Generating the Traces You Need: A Conditional Generative Model for Process Mining Data

Riccardo Graziosi[§]
Fondazione Bruno Kessler, Trento, Italy
rgraziosi@fbk.eu

Massimiliano Ronzani[§]
Fondazione Bruno Kessler, Trento, Italy
mronzani@fbk.eu

Andrei Buliga
Fondazione Bruno Kessler, Trento, Italy
Free University of Bozen-Bolzano, Bolzano, Italy
abuliga@fbk.eu

Chiara Di Francescomarino
University of Trento, Trento, Italy
c.difrancescomarino@unitn.it

Francesco Folino
ICAR-CNR, Rende, Italy
francesco.folino@icar.cnr.it

Chiara Ghidini
Free University of Bozen-Bolzano, Bolzano, Italy
chiara.ghidini@unibz.it

Francesca Meneghello
Fondazione Bruno Kessler, Trento, Italy
Sapienza University of Rome, Rome, Italy
fmeneghello@fbk.eu

Luigi Pontieri
ICAR-CNR, Rende, Italy
luigi.pontieri@icar.cnr.it

2024 6th International Conference on Process Mining (ICPM) | 979-8-3503-6503-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICPM63005.2024.10680621

***Abstract*—In recent years, trace generation has emerged as a significant challenge within the Process Mining community. Deep Learning (DL) models have demonstrated accuracy in reproducing the features of the selected processes. However, current DL generative models are limited in their ability to adapt the learned distributions to generate data samples based on specific conditions or attributes. This limitation is particularly significant because the ability to control the type of generated data can be beneficial in various contexts, enabling a focus on specific behaviours, exploration of infrequent patterns, or simulation of alternative "what-if" scenarios.**

In this work, we address this challenge by introducing a conditional model for process data generation based on a conditional variational autoencoder (CVAE). Conditional models offer control over the generation process by tuning input conditional variables, enabling more targeted and controlled data generation. Unlike other domains, CVAE for process mining faces specific challenges due to the multiperspective nature of the data and the need to adhere to control-flow rules while ensuring data variability. Specifically, we focus on generating process executions conditioned on control flow and temporal features of the trace, allowing us to produce traces for specific, identified sub-processes. The generated traces are then evaluated using common metrics for generative model assessment, along with additional metrics to evaluate the quality of the conditional generation.

***Index Terms*—Process Mining, Deep Learning, Generative AI, Conditional models**

I. Introduction

Process mining (PM) [1] is a research field that focuses on the analysis, monitoring, and improvement of business processes based on event logs. Within this field, generative models have emerged in recent years as crucial tools for generating new event trace samples that replicate process behavior [2]–[9]. These models support a range of applications, including anomaly detection [7], [10], predictive monitoring [2], what-if scenario analysis [11] and conformance checking [12]. An important yet underexplored aspect of trace generation is the ability to produce traces that follow different distributions from the training data, allowing exploration of various dimensions of interest within the process. These dimensions may include exploring *what-if* scenarios, expanding variants of interest (especially when significant for the process analysis but numerically low), or exploring resource contingency plans.

According [13], generative models can be categorized into two main families: Data-Driven Process Simulation (DDPS) and Deep Learning (DL). DDPS constructs explicit process models from data, esnuring that complete information about the simulation is always available. These models are beneficial for providing insights into specific subprocesses and allow to modify almost every aspect of the simulation. However, DDPS often relies on oversimplified assumptions, leading to unrealistic simulations and data generation. Moreover, they struggle to capture long-term dependencies. DL models are statistical models that accurately capture the correlations between features in the generated samples. Despite their accuracy, DL models are "black box" systems, making it challenging to transparently expose the underlying process model. More importantly, existing DL-based generative models are rigid, limiting the generation of distributions different from the training data. This constraint significantly inhibits the exploration of specific scenarios or dimensions of interest. Hybrid models, which integrate the accuracy of DL techniques with the transparency of explicit process models, have been recently introduced [14], [15]. While they may have potential to

We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU, the HORIZON 2020 HumanE-AI project (Grant 952026), and the PRIN project PINPOINT Prot. 2020FNEB27, CUP H23C22000280006 and H45E2100021000.

[§]Equal contribution

generate specific data for dimensions of interest, an assessment of this capability is still missing.

Conditional generative models [16] have been proven effective in various domains to mitigate the rigidity observed in DL generative models. These models generate outputs influenced by certain input variables. This offers a means to guide the generation process based on desiderata for the expected output.

In this work, we introduce a conditional variational autoencoder (CVAE) model for generating traces based on LSTM neural networks. Compared to other domains, developing a CVAE for process mining presents two main challenges:

- Process execution data have an inherently multi-perspective nature. Traces consist of sequence of temporal features with both categorical (events) and numerical (timestamps) characteristics. This complexity is further increased by the inclusion of payloads. Moreover, all these components are strongly interconnected. Therefore, due to their diverse nature, each component of the generated samples requires a dedicated module in the model architecture, which must still produce a coherent output.
- While it is desirable for the generative model to produce data with some variability, event sequences must adhere to causal constraints and diverse contextual factors. Thus, the variability must remain consistent with process constraints to ensure meaningful results.

A further contribution of our work is the proposal of a novel evaluation methodology for assessing process generation quality in a conditional context. Alongside common metrics for evaluating generative process models, such as the accuracy of the control flow and generated timestamps [17], we introduce a new analysis to measure the impact of the conditioning variable and measure the actual conditioning rate. Moreover, unlike many simulation scenarios that aim to closely replicate historical data, generative models in a conditional context should introduce variability in the generated output while remaining within process constraints. Thus, our evaluation framework includes metrics to assess the variability and conformance of the generated traces with the process rules.

