, we simply look up the corresponding cluster label for each day. If the simulation period extends beyond $\mathcal{D}$ (e.g., after the end of training), cluster labels are no longer observed and must be inferred. We do so using the following two rules: 1) If the sequence of observed global segment cluster labels $\mathcal{L}$ indicates a recurring pattern, we replicate it by determining the order and lengths of the segments and dividing the simulation period accordingly. For instance, if we have four global segments with cluster labels $\mathcal{L} = \langle 1, 2, 1, 2\rangle$, we can observe the recurring pattern that cluster 1 is followed by 2, which we then replicate accordingly. 2) If no recurring pattern is found (as

in our motivation in Fig. 1), we assume that the *most recent* global segment cluster best represents the future.

Next, we identify the earliest and latest times of the day when an arrival was observed in the training dataset $\mathcal{D}$ (Line 3) to approximate the *working hours* of the process. Subsequently, this range is divided into L equally long time bins (Line 4), as also done in Sect. 4.3.

By default, our approach initiates the simulation of new arrivals at the start time of the test set and generates cases over the same time span. However, it also supports[3] user-defined configurations, allowing for a custom simulation start and end time.

Arrival Data Generation We generate arrivals for each day $i \in \{1, \ldots, \tilde{N}\}$ in the simulation period (Line 7) by first determining the following parameters for our generation-method: the date corresponding to day i, beginning with the last date of the training data (Line 8) per default, the corresponding weekday cluster (Line 9), and the estimated global segment cluster (Line 10).

Given these parameters for day i, we iterate through each time bin of that day (Line 12) and sample inter-arrivals from the corresponding KDE in $\mathcal{E}$, which are cumulatively summed to form the actual timestamps until the end of the bin is reached. Note that weekday clusters W^j_{NoData} based on absent arrivals yield no KDE model. Respective days are instead reflected via no arrivals. The final output is the sequence of generated arrival timestamps $\mathcal{D}\prime$.

For our university library example, suppose we simulate arrivals for Jan. 1–10 using prior years as training data. Step 5 first assigns each simulated day a global cluster label using the learned segment pattern (here: term time). The weekday cluster is then determined from the calendar (weekday vs. weekend). For each day, we iterate over the L intraday bins and sample inter-arrival times from the KDE corresponding to the day's (global, weekday, bin) combination, cumulatively summing within each bin until its end time is reached. Concatenating the bin-wise samples yields the simulated timestamp sequence $\tilde{\mathbf{t}}^i$ for that day, repeated across the simulation horizon.

Experiments and results

This section outlines the experiments used to evaluate the performance of our AT-KDE approach. The evaluation consists of two parts. In Sect. 5.1, we examine how effectively AT-KDE learns arrival patterns from historical data and generates future arrival times, quantifying the similarity between the generated arrivals and the corresponding ground truth. We benchmark our approach against seven baseline approaches across 20 datasets. In Sect. 5.2, we evaluate how AT-KDE influences overall simulation quality. To do so, we first present a novel evaluation framework and subsequently apply it as the basis for assessing simulation outcomes. Implementations and additional results can be found in our repository.[4]

Evaluation 1: arrival time quality

In the first part of our evaluation, we begin by laying out the experimental setup in section "Experimental Setup". Subsequently, in section "Overall Results", we present the overall results of our experiments and compare the performance of AT-KDE to several

[3] For more details, the interested reader is referred to our repository's guidelines.

[4] https://github.com/konradoezdemir/AT-KDE

Table 2 Arrival time-related characteristics of the event logs

Event log	Traces	Time range (d.)	Arrivals per day		
			Min	Mean	Max
BPIC12	13087	151	17	86.1	233
BPIC12CW	9658	152	0	63.1	155
BPIC12O	5015	165	0	30.21	78
BPIC12W	9658	151	13	63.5	154
BPIC13C	1487	2332	0	0.6	15
BPIC17W	30276	369	0	81.8	288
BPIC19	251734	25923	0	9.7	1538
BPIC20D	10500	719	0	14.6	73
BPIC20I	6449	800	0	8.1	56
BPIC20Permit	7065	816	0	8.7	43
Env.permit	1434	478	0	3.0	28
Helpdesk	4580	1415	0	3.2	22
Hospital	99999	1130	3	88.4	132
Sepsis	1049	476	0	2.2	11
P2P	608	285	0	2.2	12
CVS	10000	63	0	156.25	253
Confidential 1000	1000	164	0	6.1	18
Confidential 2000	2000	358	0	5.6	16
ACR	954	148	0	6.4	26
Production	225	88	0	2.5	11

baseline approaches from the BPS domain and beyond. By comparing AT-KDE to other models across a diverse set of event logs, we are able to make claims with respect to flexibility, robustness and performance. Afterwards, we conduct a thorough statistical evaluation of the results in section "Statistical Evaluation". Finally, via a selection of post-hoc analyses and ablations in section "Post-hoc Analysis" and section "Ablation Studies", we connect specific model scores to potential root causes.

Experimental setup

Evaluation Data Our evaluation is based on a diverse set of 20 event logs[5] spanning domains such as financial services, public administration, and healthcare. These logs vary widely in key characteristics (cf. Table 2), including the number of arrivals, time spans, and arrival rates. For example, some processes exhibit pronounced global dynamics, such as abrupt increases in daily arrivals, while others remain relatively stable. Additionally, the logs feature a range of weekday and intraday patterns, from standard business hours (Monday to Friday, 9am–5pm) to continuous 24/7 arrivals, offering a comprehensive testing ground for our approach and benchmarks. For each event log, we derive the arrival dataset by selecting the earliest recorded timestamp for each case.

Benchmark Approaches We compare our approach against seven others, selected to ensure broad coverage of techniques from various domains:

- *Mean*: A simple baseline using the mean inter-arrival time to model arrivals.
- *Best Distribution* (Camargo et al. 2020): A static approach that selects the best fitting parametric distribution out of a set to sample new inter-arrival timestamps which are cumulatively summed, forming the prospective arrival timestamps.

[5] Datasets available at https://data.4tu.nl or https://zenodo.org/records/5734443

- *Prophet* (Taylor and Letham 2018): A dynamic time series forecasting method repurposed to estimate the number of hourly arrivals $\hat{n}$. The concrete timestamps are obtained via uniformly drawing $\hat{n}$ iid. timestamps within each hourly bin.
- *LSTM* (Hochreiter and Schmidhuber 1997): A recurrent neural network with a dedicated remember/forget cell-mechanism, which is trained to predict the next inter-arrival time based on a prefix of arrivals and other temporal features.
- *Chronos* (Ansari et al. 2024): A framework of transformer-based language model architectures repurposed for probabilistic time series forecasting.
- *XGBoost* (Chen and Guestrin 2016): A tree boosting model that predicts inter-arrival times using temporal and contextual features in a regression framework.
- *NPP* (Daw and Pender 2018): A non-homogenous point process model with self-exciting intesity function (Hawkes Process). It is fit on inter-arrival times and its samples are cumulatively summed, forming the prospective arrival timestamps.

