EventFlow: Forecasting Temporal Point Processes with Flow Matching

Gavin Kerrigan[†]
University of Oxford
kerrigan@stats.ox.ac.uk

Kai Nelson[†]
University of California, Berkeley
kai_nelson@berkeley.edu

Padhraic Smyth
University of California, Irvine
smyth@ics.uci.edu

Abstract

Continuous-time event sequences, in which events occur at irregular intervals, are ubiquitous across a wide range of industrial and scientific domains. The contemporary modeling paradigm is to treat such data as realizations of a temporal point process, and in machine learning it is common to model temporal point processes in an autoregressive fashion using a neural network. While autoregressive models are successful in predicting the time of a single subsequent event, their performance can degrade when forecasting longer horizons due to cascading errors and myopic predictions. We propose `EventFlow`, a non-autoregressive generative model for temporal point processes. The model builds on the flow matching framework in order to directly learn joint distributions over event times, side-stepping the autoregressive process. `EventFlow` is simple to implement and achieves a 20%-53% lower forecast error than the nearest baseline on standard TPP benchmarks while simultaneously using fewer model calls at sampling time.

1 INTRODUCTION

Many stochastic processes, ranging from the occurrence of earthquakes (Ogata, 1998) to consumer behavior (Xu et al., 2014), are best understood as a sequence of discrete events that occur at random times. Any observed event sequence, consisting of one or more event times, may be viewed as a draw from a temporal point process (TPP) (Daley and Vere-Jones, 2003) which characterizes the distribution over such sequences. Given a collection of observed event sequences, faithfully modeling the underlying TPP is critical in both understanding and forecasting the phenomenon of interest.

While multiple different parametric TPP models have been proposed (Hawkes, 1971; Isham and Westcott, 1979), their limited flexibility limits their application when modeling complex real-world sequences. This has motivated the use of neural networks (Du et al., 2016; Mei and Eisner, 2017) in modeling TPPs. To date, most neural TPP models are autoregressive in nature (Shchur et al., 2020a; Zhang et al., 2020; Shchur et al., 2021), where a model is trained to predict only a single subsequent event time given an observed history of events. However, in many tasks, we are interested not only in the next event, but in the *entire sequence of events* which is to follow. While autoregressive neural TPP models can be applied in this setting, their performance in many-step forecasting tasks can be unsatisfactory due to compounding errors arising from the autoregressive sampling procedure (Xue et al., 2022; Lüdke et al., 2023).

Moreover, existing models are typically trained via a maximum likelihood procedure (see Section 3) which involves computing the CDF implied by the learned model. When using a neural model, computing this CDF often requires techniques such as Monte Carlo estimation to compute the loss (Mei and Eisner, 2017). In addition, sampling from intensity-based models (Du et al., 2016; Mei and Eisner, 2017; Yang et al., 2022) is nontrivial, requiring an expensive and difficult to implement approach based on the thinning algorithm (Lewis and Shedler, 1979; Ogata, 1981; Xue et al., 2024).

Motivated by these limitations, we propose `EventFlow`, a generative model which directly learns the full joint distribution over future event times, allowing us to avoid autoregressive sampling altogether. Our proposed model extends the flow matching framework (Lipman et al., 2023; Albergo and Vanden-Eijnden, 2023; Liu et al., 2023) to the setting of TPPs. To generate a sample, we first draw a set of random event times from a simple reference distribution and then transport them through a learned vector field to obtain a sequence of predicted times. The number of events

Proceedings of the 29[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300. [†] *Work done at the University of California, Irvine.*