By integrating conditional models into process data generation, this work aims to enhance the flexibility and control of DL generative models in Process Mining. Evaluation on four different examples based on three real-world event logs shows promising results. Specifically, the generated traces are accurate, exhibit good variability, comply with process constraints, and demonstrate that the conditional generative model effectively controls the types of traces produced.

II. Background

In this section we introduce the main concepts useful to understand the remainder of the paper.

A. Event Log

An event log $\mathcal{L}$ records the executions of a business process in terms of execution *traces*. A trace x consists of a sequence of ordered events $x = \langle e_1, e_2, \dots e_n \rangle$. Events are characterized by multiple attributes (*data attributes*): primarily, an event refers to an activity label and a timestamp indicating when the activity was executed. It may also include information about the resource executing or initiating the activity, and other data attributes. Some attributes are static and consistent throughout the trace's execution, known as *trace attributes*. Data associated with events and traces in event logs are also called *data payloads*. We can represent a trace x as:

$$x = \langle (a_1, T_1, \mathbf{d}_1), \dots, (a_n, T_n, \mathbf{d}_n) \rangle \quad (1)$$

where a_i is the i-th activity executed in the trace, T_i is its timestamp, and $\mathbf{d}_i$ is a vector containing its data payload, including static trace attributes.

B. Conditional Variational Autoencoders (CVAEs)

Autoencoders are neural network architectures used for unsupervised learning tasks [18]. They consist of an encoder, E_ϕ, and a decoder, D_θ, representing non-linear transformations parametric to θ and ϕ, respectively. The encoder maps the input data x into a latent representation $\mathbf{z} = E_\phi(x)$, while the decoder output $\hat{x} = D_\theta(\mathbf{z})$ should reconstruct the original input from the latent representation. Mathematically, an autoencoder aims to minimize the reconstruction error between the input and the output:

$$\mathcal{L}_{\text{AE}}(x, \theta, \phi) = J_{rec}(x, \hat{x}). \quad (2)$$

where J_{rec} denotes some kind of distance/error function over the data space (e.g., $\|x - \hat{x}\|^2$ in the case $x, \hat{x} \in \mathbb{R}^N$).

Variational autoencoders (VAEs) [19] lift this basic autoencoder architecture to the level of a generative model where x and $\mathbf{z}$ are interpreted as observed and latent random variables, respectively, such that the joint distribution of x and $\mathbf{z}$ is factored as $p_\theta(x|\mathbf{z}) \cdot p(\mathbf{z})$, where $p_\theta(x|\mathbf{z})$ is a distribution to be learned, and the latent prior $p(\mathbf{z})$ is typically set to a standard multivariate Gaussian distribution (i.e., $p(\mathbf{z}) = \mathcal{N}(\vec{0}, \mathbf{I})$, where $\mathbf{I}$ is an identity matrix).[1]

VAEs are trained to minimize the (expectation over the real data distribution of the) following negative *Evidence Lower Bound* (ELBO) $\mathcal{L}_{\text{VAE}}$, consisting of a reconstruction loss term (echoing that in Eq. (2)) plus a regularization term, which is computed as the Kullback-Leibler (KL) divergence between a learned latent distribution $q_\phi(\mathbf{z}|x)$ and the latent prior $p(\mathbf{z})$, with a factor β controlling the strength of the regularization:

$$\mathcal{L}_{\text{VAE}}(x, \theta, \phi) = J_{\text{VAE}}(x, \theta, \phi) + \beta \cdot \text{KL}\big(q_\phi(\mathbf{z}|x) \,||\, p(\mathbf{z})\big) \quad (3)$$

where ϕ and θ denote the parameters of the encoder and decoder sub-nets, now modelling the learned distributions $q_\phi(\mathbf{z}|x)$ and $p_\theta(x|\mathbf{z})$, respectively, while the reconstruction term is $J_{\text{VAE}}(x, \theta, \phi) = -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|x)} \ln p_\theta(x|\mathbf{z})$.

Conditional variational autoencoders (CVAEs) incorporate conditional information into both the encoder and the decoder. In CVAEs, both the encoder and the decoder take the input data x and conditioning variables c as inputs [16]. The conditional variable c represents the specific condition or attribute that

[1]This allows for easily generating any $\hat{x}$ using a two-phase scheme: first $\mathbf{z}$ is sampled from the latent prior $p(\mathbf{z})$, and then $\hat{x}$ is sampled from $p(x \mid \mathbf{z})$.

guides the generation process, enabling the model to produce data samples with desired characteristics. The encoder maps x and c to the parameters of a (variational) posterior distribution $q_\phi(\mathbf{z}|x, c)$ of the latent variables $\mathbf{z}$, while the decoder models a distribution $p_\theta(x|\mathbf{z}, c)$ for generating/reconstructing the input data conditioned on both the latent representation and the conditioning variables. Optimal parameters for both the encoder and decoder are learned by minimizing (the expectation over the real data of) the following negative ELBO objective:

$$\mathcal{L}_{\text{CVAE}}(x, c, \theta, \phi) = J_{\text{CVAE}} + \beta \cdot \text{KL}(q_\phi(\mathbf{z} \mid x, c) \mid\mid p(\mathbf{z} \mid c)) \quad (4)$$

where $q_\phi(\mathbf{z}|x, c)$ is the (learned) conditional posterior distribution of the latent variables (given both the observed data and conditioning variables), $J_{\text{CVAE}}(x, c, \theta, \phi) = -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|x,c)} \ln p_\theta(x|\mathbf{z}, c)$, and $p(\mathbf{z}|c)$ is a prior distribution (conditioned on c) for the latent variables $\mathbf{z}$. Usually, for any given x and c, $q_\phi(\mathbf{z}|x, c)$ is assumed to be a multivariate Gaussian distribution with a diagonal covariance matrix.