Note that we augment *Mean, Best Distribution*, and *NPP* with a probabilistic component to consider non-working days and first and last arrival timestamps within a day, aligning with their usage in simulation models (Camargo et al. 2020; Kirchdorfer et al. 2024). Initially, we considered ARIMA-based forecasting via, for example, *auto-ARIMA*[6]. However, automatic parameter selection proved unstable across datasets, frequently yielding configurations that critically undermined the model's performance. Reliable ARIMA forecasts required manual intervention, contradicting our goal of a generalized approach. Therefore, we excluded ARIMA from our analysis.

Data Split We apply a temporal hold-out split with the first 80% of cases being in the train set and the last 20% of cases in the test set. The simulation domain is set to the test domain, i.e., we generate arrivals that match the time period of the test set.

Metrics To evaluate and compare the different arrival time prediction approaches, we use the Case Arrival Distribution Distance (CADD) metric proposed by Chapela-Campa et al. (2023). This metric aggregates the test and simulated arrival timestamps into (separate) hourly counts and computes the Earth Mover's Distance (EMD) between the resulting empirical distributions. The CADD metric is part of a larger evaluation framework, which serves as the current standard in assessing the quality of simulation outcomes.

Hyperparameters AT-KDE requires optimization of its bandwidth, which we initially estimate via Silverman's rule of thumb (Silverman 1986). Given the strong underlying assumption of said rule (normality of data), we tune the ground bandwidth via multiplication with a factor k, where $k \in (0, 200]$, selecting the product-bandwidth that performs best (CADD-wise) on the training set. Moreover, we opt for the Gaussian Kernel K (Parzen 1962). For clustering and segmentation, we adopt a sensitivity range $Z = [0.1, 1]$, a temporal window $\omega = 7$ days, a maximum $k_{\max} = 6$ weekday clusters, and a bin size $L = 3$. This configuration proved robust across diverse datasets, though practitioners may change these settings (e.g., to accommodate different arrival sparsities). For both *LSTM* and *XGBoost*, grid-search-based hyperparameter tuning is performed. For *Prophet*, we follow the tuning protocol of Camargo et al. (2022).

[6] https://alkaline-ml.com/pmdarima

Table 3 Average results of benchmarks measured by the square root of CADD. Missing entries based on convergence failure due to data size

Event log	Mean	Best Dist.	Prophet	LSTM	Chronos	XGBoost	NPP	AT-KDE
BPIC12	4.83	5.12	4.46	4.18	5.48	4.26	5.08	**3.71**
BPIC12CW	6.61	6.84	4.47	6.27	8.33	8.69	6.79	**3.21**
BPIC12O	14.75	14.55	13.61	15.00	11.72	12.73	14.59	**11.56**
BPIC12W	4.78	4.76	5.07	4.54	4.87	**3.66**	4.48	4.80
BPIC13C	66.22	66.65	13.99	27.93	14.14	13.39	68.70	**13.25**
BPIC17W	15.63	15.47	**9.65**	13.74	13.29	11.96	15.57	10.24
BPIC19	300.66	295.53	17.83	/	22.54	21.38	/	**17.58**
BPIC20D	10.33	11.55	**5.95**	6.94	8.58	6.90	11.31	6.84
BPIC20I	28.02	29.09	17.41	**15.01**	19.93	17.37	/	16.59
BPIC20P	25.06	24.47	15.08	**10.91**	13.58	12.58	25.14	13.82
Env.permit	33.48	31.99	17.97	19.62	18.19	14.98	31.76	**13.74**
HelpDesk	55.69	56.15	23.28	19.58	44.39	36.79	55.61	**19.48**
Hospital	22.59	22.45	**15.89**	22.91	31.10	22.25	22.56	22.75
Sepsis	20.97	21.55	15.60	18.33	18.19	18.52	20.86	**15.56**
P2P	25.28	25.91	22.65	26.41	24.42	20.43	13.05	**12.87**
CVS	8.97	9.02	6.97	8.01	5.78	6.99	9.09	**3.54**
Conf. 1000	12.54	12.25	6.26	7.16	6.05	9.68	12.16	**5.20**
Conf. 2000	18.23	17.97	9.86	11.33	10.15	8.99	18.30	**6.49**
ACR	9.48	9.63	6.80	**6.49**	8.63	7.34	8.78	7.00
Production	9.06	8.47	10.66	7.06	6.38	**4.15**	8.97	6.42

Overall results

Table 3 summarizes the experimental results for the 20 event logs averaged over 10 runs. Our AT-KDE approach achieves the best performance in 12 logs—clearly outperforming the benchmarks—followed by *Prophet*, *LSTM*, and *XGBoost* leading in the remaining 3/3/2 logs, respectively. The static approaches *Mean*, *Best Distribution*, and *NPP*, likely due to the rigidness of their parametric assumptions, lack the flexibility required to adapt effectively to temporal dynamics, failing to claim a single dataset. Notably, the recently proposed time series foundation framework *Chronos* does not achieve a best performance on any of the datasets. In all logs where AT-KDE is not the top performer, the gap to the leading approach is rather small. The only exception is *Hospital*, where the gap to *Prophet* is substantial. This can likely be attributed to a sudden decrease in arrival counts during the log's test period. We believe that *Prophet*'s focus on uncovering trends allowed it to anticipate this drop, whilst, for AT-KDE, the evidence hinting at this phenomenon appears too scarce.

Statistical evaluation

In addition to reporting mean performance scores for AT-KDE and all baselines, we assess whether the observed differences are statistically significant. All computations can be found in our repository.

Overall Equality of Model Performance Using a significance level of $\alpha = 1\%$, we first test the hypothesis that all models achieve the same performance across event logs (cf. Table 3):

$$H_0 : \text{all models have equal performance} \quad \text{vs.} \quad H_1 : \text{at least one model differs.}$$

Table 4 Pairwise Wilcoxon signed-rank tests of AT-KDE against baseline models (Holm correction over 7 tests)

Model	Logs (of 20)	p-value	p_{Holm}	Sig. at $\alpha = 1\%$
Mean	20	4.77×10^{-6}	3.34×10^{-5}	Yes
Best Distribution	20	4.77×10^{-6}	3.34×10^{-5}	Yes
Prophet	20	3.65×10^{-3}	8.37×10^{-3}	Yes
LSTM	19	3.09×10^{-3}	8.37×10^{-3}	Yes
Chronos	20	9.54×10^{-6}	4.77×10^{-5}	Yes
XGB	20	2.79×10^{-3}	8.37×10^{-3}	Yes
NPP	18	3.81×10^{-5}	1.53×10^{-4}	Yes

Since not every model produces a score for every log, we employ the Skillings–Mack test (Skillings and Mack 1981), a generalization of Friedman's rank test to incomplete data.[7] The test yields $p \approx 2.85 \times 10^{-11}$, which provides strong evidence against H_0 at the level $\alpha = 1\%$. Hence, we have sufficient ground to assume that at least one model differs in performance, yielding us the possibility to proceed with a detailed comparative analysis.