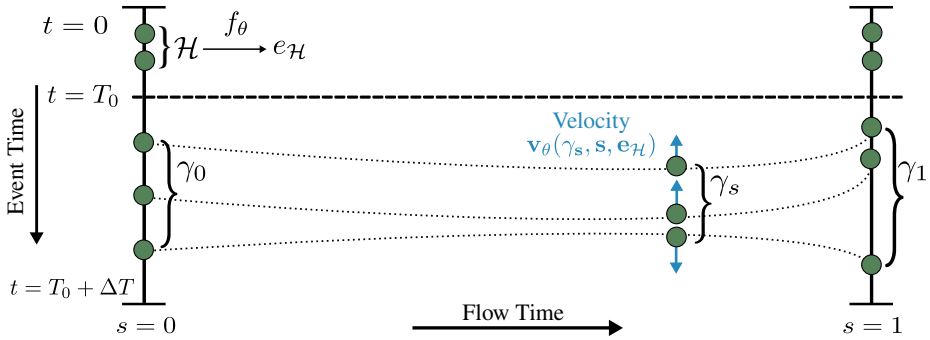

Figure 1: All illustration of forecasting with our `EventFlow` method. The horizontal axis indicates the flow time s, and the vertical axis indicates the support of the TPP $\mathcal{T} = [0, T]$. We first encode the observed history $\mathcal{H}$ into an embedding $e_\mathcal{H} = f_\theta(\mathcal{H})$. At $s = 0$, we independently draw n events in the forecasting window $[T_0, T_0 + \Delta T]$ from a fixed reference distribution, constituting a sample γ_0 from a mixed-binomial TPP. Each event can be thought of as a particle, which is assigned a velocity by a neural network $v_\theta(\gamma_s, s, e_\mathcal{H})$. Each particle flows along its velocity field until reaching its terminal point at $s = 1$, whereby we obtain a forecasted sequence γ_1.

is held fixed during this transformation, decoupling the modeling of event counts from that of event times. See Figure 1 for an illustration.

This yields a flexible formulation that can be used for both unconditional generation (sampling from the underlying TPP) and conditional generation (forecasting future events given a history). More broadly, `EventFlow` provides an alternative perspective on TPP modeling: rather than specifying an intensity function and relying on sequential simulation, we decompose the generative process into a distribution over event counts and a flow-based model over event times. This results in a non-autoregressive model that is straightforward to train and efficient to sample from.

Empirically, we demonstrate that `EventFlow` obtains state-of-the-art performance on multi-step forecasting benchmarks and is competitive with leading approaches for unconditional generation. In particular, our model reduces forecast error by 20%–53% relative to the strongest baselines. Further, compared with existing methods, which can require hundreds of model evaluations at sampling time, we demonstrate that `EventFlow` can achieve competitive performance with only a *single* forward pass of the underlying neural network.

2 RELATED WORK

Temporal Point Processes The statistical modeling of temporal point processes (TPPs) has a long history (Daley and Vere-Jones, 2003; Hawkes, 1971; Isham and Westcott, 1979). The contemporary modeling paradigm, based on neural networks (Du et al., 2016; Shchur et al., 2021), typically operates by learning a *history encoder* and an *event decoder*. The history encoder seeks to learn a fixed-dimensional vector representation of the sequence of events which have been observed or already predicted, and the decoder seeks to model a distribution over the subsequent event time(s).

Popular choices for the history encoder include RNN-based models (Du et al., 2016; Shchur et al., 2020a; Mei et al., 2019) or attention-based models (Zhang et al., 2020; Zuo et al., 2020; Yang et al., 2022). While attention-based encoders can provide longer-range contexts, this benefit typically comes at the cost of additional memory overhead.

For the event decoder, a common approach is to parametrize a conditional intensity function using a neural network that takes as input a learned representation of the event history (Du et al., 2016; Mei and Eisner, 2017; Zuo et al., 2020). An alternative and more recent approach is to use generative models as decoders. These models often do not assume a parametric form for the decoder, enhancing their flexibility. For instance, Xiao et al. (2017b) propose the use of W-GANs to generate new events. Similarly, Shchur et al. (2020a) learn a distribution over the next event time via a normalizing flow. Lin et al. (2022) benchmark several choices of generative models, including diffusion, GANs, and VAEs. Regardless of the specific choice of decoder, these approaches are all autoregressive, making them ill-suited for multi-step forecasting tasks. Notable exceptions are the works of Lüdke et al. (2023, 2026), which propose flow and diffusion based models that avoids autoregressive sampling through an iterative refinement procedure.

Our `EventFlow` model can be viewed as a flexible, non-autoregressive decoder for TPPs, obtained by extending flow matching to continuous-time event sequences. Among existing approaches, Lüdke et al. (2023, 2026) are most closely related in spirit. However,

their method relies on a relatively involved training and sampling pipeline based on iterative refinement. In contrast, `EventFlow` admits a particularly simple regression-based formulation which is straightforward to implement. While previous works have studied marked TPPs (Du et al., 2016; Mei and Eisner, 2017; Draxler et al., 2025; Chang et al., 2025), `EventFlow` is focused on modeling the event times themselves.

Flow Matching The flow matching framework (or stochastic interpolants) (Lipman et al., 2023, 2024; Albergo and Vanden-Eijnden, 2023; Liu et al., 2023) describes a class of generative models which are closely related to normalizing flows (Papamakarios et al., 2021). Flow matching has been explored for applications including image generation (Dao et al., 2023; Ma et al., 2024), DNA and protein design (Stark et al., 2024; Campbell et al., 2024), and 3D modeling (Buhmann et al., 2023; Wu et al., 2023; Xiang et al., 2025), but our work is the first to apply flow matching for TPPs.