III. Related work

Generative DL models have been extensively studied in recent years in the PM field. The primary idea behind most of these models, in the context of predictive process monitoring, is to generate a trace by iteratively predicting the next activity for a prefix. For instance, [2] introduces an LSTM-based method [20] that generates the remaining sequence of events with associated timestamps, given a trace prefix. This approach uses one-hot encoding to represent activities, which struggles with high-dimensional inputs, i.e., processes with a large number of activities.

In [3], an LSTM neural network is used to generate event sequences, addressing the dimensionality issues with embeddings for activities. However, this method does not handle numerical features and thus cannot generate event timestamps. Other approaches for activity sequence generation include an LSTM-based method in [4], n-grams encoding with neural networks in [5], and Markov models, RNN, and automata-based models in [6]. Despite their variety, these methods share a common limitation: they do not generate timestamps.

In [7] and [10], generative methods based on GRU neural networks and variational auto-encoders respectively, have been applied for anomaly detection.

In [8], the authors combine elements from prior work to build an accurate LSTM-based generative model that generates events with timestamps and associated roles, and can produce traces from scratch using an "hallucination" mechanism. To ensure sufficient variability in the generated traces, the selection of the next events is performed using a random sampling method based on the predicted probability distribution outputted by the model. While this method increases the variability of the generated traces, it may occasionally produce traces inconsistent with the global process distribution. In [9], an LSTM model for predicting the next event and its timestamp, is trained adopting a generative adversarial network (GAN) approach.

A comparison in [13] between these two methods and a variant of [8] with GRU layers shows that the original LSTM-based method in [8] achieves the best results on average. [2]

In this work we provide a further contribution to the state-of-the-art in trace generation (events, timestamps and trace attributes) by employing a conditional variational auto-encoder (CVAE) based on LSTM neural networks. This approach has two main advantages: (i) it allows generating traces with good variability by sampling from the latent space, without adopting weighted random choices on the LSTM output; (ii) it adds control to the generation process by setting the conditional variables. To our knowledge, this is the first DL conditional model applied to trace generation in PM.

IV. Approach

In this section we present a conditional variational autoencoder (CVAE) architecture for the generation of traces. In this work, we consider the generation of traces with only activities a_i, timestamps T_i, and static traces attributes, which can be numerical $\mathbf{d}_{num}$ and categorical $\mathbf{d}_{cat}$. The trace (1) can then be rewritten as $x = \{(\mathbf{d}_{num}, \mathbf{d}_{cat}), \langle(a_1, T_1), \ldots, (a_n, T_n)\rangle\}$.

In order to be able to apply the CVAE to business processes, we had to adapt it by employing encoder and decoder architectures suitable for handling both sequential data (control flow and timestamps) and non-sequential data (trace attributes). The model is depicted in Figure 1.

In particular, for sequential data, we drew inspiration from seq2seq models, which are commonly used in NLP, where an encoder model maps a variable-length input sequence to a fixed size vector, which is then "unrolled" back to a variable-length sequence by a decoder model [23].

The following sections explain the preprocessing steps, the encoder and decoder architectures, the training approach, and the generation process of new traces.

A. Preprocessing

In the preprocessing phase a trace x is properly manipulated to be feed to the encoder. Each event's timestamp T_i is decomposed into two parts: (i) the *trace arrival time* T_1, which corresponds to the timestamp of the first event in the trace, and is handled by the model in the same way as a numerical trace attribute; and (ii) the *event interarrival time* $t_i := T_i - T_{i-1}$. The *event interarrival time* is normalized using the 95th percentile value instead of the maximum value. While this causes some values to be outside the $[0, 1]$ range, we found it useful in practice in order to ignore outlier timestamps, i.e. very high interarrival time, that could cause the normalization to squash *event interarrival times* to an exceedingly narrow range close to zero. Numerical trace attributes are preprocessed by normalizing them to the $[0, 1]$ interval using min-max normalization.

[2]In fact, as discussed in [21], [22], while GANs excel at generating realistic samples (e.g., high-fidelity photos, deep fakes), they often focus on limited portions of the data distribution and are prone to the notorious mode collapse problem, for which a general, effective, solution is still missing. By contrast, VAEs are effective in modeling multimodal distributions, which may well occurr in process logs, while providing control in the generative process [21].

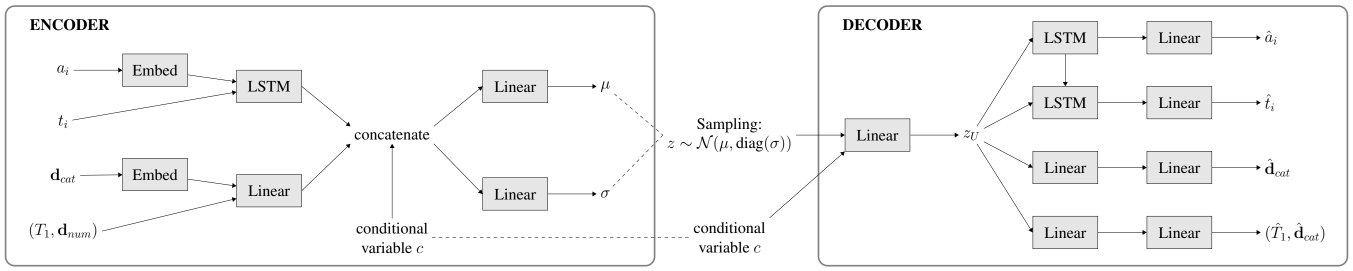

Fig. 1. A high level overview of the CVAE for process mining. Activities a_i and event interarrival times $t_i = T_i - T_{i-1}$ are incrementally processed by the LSTM subnet, while all the categorical $\mathbf{d}_{cat}$ and numerical $\mathbf{d}_{num}$ attributes of x (including trace arrival time T_1) are processed by the Linear subnet.