Pairwise Model Comparisons We begin by reiterating that the descriptive results in Table 3 exhibit a consistent pattern: AT-KDE attains lower root-CADD than every baseline on the majority of event logs. On this basis, we deem that it is reasonable to formulate a directional hypothesis, i.e., "AT-KDE improves over all baselines". To guard against overly optimistic inference, we (i) test at a level of $\alpha = 1\%$ and (ii) apply Holm–Bonferroni (Holm 1979), controlling for multiplicity-induced type-1 errors.

Formally, for every baseline $b_j \in \{\text{Mean}, \ldots, \text{NPP}\}$, $j = 1, \ldots, 7$, we test

$$H_0^{(j)} : \sqrt{\text{CADD}_{\text{AT-KDE}}} = \sqrt{\text{CADD}_{b_j}} \quad \text{vs.} \quad H_1^{(j)} : \sqrt{\text{CADD}_{\text{AT-KDE}}} < \sqrt{\text{CADD}_{b_j}},$$

using a one-sided Wilcoxon signed-rank test (Rey and Neuhäuser 2011). Our global claim "AT-KDE outperforms all baselines" shall correspond to the alternative $H_1 = \bigcap_{j=1}^{7} H_1^{(j)}$, with corresponding null $H_0 = \bigcup_{j=1}^{7} H_0^{(j)}$ ("at least one baseline is not outperformed by AT-KDE"). Since seven hypotheses are tested simultaneously, we control the family-wise error rate at $\alpha = 1\%$ via the Holm–Bonferroni procedure. The raw and Holm-adjusted p-values are reported in Table 4.

Clearly, Holm-adjusted p-values fall below 1%, so every $H_0^{(j)}$ is rejected and, hence, the union null H_0 is rejected at $\alpha = 1\%$. As a result, we report that there is sufficient evidence in our study that AT-KDE significantly outperforms all benchmark models.

Post-hoc analysis

Outline of Weekday and Intraday Simulation Quality To gain a better understanding of how the simulated arrival timestamps differ across approaches already used in BPS, we compare the hour-day distribution of arrivals in the BPIC12 log for AT-KDE, *Prophet*, and *Best Distribution* against the test data in Fig. 7. The process shows most arrivals between Monday and Wednesday, with only few at night. While *Best Distribution* fails to capture this pattern, both AT-KDE and *Prophet* are able to distinguish between day and night patterns. However, AT-KDE more accurately concentrates arrivals on the first three weekdays, whereas *Prophet* overestimates arrivals from Thursday to Sunday.

[7] https://cran.r-project.org/package=Skillings.Mack

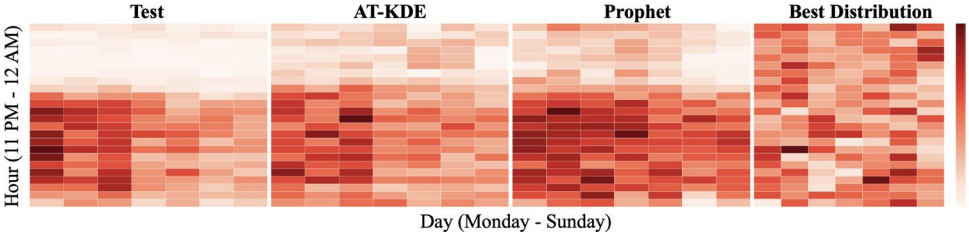
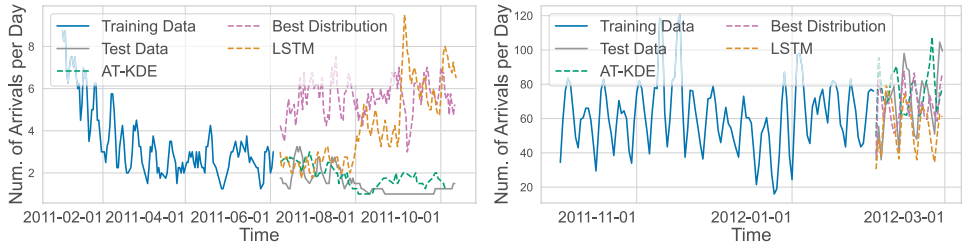


Fig. 7 Distribution of arrivals of BPIC12 per hour of each day of the week



(a) P2P (b) BPIC12W

Fig. 8 Comparison of arrivals between AT-KDE, *LSTM*, and *Best Distribution*. (**a**) P2P. (**b**) BPIC12W

Table 5 Mean execution times in seconds across all 20 logs

	Mean	BD	Prophet	LSTM	Chronos	XGB	NPP	AT-KDE
Training	0.02	1.76	745.19	25.45	102.37	77.67	270.09	13.16
Simulation	0.03	0.13	174.73	337.55	0.58	26.66	0.31	0.03

How Model Performance Relates to Data To examine in which scenarios AT-KDE outperforms existing approaches and when differences are negligible, we showcase two event logs in detail. For P2P, AT-KDE significantly outperforms the benchmarks. As shown in Fig. 8a, P2P is subject to a strong decrease in daily arrival counts during the first half of the training period. The static approach *Best Distribution* ignores this drift and takes all training data into account, leading to a substantial overestimate of daily arrivals compared to the actual test data. Notably, even a dynamic approach like the *LSTM* fails to produce realistic arrival timestamps, as it appears biased by the high arrival volumes at the start of the training period. In contrast, AT-KDE correctly identifies this drop and explicitly considers it by leveraging only post-drift data for the simulation, yielding significantly more accurate arrival times. Conversely, BPIC12W (Fig. 8b) shows no drift behavior on the global scale with arrival counts exhibiting a rather constant mean and variance throughout time. Therefore, a static approach such as *Best Distribution* can achieve comparable performance as *LSTM* or AT-KDE in this scenario.

Execution Time All experiments were run on an Apple M3 Pro (12-core CPU, 18GB RAM). As suggested by Table 5, *Mean* and *Best Distribution* achieve near-instantaneous times. Although AT-KDE's training takes slightly longer (13s), its simulation is competitively fast (0.03s). In contrast, all other benchmarks are notably slower in both phases, highlighting AT-KDE's overall efficiency.[8]

[8] The full overview including standard deviations can be found in our repository.