3 AUTOREGRESSIVE MODELS

We first provide a brief, informal review of autoregressive point process models and discuss their shortcomings. Informally, an event sequence is a set $\{t^k\}_{k=1}^n$ of increasing event times. We will use $\mathcal{H}_t$ to represent the history of a sample up to (and including) time t, i.e., $\mathcal{H}_t = \{t^k : t^k \leq t\}$. Similarly, $\mathcal{H}_{t^-} = \{t^k : t^k < t\}$ represents the history prior to time t. In the autoregressive setting, the time of a single future event t is modeled conditioned on the observed history of a sequence. This is often achieved by either directly modeling a distribution over t (Shchur et al., 2020a), or equivalently by modeling a conditional intensity function (Du et al., 2016).

More precisely, one models conditional densities of the form $p(t^k \mid \mathcal{H}_{t^{k-1}})$, which describe the distribution of the next event time given the history up to the previous event. This induces a joint distribution over event times $p(t^1, \dots, t^n)$ autoregressively via $p(t^1, \dots, t^n) = p(t^1) \prod_{k=2}^n p(t^k \mid \mathcal{H}_{t^{k-1}})$. Alternatively, we may define the *conditional intensity* $\lambda^*(t) := \lambda(t \mid \mathcal{H}_{t^-}) = p(t \mid \mathcal{H}_{t^n})/(1 - F(t \mid \mathcal{H}_{t^n}))$, where $F(t \mid \mathcal{H}_{t^n}) = \int_{t^n}^t p(s \mid \mathcal{H}_{t^n}) \, \mathrm{d}s$ is the CDF associated with $p(t \mid \mathcal{H}_{t^n})$. Informally, the conditional intensity $\lambda^*(t)$ can be thought of (Rasmussen, 2011) as the instantaneous rate of occurrence of events at time t given the history up to t^n and that no events have occurred in (t^n, t). By integrating $\lambda^*(t)$, one can verify

$$F(t \mid \mathcal{H}_{t^n}) = 1 - \exp\left(-\int_{t_n}^t \lambda^*(s) \, \mathrm{d}s\right) \tag{1}$$

$$p(t \mid \mathcal{H}_{t^n}) = \lambda^*(t) \exp\left(-\int_{t_n}^t \lambda^*(s) \, \mathrm{d}s\right) \tag{2}$$

and thus one may recover the conditional distribution from the conditional intensity under mild additional assumptions (Rasmussen, 2011, Prop 2.2).

The Likelihood Function Suppose we observe an event sequence $\{t^k\}_{k=1}^n$ on the interval $[0, T]$. The *likelihood* of this sequence can be seen loosely as the probability of observing events at these times and no others within the observation window. The likelihood may be expressed in terms of either the density or intensity via

$$L\left(\{t^k\}_{k=1}^n\right) = p(t^1, \dots, t^n)\left(1 - F(T \mid \mathcal{H}_{t^n})\right) \tag{3}$$

$$= \left(\prod_{k=1}^n \lambda^*(t^k)\right) \exp\left(-\Lambda^*(T)\right) \tag{4}$$

where $\Lambda^*(T) = \int_0^T \lambda^*(s) \, \mathrm{d}s$ is the total intensity. The survival term $1 - F(T \mid \mathcal{H}_{t^n})$ accounts for the absence of events in $(t^n, T]$. Autoregressive models are typically trained by maximizing this likelihood (Du et al., 2016; Mei and Eisner, 2017; Shchur et al., 2020a). Critically, we emphasize that $p(t^1, \dots, t^n)$ captures a joint density over n event times, but does not account for what occurs after the final event t^n. In contrast, the likelihood $L\left(\{t^k\}_{k=1}^n\right)$ corresponds to observations in a finite window $[0, T]$, and therefore includes an additional CDF term to account for the fact that no events were observed in $(t^n, T]$.

It is worth noting that evaluating $L(\{t_k\})$ can be nontrivial in practice. For models that parametrize $\lambda^*(t)$ via a neural network (Du et al., 2016; Mei and Eisner, 2017), computing the total intensity $\Lambda^*(T)$ is often done via a Monte Carlo integral, requiring many forward passes of the model to evaluate $\lambda^*(t)$ at different values of t. Models which directly parametrize the density $p(t \mid \mathcal{H}_t)$ suffer from the same drawback when computing the corresponding CDF in Equation (3). Moreover, some approaches, such as the diffusion-based approach of Lin et al. (2022), are only trained to maximize an ELBO of $p(t_k \mid t_1, \dots, t_{k-1})$, and are thus unable to compute the proper likelihood in Equation (3).