B. Encoder

The goal of the encoder is to map any trace x and conditioning variable c to the mean vector $\boldsymbol{\mu}_{x,c}$ and variance vector $\boldsymbol{\sigma}_{x,c}$ that fully specify the multivariate Gaussian distribution $q_\phi(\mathbf{z}|x,c) \equiv \mathcal{N}(\boldsymbol{\mu}_{x,c}, \text{diag}(\boldsymbol{\sigma}_{x,c}))$, with a diagonal covariance matrix, for the latent variables $\mathbf{z}$ (the subscripts in $\boldsymbol{\mu}_{x,c}$ and $\boldsymbol{\sigma}_{x,c}$ will be omitted whenever the dependency of these parameters on x and c is clear from the context). Firstly, in the encoder each categorical variable (activities and categorical attributes) is passed through its own embedding layer and transformed into a numerical vector. Then, the encoder consists of two paths: the first makes use of an LSTM layer to handle activities and timestamps, the second employs a fully connected layer for each trace attribute. Thereafter, outputs of the two paths are concatened together, the conditional variable is added, and lastly two fully connected layers are used to map to mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}$ vectors.

C. Decoder

After sampling a latent space vector $\mathbf{z}$ from the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_{x,c}, \text{diag}(\boldsymbol{\sigma}_{x,c}))$ identified by the encoder, the goal of the decoder is to generate a trace from $\mathbf{z}$ that is as similar as possible (modulo a certain level of variability) to the original trace x.

First of all, since we are in a conditional setting, the conditional variable c is concatenated to $\mathbf{z}$. Then, a fully connected layer upsamples $\mathbf{z}$ to a higher dimension vector $\mathbf{z}_U$. This approach, even though it has been criticized in previous NLP works [24], has proven effective in our case to improve decoder performances. Activities and timestamps are obtained from $\mathbf{z}_U$ using two different autoregressive LSTMs. At each time step i, the activity LSTM takes as input both $\mathbf{z}_U$ and the previously reconstructed activity $\hat{a}_{i-1}$ (for the first time step, a special End Of Trace (EOT) token is used) and outputs the current activity $\hat{a}_i$. The timestamp LSTM takes as input, in addition to $\mathbf{z}_U$ and the previous event interarrival time $\hat{t}_{i-1}$, also the current reconstructed activity $\hat{a}_i$, and outputs the current event interarrival time $\hat{t}_i$.

Different configurations has been tested, namely (i) not using the latent space as input for each time step, (ii) not conditioning the timestamp LSTM to the current activity $\hat{a}_i$ and (iii) using a shared LSTM for both activities and timestamps. However, the first configuration yielded poor reconstructed control flows, whereas the second and third configurations resulted in poor timestamp reconstruction.

Categorical and numerical attributes reconstruction follows different paths, one for each attribute, composed of a sequence of two fully connected layers and ReLU activations. For both activity and categorical attributes predictions, the model outputs a probability distribution for each possible value and the argmax operator is used to select the most likely one.

D. Training

We train the model end-to-end with backpropagation, optimizing the CVAE loss function (4). In particular, the reconstruction loss of a reconstructed trace $\hat{x} \sim p_\theta(x|\mathbf{z},c)$ with respect to its ground-truth trace x is the sum of the following loss components:[3]

- Binary Cross Entropy (BCE) loss of each trace activity
- Mean Squared Error (MSE) loss of each event interarrival time
- BCE loss of each categorical attribute of the trace
- MSE loss of each numerical attribute of the trace

To prevent vanishing of the KL divergence loss we make use of a technique called *KL cyclical annealing*, which varies β following a linear cyclical schedule as training progresses [25]. When $\beta < 1$, the model is able to focus more on improving reconstruction. In our tests without cyclical annealing, the KL divergence loss decreased to nearly zero whereas the reconstruction loss did not improve much.

E. Generation

After the model has been trained, new traces can be generated by randomly sampling a latent space vector $\mathbf{z}$ from the multivariate standard Gaussian distribution $\mathcal{N}(\vec{0}, \mathbf{I})$, attaching a conditional variable to it and feeding the resulting vector to the decoder network. The conditional variable makes it possible to limit the generation of traces to the specified variable only.

The decoder activity LSTM recurrently generates activities for the trace until the EOT token gets generated or until a fixed

[3]Using these reconstruction loss components corresponds to evaluating (i) the likelihood of every activity and categorical trace attribute against the respective categorical (softmax-normalized) distribution predicted by the decoder, and (ii) the likelihood of every timestamp and numerical trace attribute against a Gaussian distribution (representing a sort of additive noise) located at respective real-valued prediction returned by the decoder.

TABLE I
DATASETS DESCRIPTION

Dataset	Trace #	Variant #	Activity #	Avg. trace length	Avg. trace cycle time	Conditional ratio
Sepsis	782	733	16	17	38.2d	14%
Bpic2012_A	4685	3790	36	40	18.7d	17%
Bpic2012_B	4685	3790	36	40	18.7d	28%
Traffic_Fines	129615	200	10	4	20.2wks	4%

maximum trace length is reached. For each activity, the LSTM outputs a probability distribution and the argmax operator is used to choose the most likely activity.

To reconstruct timestamps, the decoder makes use of three pieces of information, namely (1) an arbitrary start timestamp τ, (2) the *trace arrival time* $\hat{T}_1$ and (3) the *event interarrival times* $\hat{t}_i$. The timestamp $\hat{T}_i$ of activity $\hat{a}_i$ is computed as follows: $\hat{T}_i = \tau + \hat{T}_1 + \sum_{k=1}^{i} \hat{t}_k$. The start timestamp τ can be chosen arbitrary, but it is usually set to the first timestamp of the entire log, or to the first or last timestamp of the test log. Also note that, given the way trace arrival times T_i are processed, the model generates traces that follow the arrival time distribution of the training set.