Table 6 Average results (with std.) over 10 runs of AT-KDE for different intraday bin sizes $L \in \{1, 3, 5, 7\}$, measured by the square root of CADD. Mean rank (lower is better) is computed per event log across L (ties share rank), then averaged across logs

Event log	$L = 1$	$L = 3$	$L = 5$	$L = 7$
BPIC12	4.11 (0.29)	3.71 (0.31)	4.08 (0.24)	3.91 (0.57)
BPIC12CW	3.50 (0.06)	3.21 (0.39)	3.36 (0.52)	3.20 (0.62)
BPIC12O	10.94 (0.24)	11.56 (0.25)	11.01 (0.29)	10.95 (0.31)
BPIC12W	4.49 (0.38)	4.80 (0.37)	4.51 (0.97)	4.49 (0.49)
BPIC13C	12.58 (1.08)	13.25 (0.60)	13.01 (0.88)	12.72 (0.49)
BPIC17W	9.72 (0.26)	10.24 (0.33)	9.65 (0.20)	9.54 (0.29)
BPIC19	17.16 (0.11)	17.58 (0.16)	17.36 (0.20)	17.07 (0.31)
BPIC20D	6.97 (0.55)	6.84 (0.33)	7.00 (0.42)	6.54 (0.29)
BPIC20I	16.48 (0.59)	16.59 (0.23)	16.21 (0.33)	16.06 (0.37)
BPIC20P	14.24 (0.47)	13.82 (0.40)	13.92 (0.28)	14.07 (0.27)
Env.permit	14.46 (1.35)	13.74 (0.97)	13.78 (0.56)	14.61 (0.35)
HelpDesk	19.94 (0.66)	19.48 (0.71)	19.51 (0.76)	19.68 (0.58)
Hospital	22.83 (0.11)	22.75 (0.23)	22.66 (0.46)	22.74 (0.34)
Sepsis	17.72 (2.58)	15.56 (2.58)	18.09 (3.00)	16.51 (2.10)
P2P	13.57 (1.65)	12.87 (2.74)	12.83 (2.33)	12.55 (2.45)
CVS	3.49 (0.19)	3.54 (0.30)	3.49 (0.18)	3.60 (0.27)
Conf. 1000	6.03 (0.58)	5.20 (0.51)	5.05 (0.81)	4.41 (0.35)
Conf. 2000	6.52 (0.73)	6.49 (0.64)	6.32 (0.57)	6.51 (0.55)
ACR	7.80 (0.78)	7.00 (0.77)	6.91 (0.80)	7.21 (1.22)
Production	9.04 (1.07)	6.42 (1.21)	5.48 (1.02)	6.26 (0.26)
Mean rank	3.10	2.60	2.28	2.03

Ablation studies

Sensitivity Analysis for Intraday-Bin Hyperparameter In the main benchmark, we set the number of intraday bins to $L = 3$. We assess sensitivity to this choice by evaluating $L \in \{1, 3, 5, 7\}$ on all 20 event logs. For each $(\log, L)-$combination, we report the mean and standard deviation of $\sqrt{\text{CADD}}$ over 10 repetitions. To summarize performance across logs, we compute within-log ranks (1 = best/lowest $\sqrt{\text{CADD}}$; ties receive the average rank) and report the mean rank over logs. The results are shown in Table 6.

Across datasets, AT-KDE is not highly sensitive to the exact value of L: differences are typically modest and frequently within one standard deviation. Nevertheless, the mean-rank summary indicates a consistent preference for finer intraday discretization (mean rank: 3.10 for $L = 1$, 2.60 for $L = 3$, 2.28 for $L = 5$, and 2.03 for $L = 7$). This pattern suggests that, when intraday structure exists, additional bins can help capture within-day heterogeneity.

The effect is particularly visible in logs with pronounced within-day dynamics. For example, *Confidential 1000* improves steadily with larger L, and *Production* decreases from 9.04 ($L = 1$) to 5.48 ($L = 5$), indicating that intraday binning can materially improve arrival-time modeling in such cases.

Despite this, we retain $L = 3$ as a default because it offers a favorable robustness–complexity trade-off. Increasing L reduces the amount of data per bin and may introduce sparsity in smaller logs or low-intensity regimes, which can destabilize KDE estimates and increase runtime. We therefore recommend $L = 3$ as a conservative default and $L \in \{5, 7\}$ when exploratory diagnostics (e.g., intraday heatmaps such as Fig. 7) indicate strong within-day heterogeneity and sufficient sample size per bin.

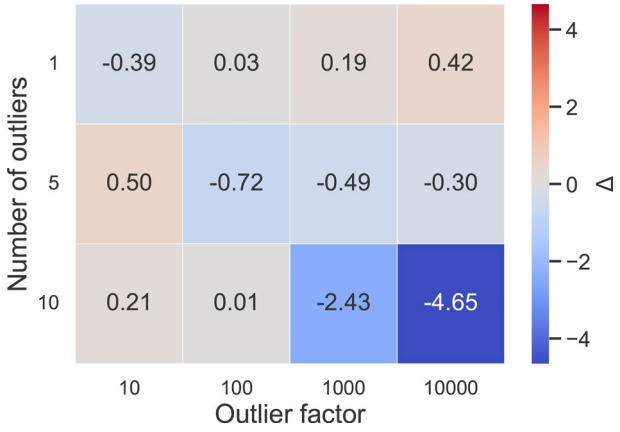


Fig. 9 Difference heatmap illustrating the relative impact of injected outliers on arrival modeling performance. The heatmap shows the signed difference in normalized CADD values between AT-KDE and an exponential baseline (Δ). Negative values (blue) indicate a smaller relative performance degradation for AT-KDE, while positive values (red) indicate a smaller relative degradation for the exponential model

Finally, we note that CADD emphasizes aggregate deviations and may underweight subtle intraday mismatches. We therefore view the quantitative sensitivity results as complementary to the qualitative intraday diagnostics presented earlier.

Sensitivity of AT-KDE to Outliers To analyze the effect of outliers on arrival modeling performance, we conduct a controlled synthetic experiment in which extreme inter-arrival times were injected into an otherwise stationary arrival process. Specifically, 1000 arrival timestamps were generated by iteratively sampling inter-arrival times from an exponential distribution with a scale parameter of $\lambda = 30$ minutes. To introduce irregularities, we randomly injected $n \in \{1, 5, 10\}$ outliers by resampling selected inter-arrival times from the exponential distribution whose scale parameter λ was multiplied by an outlier factor $f \in \{10, 100, 1000, 10000\}$. Larger values of f therefore correspond to increasingly extreme deviations, resulting in substantially longer inter-arrival times compared to the baseline process.

The first 80% of the generated arrival timestamps are used for training, while the remaining 20% form the test set. We compare AT-KDE against a static exponential arrival model (being part of *Best Distribution*) in terms of their ability to reproduce the arrival behavior observed in the test period. To isolate the relative effect of injected outliers, the CADD values obtained for each outlier configuration are normalized by the corresponding CADD measured on the same dataset without outliers. Consequently, the reported values reflect the relative performance degradation induced by outliers for each method, rather than differences in absolute CADD values between methods.