Sampling Autoregressive Models In many tasks, we are interested not only in an accurate model of the intensity (or distribution), but also sampling new event sequences. For instance, when forecasting an event sequence, we may want to generate several forecasts in order to provide uncertainty quantification over these predictions. However, sampling from existing models can be difficult. The flow-based model of Shchur et al. (2020a) requires a numerical approximation to the inverse of the model to perform sampling.

The diffusion-based approach of Lin et al. (2022) can require several hundred forward passes of the model to generate a *single* event time, rendering it costly when generating long sequences. Moreover, the predictive performance of autoregressive models is often unsatisfactory on multi-step generation tasks due to the accumulation of errors over many steps (Lin et al., 2021; Lüdke et al., 2023). This difficulty is particularly pronounced for intensity-based models (Du et al., 2016; Mei and Eisner, 2017; Zhang et al., 2020), where naively computing the implied distribution in Equation (2) is prohibitively expensive. Instead, sampling from intensity-based models is typically achieved via the thinning algorithm (Ogata, 1981; Lewis and Shedler, 1979). However, this algorithm has several hyperparameters to tune, is challenging to parallelize, and can be difficult for practitioners to implement (Xue et al., 2024). For instance, the thinning algorithm requires one to know an upper bound on the intensity, which is in practice selected heuristically.

4 EVENTFLOW

Motivated by the limitations of autoregressive models, we propose `EventFlow`, which has a number of distinct advantages over prior work. First, `EventFlow` directly models the joint distribution over event times, thereby avoiding autoregression entirely. Second, our model is likelihood-free at training time, avoiding the Monte Carlo estimates needed to estimate the likelihood in Equation (3). Third, sampling from our model amounts to solving an ordinary differential equation. This is straightforward to implement and parallelize, allowing us to avoid the difficulties of thinning-based approaches. In fact, `EventFlow` can achieve competitive performance using only a *single* forward pass at sampling time.

We build upon the flow matching (or stochastic interpolant) framework (Lipman et al., 2023; Albergo and Vanden-Eijnden, 2023; Liu et al., 2023) to develop our model. We begin below by focusing on the unconditional setting (i.e., generating an event sequence without being given an observed history), followed by an extension for conditional generation.

Preliminaries We first introduce some necessary background and notation. Let $\mathcal{T} = [0, T]$ be a finite length interval. The set Γ denotes the *configuration space* of $\mathcal{T}$ (Albeverio et al., 1998), i.e., the set of all finite counting measures on the set $[0, T]$. A point $\gamma \in \Gamma$ corresponds to a measure of the form $\gamma = \sum_{k=1}^{n} \delta[t^k]$, i.e., a finite collection of Dirac deltas located at event times $t^k \in \mathcal{T}$. A *temporal point process (TPP)* on $\mathcal{T}$ is a probability distribution μ over the configuration space Γ. We use $\mathbb{P}(\Gamma)$ to denote the set of all such distributions. Informally, a TPP μ is a distribution over finite sequences $\gamma \in \Gamma$ whose events are in $\mathcal{T}$. We use $N : \Gamma \to \mathbb{Z}_{\geq 0}$ to denote the counting functional, i.e., $N(\gamma)$ is the number of events in the sequence γ.[1] While it is common to represent TPPs as distributions over random sets of event times, in our approach it will be more convenient to represent TPPs as random measures (Kallenberg, 2017).

Under mild assumptions, a TPP μ can be fully characterized (Daley and Vere-Jones, 2003, Prop. 5.3.II) by a probability distribution which specifies the number of events and a *collection* of joint densities corresponding to the event times themselves. In a slight abuse of notation, we will write $\mu(n)$ for the corresponding distribution over event counts, and $\{\mu^n(t^1, \ldots, t^n)\}_{n=1}^{\infty}$ for the collection of joint distributions. In other words, for any given $n \in \mathbb{Z}_{\geq 0}$, the probability of observing n events in the interval $\mathcal{T}$ is $\mu(n)$, and $\mu^n(t^1, \ldots, t^n)$ describes the corresponding joint distribution of event times. We further restrict each μ^n to be supported only on the ordered sets, so that $t^1 < t^2 < \cdots < t^n$.