V. EVALUATION

In this section, we present the evaluation methodology used to assess the outcomes of conditional generative models for trace generation. We compare our method (denoted as ***cvae***) with the DL generative model from [8].[4] We aim to address the following research questions:

RQ1 Quality of Generated Traces: How is the quality of the generated traces, in terms of temporal and control-flow dimensions, compared to other state-of-the-art generative models?

RQ2 Variability vs. Compliance: What is the trade-off between the variability of the generated traces and their compliance with the original process, compared to other state-of-the-art generative models?

RQ3 Effectiveness of Conditional Control: How effective is the control provided by the conditional variable in guiding the generative model to produce specific types of traces?

RQ1 aims to investigate the overall quality of the generation, focusing on three main aspects: the control-flow, the temporal distribution of events and the distribution of trace cycle times. **RQ2** aims at assessing the capability of the model to produce original traces different from the one of the training set, while keeping them meaningful with respect to process constraints. Finally, **RQ3** aims at analysing the effectiveness of the model's conditional mechanism and its ability to correctly reproduce the various types of traces defined by the conditional variable.

A. Datasets

Our evaluation considers four examples based on three real-world event logs, preprocessed to remove incomplete traces. For each example, we define a binary conditioning on trace execution, that distinguishes between two relevant subprocesses, enabling us to generate only the desired traces by tuning the conditional variable.

- **Sepsis cases** [26] contains events of sepsis cases from a hospital. We filtered out incomplete traces with a missing "Release" activity. We consider the following conditional labelling: the patient has a relapse and returns to the emergency room within 28 days from discharge. We denote this example as ***Sepsis***.
- **BPIC2012** [27] contains a loan application process. We filtered out incomplete traces and those corresponding to ineligible applications, keeping those for which at least one offer was created. We consider two different conditional labellings for this log: (i) the reject of a loan offer by the applicant, indicated by the presence of the activity "O_DECLINED-COMPLETE", denoted as ***Bpic2012_A***; (ii) the creation of multiple loan offers by the bank, indicated by the presence of multiple "O_CREATED-COMPLETE" events, denoted as ***Bpic2012_B***.[5]
- **Traffic fines** [28] is an event log of an information system managing road traffic fines. We filtered out incomplete traces, whose last activity is "Send Fine". The conditional labelling is defined by the presence of an appeal request by the offender to the judge or the prefecture. This corresponds to the presence of any of the following activities: "Appeal to Judge", "Send Appeal to Prefecture", "Insert Data Appeal to Prefecture", "Notify Result Appeal to Offender", "Receive Result Appeal from Prefecture". This example is denoted as ***Traffic_Fines***.

In Table I, we report details on the four datasets, highlighting their variety in terms of number of traces, activity labels, cycle times, and *conditioning ratios*, i.e., the percentage of traces for which the conditional labelling is true.

B. Methodology

Each dataset is obtained by adding a trace attribute to every trace, containing the value of the conditional labelling, as defined in the previous section. Each dataset is then split in training, validation and test set in chronological order.[6] Our model is trained using the following hyperparameters: Embedding size (categorical attributes and activities) = 5, LSTM hidden size = 200, Latent space size = 10, Learning rate = 3×10^{-4}, Dropout = 5%, Batch size = 256, Number of KL annealing cycles = 8.

During training, the validation set is used to activate an early stopping mechanism, which stops the training when the loss function, computed on the validation set, does not improve for 100 epochs, and outputs the model with the best loss. This only applies when evaluating the full loss function ($\beta = 1$, i.e., KL annealing cycles are excluded by early stopping).

[4]We use the release *First version refactory* in the repository cited in [8].

[5]The BPIC2012 log also contains information about the events' lifecycle transitions: "SCHEDULE", "START", and "COMPLETE". This information is concatenated with the activity label of each event in order to reproduce it.

[6]In general, we apply a 70%-10%-20% split, with the exception of ***Traffic_Fines***, for which we take a 5% test set that already contains 6480 traces, which is in the same order of magnitude of the other test logs.