Figure 9 presents a difference heatmap based on these normalized values, visualizing the signed difference between AT-KDE and the exponential baseline. Negative values (shown in blue) indicate that AT-KDE is less affected by the injected outliers relative to its baseline performance, whereas positive values (shown in red) indicate a smaller relative performance degradation for the exponential model. The differences are displayed using a signed logarithmic transformation, which preserves the direction of the effect while compressing large magnitudes.

Overall, the results indicate that neither AT-KDE nor the static *Exponential* baseline is uniformly more robust to outliers than the other. Instead, the relative performance depends on both the frequency and severity of the injected irregularities. In scenarios

with few outliers or moderate outlier factors, the exponential model often exhibits a smaller relative performance degradation. This behavior is expected, as the exponential distribution naturally accommodates rare long inter-arrival times through its heavy-tailed form, whereas AT-KDE explicitly represents such deviations in the learned density, even when they occur infrequently. As the number and magnitude of outliers increase, the relative degradation of AT-KDE becomes comparable to, and in many settings smaller than that of the exponential baseline. Importantly, AT-KDE does not exhibit spurious improvements in low-structure regimes, nor does it deteriorate disproportionately under extreme noise. Overall, these findings suggest that AT-KDE responds to outliers in a controlled and stable manner and becomes competitive primarily when irregularities accumulate, rather than universally outperforming static arrival models.

Evaluation 2: effect on simulation quality

Having demonstrated that our AT-KDE approach produces more accurate case arrival times than existing approaches in Sect. 5.1, we now examine how these improvements translate into impact on overall simulation quality. We first showcase a novel, utility-centered, evaluation framework in subsubsection 5.2.1. Then, subsubsection 5.2.2 details the experimental setup, and subsubsection 5.2.3 reports how different arrival modeling approaches affect simulation quality under this framework.

Utility-based evaluation framework

The standard evaluation of simulation outcomes in process mining relies primarily on EMD-based distances, including the CADD metric used in Sect. 5.1 (Chapela-Campa et al. 2023). Although such metrics capture distributional similarity, they say little about the utility of simulated event logs, particularly under strong temporal dynamics. Following the utility-centered framework of Özdemir et al. (2025), we assess simulation models by how well simulated data supports representative process behavior in downstream tasks. Concretely, instead of comparing simulated logs to held-out real executions via distance metrics, we train predictive process monitoring (PPM) models on simulated data and evaluate whether their performance is comparable to PPM models trained on real data. The resulting performance gap to the real-data baseline provides a direct measure of utility in the simulated logs.

Motivation We argue that the standard evaluation of BPS models comprises two key issues. In the following, we shortly summarize these issues and refer the interested reader to Özdemir et al. (2025) for more details and proofs.

First, the standard evaluation method frames simulation as a forecasting problem, evaluating the quality of a BPS model by comparing simulated event logs to unseen future process executions. We argue that this does not evaluate the model's accuracy in capturing the as-is process, particularly when the process behavior in the test period substantially differs from that of the training period. As a result, the standard evaluation makes it difficult to determine whether poor results stem from an inaccurate BPS model or from dynamics uncaptured in the training data, such as an increasing workload due to higher customer demand reflected by more frequent case arrivals during the test period.

Second, the EMD, as used in many metrics of the standard evaluation (e.g., CADD), can obscure temporal patterns entirely. Moreover, it tends to be biased towards favoring

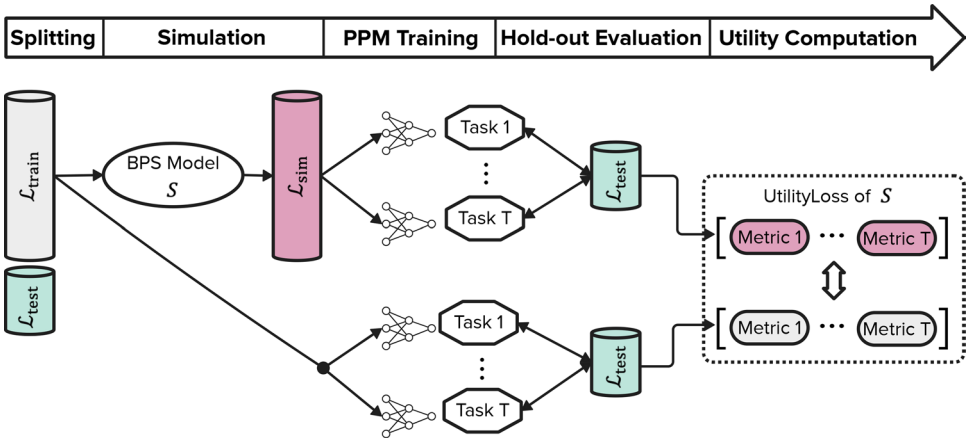


Fig. 10 Overview of proposed BPS evaluation framework

estimates close to the mean of a ground-truth distribution, potentially yielding misleading conclusions about simulation quality.

In light of these limitations, we propose to use a framework that is robust to varying temporal dynamics and capable of quantifying simulation utility.

Framework Description The core idea is to assess the quality of BPS models by measuring how well their simulated data supports downstream prediction tasks compared to real training data. As shown in Fig. 10, our proposed evaluation framework consists of five steps. We describe each of these next, while also highlighting key differences to the current standard evaluation.

Splitting The first step is to split an event log temporally into train log $\mathcal{L}_{\text{train}}$ and test log $\mathcal{L}_{\text{test}}$.

Simulation Then, $\mathcal{L}_{\text{train}}$ is used to train a BPS model S. Unlike the standard evaluation, which aims to align $\mathcal{L}_{\text{sim}}$ with the characteristics of $\mathcal{L}_{\text{test}}$—implicitly treating simulation as a forecasting task and leading to an objective mismatch—we ensure that S generates a simulated log $\mathcal{L}_{\text{sim}}$ that mirrors the properties of $\mathcal{L}_{\text{train}}$. Specifically, $\mathcal{L}_{\text{sim}}$ should contain the same number of cases as $\mathcal{L}_{\text{train}}$ and start at the same point in time.

PPM Training Next, PPM models are fit w.r.t. T different *downstream* tasks, separately using real data $\mathcal{L}_{\text{train}}$ and simulated data $\mathcal{L}_{\text{sim}}$. This results in two distinct predictive models per task $t \in \{1, \ldots, T\}$: $\text{PM}_t(\mathcal{L}_{\text{train}})$ and $\text{PM}_t(\mathcal{L}_{\text{sim}})$. To support a comprehensive evaluation, the framework can be instantiated with a broad set of PPM prediction tasks. If all four process perspectives commonly considered in BPS evaluation (Chapela-Campa et al. 2023)—*control-flow, resource, temporal, congestion*—are of interest, one may use a standard task selection covering these dimensions: next activity prediction (NAP), next role prediction (NRP), next processing time prediction (NPP), next waiting time prediction (NWP), and remaining time prediction (RTP). Additional tasks can be added when required by the application. Conversely, when the evaluation targets a single perspective (e.g., the congestion dimension), one should restrict the task set accordingly, for instance to NWP and RTP.