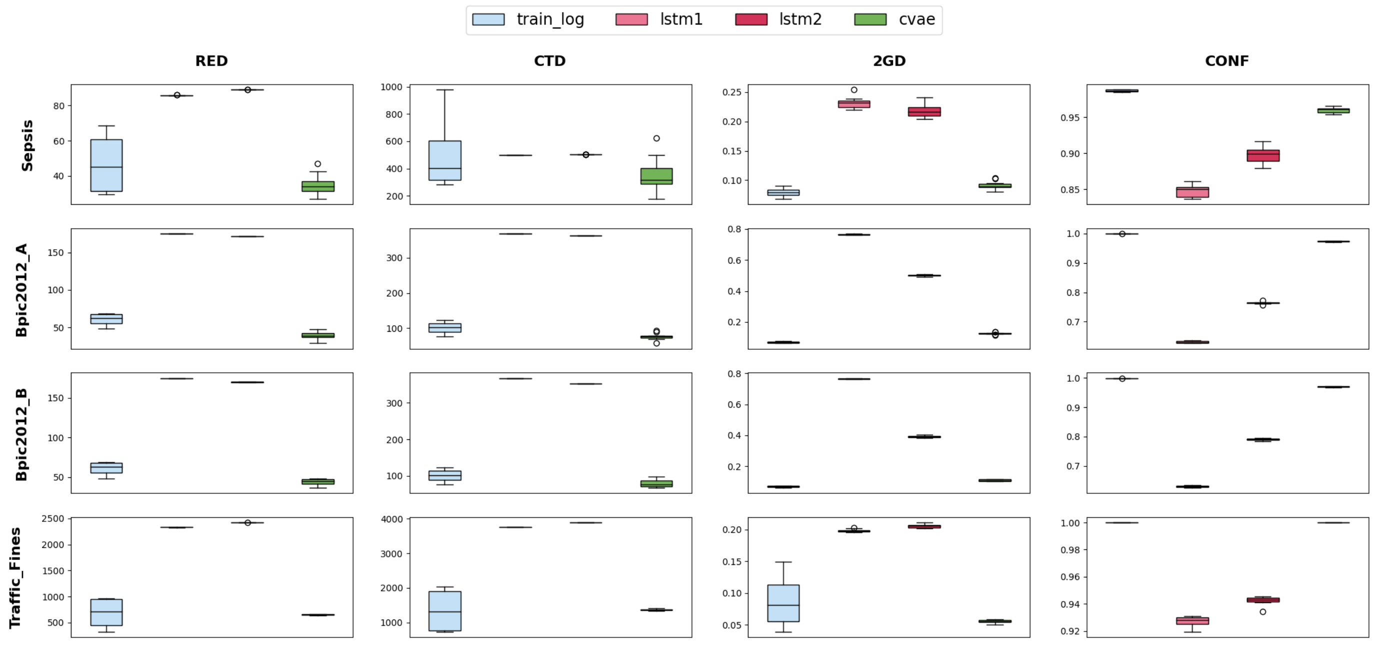

Fig. 2. Conditioning-free generation (**RQ1**): results obtained for the proposed ***cvae*** method, the competitors ***lstm1*** and ***lstm2*** and the "optimistic" baseline ***train_log*** on the metrics, RED, CTD, 2GD (the lower the better), CONF (the higher the better).

The trained model is used to generate 10 different logs, which are used for the assessment. Each generated log contains the same number of traces as the corresponding test set. The generation is guided by setting the conditional variable to reproduce the same conditional ratio found in the training log.

C. Metrics

We present the framework for the evaluation of the generative model. To assess the quality of the generated traces, we adopt a subset of the metrics introduced in [17] within the domain of simulation models. These metrics quantify the "dissimilarity" between the generated traces and those in the test set by computing the Earth Mover's Distance (EMD) of their respective distributions along different temporal and control flow dimensions:

- Event time distribution within the trace, in the case of metric *Relative Event Distribution* (RED);[7]
- Cycle times, in the case of metric *Cycle Time Distribution* (CTD);
- Event class N-grams, in the case of metric N*-Gram Distance* (NGD), which we specifically computed by setting $N = 2$ (2GD), using the directly-follow graphs of the generated and test traces.

Inspired by the work in couterfactual generation in PM [29], we consider a further metrics measuring the compliance of the generated traces with the process implicit constraints:

- *Conformance score* (CONF) measures the average number of DECLARE constraints [30] satisfied by the generated traces. The DECLARE constraints are mined from the event log with a support of 90%.[8]

To quantify the *variability* of the generated traces we focus on the control-flow and compute the number of new variants generated with respect to those in the training and test log.

Finally, to assess the conditional generation we compute the *actual conditional ratio* of the generated traces. To do so we recompute a-posteriori the conditional labelling, as defined in Sect. V-A, and compare the conditional rate with the ones of the training and test log.

D. Benchmarks

As mentioned at the beginning of this section we compare our method with the LSTM model introduced in [8], which has been identified as the state-of-the-art deep generative model for PM in [13]. Since this method is not aware of the conditional variable, we leverage it in the following two ways:

- we train it on the full training set and use it to unconditionally generate traces. We denote this method as ***lstm1***;
- we train two separate models on the two subsets of the training data identified by the value of the binary conditional variable. We then generate traces using both models to reproduce the conditional ratio of the training set. We denote this method as ***lstm2***.

To provide an additional point of comparison, we also consider the original log itself as a baseline reference for the metrics RED, CTD, and 2GD. To achieve this, we split the

[7]We adopt the RED metric instead of the Absolute Event Distribution (AED) metric from [17] because we want to assess the quality of individual generated traces, focusing on the temporal distribution of events within the trace, rather than on the overall log temporal horizon.

[8]To ensure fairness, only generated variants that do not already appear in the training log are used for this analysis.

training plus the validation set into four parts based on the chronological order. Each of these components contains the same number of traces as the test set, thus enabling a direct comparison when computing the aforementioned metrics. We denote this baseline as ***train_log***.

VI. Results

In this section, we present the obtained results.[9] We start by showcasing an example of conditional generation with ***cvae*** in Table II, where we list the control-flow of two pairs of traces generated for the experiment ***Traffic_Fines***. Each pair of traces is generated by the same value of the latent variable $\mathbf{z}$ but with different values of the conditional variable c. These two examples can be interpreted as "what-if" scenarios. When $c = \mathrm{F}$, in both cases the fine is paid by the offender within a short time. When $c = \mathrm{T}$, the offender appeals in the two traces to the Judge and to the Prefecture, respectively. This causes delays, and, as a consequence, a penalty is added six months after from the issuance of the fine, and this is known to be a process constraint [31] — when answering **RQ3** we will show that the robustness under process constraints is not an accident of this example, but is a general feature of our model. The first example ends with the *Payment* activity, indicating that the appeal has been denied. In contrast, the second example shows that the appeal to the prefecture has been accepted, and no payment is due.