Hold-out Evaluation The PPM models fit to both real data ($\mathcal{L}_{\text{train}}$) and simulated data ($\mathcal{L}_{\text{sim}}$) are assessed by their performance on the hold-out test log ($\mathcal{L}_{\text{test}}$). Each task $t \in \{1, \ldots, T\}$ is evaluated using a specific metric $\mathcal{M}_t$, such as *MAE* for RTP and accuracy for NAP. Rather than combining these metrics into a single score, we maintain a

process perspective-specific evaluation to preserve interpretability. Thus, for an event log $\mathcal{L} \in \{\mathcal{L}_{\text{train}}, \mathcal{L}_{\text{sim}}\}$, the resulting metrics for T tasks—averaged over K model architectures—are represented as a vector:

$$\mathcal{M}(\mathcal{L}) = \left[\frac{1}{K} \sum_{k=1}^{K} \mathcal{M}_1^{\mathcal{L}_{\text{test}}}(\text{PM}_1^k(\mathcal{L})), \ldots, \frac{1}{K} \sum_{k=1}^{K} \mathcal{M}_T^{\mathcal{L}_{\text{test}}}(\text{PM}_T^k(\mathcal{L})) \right] \tag{1}$$

Utility Computation To evaluate how well a BPS model S preserves the predictive utility of the original process data, the performance gap between PPM models trained on real versus simulated logs across all downstream tasks is quantified as follows. As introduced above, the task-specific performance metrics computed on the hold-out log yield the vectors $\mathcal{M}(\mathcal{L}_{\text{train}})$ and $\mathcal{M}(\mathcal{L}_{\text{sim}})$, which serve as proxies for the practical utility of $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{sim}}$, respectively. The *utility loss* of S is then defined as the element-wise absolute difference between these vectors, measuring how much performance is lost when training on simulated instead of real data. Small absolute deviations in the task-specific metrics indicate that S successfully preserves the predictive structure of $\mathcal{L}_{\text{train}}$, whereas large deviations—in either direction—signal a loss of fidelity. Formally,

$$\text{UtilityLoss}(S) = \left| \mathcal{M}(\mathcal{L}_{\text{train}}) - \mathcal{M}(\mathcal{L}_{\text{sim}}) \right|. \tag{2}$$

Applicability Assessment Before applying the framework to simulations obtained from different case arrival approaches, we first validate its applicability by examining whether controlled modifications to a BPS model lead to the expected loss of utility. Following Özdemir et al. (2025), an effective evaluation should (i) identify inaccurate models and (ii) reveal which process perspective they fail to capture appropriately. To this end, we focus on the synthetic evaluation of Özdemir et al. (2025), which is based on the *Loan Application* process designed by Chapela-Campa et al. (2023).

Ground-Truth Data In total, the respective event log contains 1000 traces and 7492 events. We temporally split it trace-wise into 80% training (average cycle time of 0.42 days) and 20% test data (average cycle time of 0.46 days). The underlying process comprises 12 activities, starting with *Check application form completeness*. Its control-flow structure includes a loop, a parallel branch involving three activities, three exclusive gateways, and three possible end points: *Approve application*, *Reject application*, and *Cancel application*. In total, 19 distinct resources execute the process.

Scenarios Given the training log of this process, we use the *Simod* BPS approach (Camargo et al. 2020) to discover a ground-truth simulation model (Loan_{GT}) and derive several modified variants, following Chapela-Campa et al. (2023). These variants allow us to assess whether our framework penalizes BPS models in the respective downstream tasks when they deviate from the ground truth. We emphasize the *congestion* perspective, as it aligns most directly with the objective of accurate arrival modeling. For each of the following models, we simulate 10 event logs that match the training log in terms of start time and number of cases:

- Loan_{SEQ}: arranging the three parallel activities as a sequence.
- Loan_{S-G}: altering, on top of Loan_{SEQ}, the branching probabilities.
- Loan_{RC}: halving the available resources.
- Loan_{EXT}: adding extraneous waiting time to delay the start of activities.
- Loan_{DUR}: increasing the duration of the activities of the process.

Table 7 Average utility loss (with standard deviation) for modifications of the LOAN_{GT} process. NPP, NWP, and RTP errors are measured in minutes

	NAP	NRP	NPP	NWP	RTP
LOAN_{GT}	0.71 (0.01)	0.75 (0.00)	67.60 (2.80)	12.13 (1.48)	238.26 (8.38)
LOAN_{SEQ}	0.26 (0.03)	0.26 (0.04)	16.12 (3.29)	12.03 (0.52)	35.88 (18.04)
$\text{LOAN}_{S\text{-}G}$	0.32 (0.07)	0.35 (0.04)	12.40 (2.91)	715.84 (112.23)	4131.58 (1802.63)
LOAN_{RC}	0.00 (0.01)	0.12 (0.14)	16.55 (3.31)	164.28 (63.69)	771.89 (368.17)
LOAN_{EXT}	0.00 (0.01)	0.00 (0.00)	24.30 (5.19)	74.37 (6.92	508.74 (53.15)
LOAN_{DUR}	0.00 (0.01)	0.00 (0.00)	49.63 (7.42)	7.27 (6.01)	263.29 (39.86)
LOAN_{CAL}	0.00 (0.01)	0.00 (0.00)	0.44 (5.38)	0.45 (1.68)	5.43 (8.93)
LOAN_{ARR}	0.00 (0.01)	0.00 (0.00)	10.60 (2.28)	35.14 (3.14)	90.1 (15.21)

- Loan_{CAL}: changing resource working schedules from 9am-5pm to 2pm-10pm.
- Loan_{ARR}: increasing the rate of case arrivals.

Tasks, Metrics, PPM Approach For downstream utility assessment, we consider the five PPM tasks described above. We use *accuracy* as the metric for the two classification tasks (NAP and NRP) and *MAE* for the three regression tasks (NPP, NWP, and RTP). As predictive architecture for all tasks, we employ *ProcessTransformer* (Bukhsh et al. 2021). Utility loss for each modified model is then computed from the task-specific metrics as defined in Eq. 2.

Impact on Process Perspectives Table 7 reports the predictive performance obtained with simulated data from the ground-truth model Loan_{GT} and the corresponding utility losses for each modification, illustrating how the framework detects alterations in the respective process perspectives.

Congestion In the *congestion* perspective, Loan_{S-G}, Loan_{RC}, Loan_{EXT}, and Loan_{ARR} exhibit notable losses in NWP, reflecting the expected increase in waiting times due to sequentialization, resource contention, added delays, and higher arrival rates. These congestion effects also propagate to RTP, where extended waiting times accumulate into substantially longer cycle times. Notably, the scenario with an increased case arrival rate (Loan_{ARR}) yields the smallest UtilityLoss among the four scenarios. The reason is that the available resources still provide sufficient capacity to handle the higher workload without causing a disproportionate increase in waiting times. Moreover, activity execution times in this process are generally short, so processing more cases has a comparatively smaller impact than restructuring the control-flow, as done in the Loan_{S-G} scenario.

Other Perspectives For the *control-flow* perspective, only modifications that alter the activity order (Loan_{SEQ} and Loan_{S-G}) are expected to differ from the ground truth, and the framework assigns the corresponding UtilityLoss accordingly. In the *resource* perspective, deviations arise not only from altered control-flow (Loan_{SEQ}, Loan_{S-G}) but also from reduced resource availability (Loan_{RC}). The framework detects these changes through increased NRP losses. For the *temporal* perspective, the strongest effect appears in Loan_{DUR}, where modified activity durations result in substantial UtilityLoss in NPP and spill over into RTP, showing the expected propagation of timing deviations. Finally, Loan_{CAL}—which alters only absolute timestamps—correctly receives near-zero UtilityLoss, demonstrating that the framework evaluates temporal relationships rather than absolute time values.

Overall, these results confirm that the framework not only detects deviations from the ground-truth model but also accurately pinpoints the specific process perspective in

which each deviation occurs. This provides a meaningful basis for comparing the simulation quality obtained from different arrival modeling approaches, as we do next.

Experimental setup

Evaluation Data For this second evaluation, we use five event logs from Table 2: *BPIC12W*, *ACR*, *P2P*, *Production*, and *Confidential 1000*. We focus on these logs because most available datasets do not provide start and end timestamps for each event, which is essential for data-driven simulation. These logs are therefore widely used in BPS research (López-Pintado and Dumas 2022; Meneghello et al. 2023; Kirchdorfer et al. 2024).

Data Split Consistent with Evaluation 1, we apply a temporal hold-out split with 80% training and 20% testing data. As before, we train the arrival models on the training split. However, unlike in Evaluation 1, we do not generate arrivals for the test period. Instead, each model generates case arrivals that replicate the training period. This aligns with our framework, which evaluates how well simulated data preserves the predictive utility of the original training data.

Benchmark Approaches We compare AT-KDE against the following benchmark approaches: two static approaches (*Mean*, *Best Distribution*) and two dynamic approaches (*LSTM* and *XGBoost*).

Simulation We use *AgentSimulator* (Kirchdorfer et al. 2024) as the simulation discovery model and engine. For each arrival modeling approach, we replace only the arrival component of *AgentSimulator* while keeping all other components identical, ensuring that any differences in simulation quality are attributable solely to the arrival model. For every event log and arrival approach, we generate 10 simulated event logs, each mirroring the temporal span of the training period. The resulting simulations are then used to train the downstream PPM models within our utility-based framework.

PPM Tasks and Approach Since the arrival modeling approaches affect only the case arrival pattern, we focus on congestion-related prediction tasks within the utility-based framework, namely *NWP* and *RTP*. As predictive architecture, we employ the LSTM model variant proposed by Camargo et al. (2019). For each process and arrival approach, we train one PPM model on each of the 10 simulated logs and one PPM model on the true training log. The task-specific metrics are averaged across the 10 simulated replications to serve as $\mathcal{M}(\mathcal{L}_{\text{sim}})$, and over the true training data to provide $\mathcal{M}(\mathcal{L}_{\text{train}})$. Both vectors are used to compute the utility loss as defined in Eq. 2.

Results

The results of Evaluation 2 are reported in Table 8. For each event log (row) and arrival model (column), we show the utility loss vector $\text{UtilityLoss}(S) \in \mathbb{R}^2_+$, whose components correspond to the NWP and RTP tasks, respectively. The final row contains the task-wise average across event logs.

Overall, AT-KDE achieves the lowest average utility loss for both NWP and RTP. While the improvement over the benchmarks in RTP is relatively modest, the gains in NWP are substantially more pronounced.

In general, the different arrival modeling approaches appear to influence overall simulation quality only moderately. For example, in the *BPIC12W* log, the utility losses for both tasks are nearly identical across all approaches. This outcome is consistent with our findings in Evaluation 1 (Sect. 5.1), where the models achieved similar performance

Table 8 Average utility loss per event log and arrival modeling approach. Errors measured in hours. Missing entries based on convergence failure

Event log	Task	Mean	Best Dist.	LSTM	XGBoost	AT-KDE
ACR	NWP	0.01	0.02	0.01	0.02	0.01
	RTP	0.04	0.15	0.01	0.11	0.05
P2P	NWP	0.72	0.71	0.73	0.67	0.63
	RTP	23.07	22.53	22.34	22.61	20.04
Production	NWP	36.17	41.81	31.50	20.84	20.48
	RTP	54.42	53.37	54.56	54.77	54.64
Conf. 1000	NWP	6.72	6.98	0.04	/	0.01
	RTP	2.45	1.90	2.30	/	1.87
BPIC12W	NWP	0.02	0.02	0.02	0.02	0.02
	RTP	5.27	4.91	4.17	5.22	5.53
Average	NWP	8.73	9.91	6.46	5.39	**4.23**
	RTP	17.05	16.57	16.68	20.68	**16.43**

with respect to root-CADD, likely due to the absence of substantial temporal shifts in the ground-truth data (Fig. 8b).

At the same time, several logs exhibit clear advantages of AT-KDE. In the *Production* log, the NWP error under AT-KDE is approximately half that of the *Best Distribution* approach. Likewise, AT-KDE outperforms all baselines in RTP on the *P2P* dataset and generally delivers the best results across both tasks for *Confidential 1000*.

Despite these improvements, the overall gains remain modest. We attribute this to two main factors. First, in three of the five datasets AT-KDE did not achieve the best performance in Evaluation 1, so large improvements in the simulation-based evaluation are not to be expected. Second, the arrival model constitutes only one of several components in a BPS model. Given that the benchmark arrival models are not particularly poor, their negative impact on overall simulation quality is inherently limited. Nevertheless, AT-KDE consistently yields the most accurate and robust results across datasets, underlining its potential as a default arrival modeling choice in BPS.

Conclusion

In this work, we introduced *Auto Time Kernel Density Estimation* (AT-KDE), an approach for learning case-arrival models from data and generating new arrival times for BPS. AT-KDE combines a divide-and-conquer strategy with kernel density estimation to capture global, weekday, and intraday arrival dynamics effectively. By partitioning the arrival data into segments and modeling each with its own KDE, the approach produces new arrival timestamps that reflect the dynamic nature of organizational processes, thereby supporting more accurate simulation outcomes. Our extensive evaluations underscore the importance of incorporating such temporal dynamics: AT-KDE consistently achieves higher accuracy and robustness in generating realistic arrival times than existing approaches, while maintaining practical runtime efficiency. In our analysis of how different case-arrival models affect overall simulation quality, the absolute differences between approaches were generally modest. Nonetheless, despite these relatively small differences, AT-KDE consistently demonstrated the strongest results on average. This indicates that even when the overall impact of the arrival model on simulation outcomes is limited, AT-KDE's ability to more faithfully capture arrival dynamics still translates into measurable and typically superior simulation quality.