We start the analysis of the results by answering **RQ1**. In Figure 2 we plot the values of the temporal (RED, CTD) and control-flow (2GD) metrics computed for each of the four experiments described in Section V-A. We can observe that ***cvae*** consistently outperforms the competing ***lstm*** models in all three metrics: relative event distribution (RED), trace cycle time (CTD) and 2-grams (2GD). It is also worth noting that the ***cvae*** boxplots often overlap with the ***train_log*** ones, indicating that it correctly reproduces the distributions of the training set for these three metrics. In summary, the quality of the traces generated by the ***cvae*** appears to be much higher than those of the ***lstm*** models on both the control-flow and temporal aspects. Finally, we observe that across different logs and metrics, the boxplots of all three generative models are significantly narrower than the ones of the baseline. Compared to a pure data sampling approach (as in the baseline), these models appear more stable in producing representative trace samples. However, all three methods likely fail to capture the full variability of the original logs.

To address **RQ2**, we report in Table III the average number of variants generated by the models that already appear in the training or test sets. We observe that, compared to the ***lstm1*** and ***lstm2*** models, which almost always generate new variants, depending on the dataset, from 15% to 77% (from 1% to 42%) of the variants generated by the ***cvae*** model already appear in the training log (test log). Looking at the CONF values reported in Figure 2, we see that ***cvae*** achieves a significantly higher conformance, ranging from 5% to more than 30% higher than the ***lstm*** models and very close to the maximal "self"-conformance value (reached by the ***train_log*** baseline). This indicates that almost all the variability generated by the ***cvae*** model complies with the process implicit constraints, while the ***lstm*** models introduce a noticeable degree of non-conformity. This difference is likely due to the different "hallucination" mechanisms used by the two architectures to generate diverse outputs. Specifically, the sampling in the latent space used by the ***cvae*** is more robust compared to the random choice selection of the LSTM network's output used by the ***lstm*** models. The latter method has a relatively small but appreciable probability of sampling an incongruous next activity during the trace generation.

Finally, we address **RQ3**. Table IV reports the original conditional ratios for the training and test logs, as well as those computed a-posteriori for generated logs. This analysis, using the definition from Sect. V-A, determines the conditional labels of the generated traces. It helps assess whether the generative models accurately reproduce the labels they were constrained by during inference and identifies any discrepancies in label reproduction. We expect that the generated logs reproduce the ratio observed in the training set. We observe that ***cvae*** consistently reproduces a conditional ratio that differs from the training ratio of at most 2%. This is not the case for none of the two ***lstm*** models. While ***lstm2*** generally performs well, it fails to reproduce the correct ratio specifically in the ***Sepsis*** log, despite being trained separately on the two conditioned subsets of the log. Notably, the ***Sepsis*** log is the only example that includes temporal conditioning. The robustness of the conditional generation of the ***cvae*** model is further confirmed by the computation of the RED, CTD, 2GD e CONF metrics separately on the two log subsets.[10]

VII. Conclusions

In this paper, we introduced a conditional variational auto encoder for trace generation. We showed that our method outperforms current state-of-the-art generative models for trace generation in terms of the quality of the traces' control-flow, cycle time and event temporal distribution, as well as of their compliance with process implicit constraints. Moreover, we showed that it is possible to robustly control the generation process by setting the conditional variable, so as to generate only the traces of interest or to simulate "what-if" scenarios. In the future, we plan to extend the log generation by also taking into account resources and to improve the reproduction of the traces temporal distribution.

References

[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.

[2] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, "Predictive business process monitoring with lstm neural networks," in *Advanced Information Systems Engineering - 29th Int. Conf., CAiSE 2017*, ser. LNCS, vol. 10253. Springer, 2017, pp. 477–492.

[9] The implementation to reproduce the experiments, the datasets, and the additional results — including the temporal distributions of each activity and trace attribute value as well as other evaluation metrics — can be accessed at the following link: https://github.com/rgraziosi-fbk/cvae-process-mining.

[10] The results of the analysis are available at the reproducibility link.

TABLE II
EXAMPLE OF GENERATIONS

c	trace
F	⟨ (Create Fine, 29/12/99) , (Payment, 9/1/00) ⟩
T	⟨ (Create Fine, 1/1/00), (Send Fine, 15/4/00), (Insert Fine Notification, 27/4/00), (Appeal to Judge, 27/5/00), (Add penalty, 29/6/00), (Payment, 23/9/01) ⟩
F	⟨ (Create Fine, 2/1/00), (Send Fine, 12/3/00), (Payment, 12/4/00) ⟩
T	⟨ (Create Fine, 1/1/00), (Send Fine, 15/4/00), (Insert Fine Notification, 29/4/00), (Insert Date Appeal to Prefecture, 6/6/00), (Add penalty, 28/6/00), (Send Appeal to Prefecture, 21/7/00) ⟩

TABLE III
VARIANT ANALYSIS

Model	Dataset	Variant # (total)	Variant # (in training)	Variant # (in test)
cvae	***Sepsis***	152.7	52.8	2
	Bpic2012_A	880.4	125.1	43.8
	Bpic2012_B	882.2	153	48.9
	Traffic_Fines	60.7	46.6	25.6
lstm1	***Sepsis***	157	0.1	0.1
	Bpic2012_A	937	0	0
	Bpic2012_B	937	0	0
	Traffic_Fines	282.7	16.1	9.8
lstm2	***Sepsis***	157	0.8	0.0
	Bpic2012_A	936.6	1	0.6
	Bpic2012_B	936.6	0.3	0.3
	Traffic_Fines	218.6	42	26.5

TABLE IV
CONDITIONAL RATIO

dataset	training	test	***cvae***	***lstm1***	***lstm2***
Sepsis	14.8%	11.5%	15%	28.6%	38.3 %
Bpic2012_A	16.5%	19.2%	15.3%	14.4%	15.1%
Bpic2012_B	28.2%	26%	27%	31.5%	27.3%
Traffic_Fines	3.2%	3.6%	4%	3%	4%

[3] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decision Support Systems*, vol. 100, pp. 129–140, 2017.