Limitations Despite its strong empirical performance, AT-KDE is still confronted with limitations. First, its effectiveness relies on careful calibration tailored to each process to ensure optimal performance. Moreover, if future arrivals should be generated and the training data lacks clear patterns, AT-KDE falls back on leveraging only the most recent behavior—a heuristic that, while usually adequate, can miss subtler trends captured by more complex deep-learning models.

Future Work While initially developed for modeling arrival times, our divide-and-conquer approach may be extended to capture further process dynamics, such as evolving activity durations or delays. Furthermore, its underlying principles are broadly applicable beyond BPS, offering value in any domain where time-dependent behavior plays a critical role in arrival modeling.

Acknowledgements
We thank Stjepan Kusenic for his support during the initial development of our approach. We are also grateful to our colleague Keyvan Amiri Elyasi for his contributions to the evaluation framework. Finally, we thank Timotheus Kampik for his valuable feedback on the initial idea.

Author contributions
All authors contributed to the work's conception; L.K. and K.Ö. suggested the main idea, implemented the approach, and conducted the experiments; L.K., K.Ö., and H.A. wrote and revised the manuscript throughout all its versions.

Funding
Open Access funding enabled and organized by Projekt DEAL. No funding was received to assist with the preparation of this manuscript.

Data availability
The implementation, datasets, and additional results are available through the sources linked in the Evaluation section.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 27 November 2025 / Accepted: 7 April 2026
Published online: 29 April 2026

References

Ansari AF, Stella L, Turkmen C et al (2024) Chronos: learning the language of time series. TMLR
Bukhsh ZA, Saeed A, Dijkman RM (2021). Processtransformer: predictive business process monitoring with transformer network. arXiv preprint:2104.00721
Camargo M, Dumas M, González-Rojas O (2020) Automated discovery of business process simulation models from event logs. Decis Support Syst 134:134. https://doi.org/10.1016/j.dss.2020.113284
Camargo M, Dumas M, Rojas OG (2019) Learning accurate LSTM models of business processes. In: BPM. Springer
Camargo M, Dumas M, Rojas OG (2022) Learning accurate business process simulation models from event logs via automated process discovery and deep learning. In: CAiSE. Springer
Chapela-Campa D, Benchekroun I, Baron O, Dumas M, Krass D, Senderovich A (2023) Can I trust my simulation model? measuring the quality of business process simulation models. In: BPM. Springer
Chen T, Guestrin C (2016) Xgboost: a scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'16, Association for Computing Machinery, New York, NY, USA, 785–794
Daw A, Pender J (2018) Queues driven by hawkes processes. Stochastic Syst 8(3):192–229. https://doi.org/10.1287/stsy.2018.0014
Dumas M (2021) Constructing digital twins for accurate and reliable what-if business process analysis. In BPM Workshops, vol 2938. pp 23–27. CEUR-WS.org
Dumas M, Rosa ML, Mendling J, Reijers HA (2013) Fundamentals of business process management. Springer
Ester M, Kriegel HP, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD. AAAI Press
Hamilton JD (1994) Time series analysis. Princeton University Press
Hochreiter SLL, Schmidhuber J (1997 November) Long short-term memory. Neural Computation 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 6(2):65–70

Kirchdorfer L, Blümel R, Kampik T, van der Aa H, Stuckenschmidt H (2024) Agentsimulator: an agent-based approach for data-driven business process simulation. In ICPM, IEEE, pp. 97–104
Kirchdorfer L, Özdemir K, Kusenic S, H, Stuckenschmidt H (2025, May). Codebase: a divide-and-conquer approach for modeling arrival times in business process simulation
Kirchdorfer L, Özdemir K, Kusenic S, van der Aa H, Stuckenschmidt H (2025) A divide-and-conquer approach for modeling arrival times in business process simulation. BPM, Volume 16044 Lect Notes Comput Sci 325–342
Kontopoulou VI, Panagopoulos AD, Kakkos I, Matsopoulos GK (2023) A review of arima vs. machine learning approaches for time series forecasting in data driven networks. Future Internet 15(8):255. https://doi.org/10.3390/fi15080255
Lindemann B, Müller T, Vietz H, Jazdi N, Weyrich M (2021) A survey on long short-term memory networks for time series prediction. Procedia Cirp 99:650–655. https://doi.org/10.1016/j.procir.2021.03.088
López-Pintado O, Dumas M (2022) Business process simulation with differentiated resources: does it make a difference? In: BPM. Springer
Martin N, Depaire B, Caris A (2015) Using event logs to model interarrival times in business process simulation. In: BPM workshops. Springer, pp 255–267
Meneghello F, Francescomarino CD, Ghidini C (2023) Runtime integration of machine learning and simulation for business processes. In ICPM, IEEE
Özdemir K, Kirchdorfer L, Elyasi KA, Aa HVD, Stuckenschmidt H (2025) Rethinking business process simulation: a utility-based evaluation framework. In: BPM. Springer, pp 128–144
Parzen E (1962) On Estimation of a probability density function and mode. In: The annals of mathematical statistics *33*(3). , pp 1065–1076
Porto BM, Fogliatto FS (2024) Enhanced forecasting of emergency department patient arrivals using feature engineering approach and machine learning. BMC Med Inf Decis Mak 24(1):377. https://doi.org/10.1186/s12911-024-02788-6
Rey D, Neuhäuser M (2011. Wilcoxon-Signed-Rank Test, pp. Springer, Berlin, Heidelberg Berlin Heidelberg, pp 1658–1659
Rozinat A, Mans RS, Song M, van der Aalst WMP (2009) Discovering simulation models. Inf Syst 34(3):305–327. https://doi.org/10.1016/j.is.2008.09.002
Silverman BW (1986) Density Estimation for statistics and data analysis. Chapman & Hall, London
Skillings JH, Mack GA (1981) On the use of a friedman-type statistic in balanced and unbalanced block designs. Technometrics 23(2):171–177. https://doi.org/10.1080/00401706.1981.10486261
Taylor SJ, Letham B (2018) Forecasting at scale. Am Sci 72(1):37–45. https://doi.org/10.1080/00031305.2017.1380080
van der Aalst WMP (2015) Business process simulation survival guide. In: Rosemann M, International Handbooks on Information Systems, ed vom Brocke, J. and Handbook on business process management 1, introduction, methods, and information Systems, 2nd ed. Springer, pp 337–370
Villani C (2008) *Optimal transport: old and New*. Grundlehren der mathematischen Wissenschaften. Springer, Berlin Heidelberg
Ward Jr JH (1963) Hierarchical grouping to optimize an objective function. J Am Stat Assoc 58(301):236–244. https://doi.org/10.1080/01621459.1963.10500845

Publisher's Note