[4] L. Lin, L. Wen, and J. Wang, "Mm-pred: A deep predictive model for multi-attribute event sequence," in *Proc. of the 2019 SIAM Int. Conf. on Data Mining, SDM 2019*. SIAM, 2019, pp. 118–126.

[5] N. Mehdiyev, J. Evermann, and P. Fettke, "A multi-stage deep learning approach for business process event prediction," 07 2017, pp. 119–128.

[6] N. Tax, I. Teinemaa, and S. J. van Zelst, "An interdisciplinary comparison of sequence modeling methods for next-element prediction," *Software and Systems Modeling*, vol. 19, pp. 1345 – 1365, 2018.

[7] T. Nolle, A. Seeliger, and M. Mühlhäuser, "Binet: Multivariate business process anomaly detection using deep learning," in *Business Process Management - 16th Int. Conf., BPM 2018, Proc.*, ser. LNCS, vol. 11080. Springer, 2018, pp. 271–287.

[8] M. Camargo, M. Dumas, and O. G. Rojas, "Learning accurate LSTM models of business processes," in *Business Process Management - 17th Int. Conf., BPM 2019, Proc.*, ser. LNCS, vol. 11675. Springer, 2019, pp. 286–302.

[9] F. Taymouri, M. L. Rosa, S. M. Erfani, Z. D. Bozorgi, and I. Verenich, "Predictive business process monitoring via generative adversarial nets: The case of next event prediction," in *Business Process Management - 18th Int. Conf., BPM 2020, Proc.*, ser. LNCS, vol. 12168. Springer, 2020, pp. 237–256.

[10] P. Krajsic and B. Franczyk, "Variational autoencoder for anomaly detection in event data in online process mining," in *Proc. of the 23rd Int. Conf. on Enterprise Information Systems, ICEIS 2021, Volume 1*. SCITEPRESS, 2021, pp. 567–574.

[11] M. Camargo, M. Dumas, and O. González-Rojas, "Automated discovery of business process simulation models from event logs," *Decision Support Systems*, vol. 134, p. 113284, 2020.

[12] M. F. Sani, J. J. G. Gonzalez, S. J. van Zelst, and W. M. P. van der Aalst, "Conformance checking approximation using simulation," in *2nd Int. Conf. on Process Mining, ICPM 2020*. IEEE, 2020, pp. 105–112.

[13] M. Camargo, M. Dumas, and O. G. Rojas, "Discovering generative models from event logs: data-driven simulation vs deep learning," *PeerJ Comput. Sci.*, vol. 7, p. e577, 2021.

[14] ——, "Learning accurate business process simulation models from event logs via automated process discovery and deep learning," in *Advanced Information Systems Engineering - 34th Int. Conf., CAiSE 2022, Proc.*, ser. LNCS, vol. 13295. Springer, 2022, pp. 55–71.

[15] F. Meneghello, C. D. Francescomarino, and C. Ghidini, "Runtime integration of machine learning and simulation for business processes," in *5th Int. Conf on Process Mining, ICPM 2023*. IEEE, 2023, pp. 9–16.

[16] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," in *Advances in Neural Information Processing Systems 28 NeurIPs 2015*, 2015, pp. 3483–3491.

[17] D. Chapela-Campa, I. Benchekroun, O. Baron, M. Dumas, D. Krass, and A. Senderovich, "Can I trust my simulation model? measuring the quality of business process simulation models," in *Business Process Management - 21st Int. Conf., BPM 2023, Proc.*, ser. LNCS, vol. 14159. Springer, 2023, pp. 20–37.

[18] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[19] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[21] P. Sharma, M. Kumar, H. K. Sharma, and S. M. Biju, "Generative adversarial networks (GANs): Introduction, Taxonomy, Variants, Limitations, and Applications," *Multimedia Tools and Applications*, Mar. 2024. [Online]. Available: https://doi.org/10.1007/s11042-024-18767-y

[22] S. Tomar and A. Gupta, *A Review on Mode Collapse Reducing GANs with GAN's Algorithm and Theory*. Cham: Springer International Publishing, 2023, pp. 21–40. [Online]. Available: https://doi.org/10.1007/978-3-031-43205-7_2

[23] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[24] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, "Generating sentences from a continuous space," *arXiv preprint arXiv:1511.06349*, 2015.

[25] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin, "Cyclical annealing schedule: A simple approach to mitigating kl vanishing," *arXiv preprint arXiv:1903.10145*, 2019.

[26] F. Mannhardt, "Sepsis cases - event log," 2016.

[27] B. van Dongen, "Bpi challenge 2012," Apr 2012.

[28] M. M. de Leoni and F. Mannhardt, "Road traffic fine management process," 2015.

[29] A. Buliga, C. D. Francescomarino, C. Ghidini, and F. M. Maggi, "Counterfactuals and ways to build them: Evaluating approaches in predictive process monitoring," in *Advanced Information Systems Engineering - 35th Int. Conf., CAiSE 2023, Proc.*, ser. LNCS, vol. 13901. Springer, 2023, pp. 558–574.

[30] S. Schönig, C. Di Ciccio, F. M. Maggi, and J. Mendling, "Discovery of multi-perspective declarative process models," in *Service-Oriented Computing - 14th Int. Conf., ICSOC 2016, Proc.*, ser. LNCS, vol. 9936. Springer, 2016, pp. 87–103.

[31] F. Mannhardt, M. de Leoni, H. Reijers, and W. Aalst, "Balanced multi-perspective checking of process conformance," *Computing*, 02 2015.