Let μ_1 represent the data distribution and μ_0 represent a reference distribution, i.e., both $\mu_0, \mu_1 \in \mathbb{P}(\Gamma)$ are distributions over event sequences. To construct our model, we will define a *path* of distributions $(\eta_s)_{s\in[0.1]} \subset \mathbb{P}(\Gamma)$ which approximately interpolates from our reference TPP to our data TPP. Intuitively, this path provides us with a way to *transform* samples: we can draw a sequence of events from μ_0 and gradually move them along this path to obtain a sequence approximately distributed according to μ_1. Throughout, we use $s \in [0, 1]$ to denote a flow time and $t \in [0, T]$ to denote an event time. These two time axes are in a sense orthogonal to one another (see Figure 1).

Balanced Couplings Towards constructing such a path, our first step is to define a notion of *coupling*, which allows us to pair event sequences drawn from the reference distribution μ_0 with those drawn from the data distribution μ_1. Formally, a *coupling* between two TPPs $\mu_0, \mu_1 \in \mathbb{P}(\Gamma)$ is a joint probability distribution $\rho \in \mathbb{P}(\Gamma \times \Gamma)$ over pairs of event sequences (γ_0, γ_1) such that the marginal distributions of ρ are μ_0 and μ_1. Sampling $(\gamma_0, \gamma_1) \sim \rho$ from a coupling gives us paired sequences of events.

While couplings are broadly used in transport-based generative modeling (Tong et al., 2024; Villani et al., 2009), we introduce a new class of couplings which is particularly well-suited to the TPP setting. We say that the coupling ρ is *balanced* if draws $(\gamma_0, \gamma_1) \sim \rho$

[1] This can be thought of in terms of the counting process, i.e., $N(\gamma)$ corresponds to the value of the associated counting process at the ending time T, or the total number of events in γ that have occurred in the interval $[0, T]$.

are such that $N(\gamma_0) = N(\gamma_1)$ almost surely. That is, balanced couplings only pair event sequences with equal numbers of events. This ensures that when we later construct interpolations between paired sequences, we do not need to add or remove events, simplifying both training and sampling. We will use $\Pi_b(\mu_0, \mu_1)$ to denote the set of balanced couplings.

The following proposition characterizes when balanced couplings exist. In particular, $\Pi_b(\mu_0, \mu_1)$ is nonempty if and only if the event count distributions of μ_0 and μ_1 are identical, imposing a structural constraint on suitable choices of μ_0: the reference TPP μ_0 must have the same event count distribution as the data TPP μ_1.

Proposition 1 (Existence of Balanced Couplings)**.** *Let $\mu_0, \mu_1 \in \mathbb{P}(\Gamma)$ be two TPPs. The set of balanced couplings $\Pi_b(\mu_0, \mu_1)$ is nonempty if and only if have the same distribution over event counts, i.e., $\mu_0(n) = \mu_1(n)$ for every $n \in \mathbb{Z}_{\geq 0}$.*

We provide a proof in Appendix B.

In practice, we follow a simple strategy for choosing both the reference TPP μ_0 and the coupling ρ. Let q be a density on $\mathcal{T}$ (e.g., uniform). We take μ_0 to be a mixed binomial process (Kallenberg, 2017, Ch. 3) whose event count distribution is given by that of the data $\mu_1(n)$, and joint event time distributions given by independent products of q (up to sorting). Sampling $\gamma_0 \sim \mu_0$ amounts to sampling $n \sim \mu_1(n)$ from the data event count distribution followed by sampling and sorting n i.i.d. times $t^k \sim q$. Under this choice, a sample from a balanced coupling $\rho \in \Pi_b(\mu_0, \mu_1)$ can be produced by first sampling a data sequence $\gamma_1 \sim \mu_1$, followed by sampling $N(\gamma_1)$ events independently from q and sorting to produce a paired draw $\gamma_0 \sim \mu_0$.

Interpolant Construction We now construct our path $(\eta_s)_{s\in[0,1]} \subset \mathbb{P}(\Gamma)$ using a two-stage procedure. First, for a given pair of sequences (γ_0, γ_1) drawn from a balanced coupling, we define a sequence-level interpolation $(\gamma_s)_{s\in[0,1]}$ that smoothly transforms γ_0 into γ_1. Second, the distribution η_s is obtained by averaging these interpolated sequences over the coupling. We extend flow matching techniques (Lipman et al., 2023; Tong et al., 2024) to define the interpolants, and emphasize that fixing the number of events via a balanced coupling is essential for this construction.

To that end, let ρ be any balanced coupling of the reference distribution μ_0 and the data distribution μ_1, and suppose $z := (\gamma_0, \gamma_1) \sim \rho$ is a draw from this coupling. As ρ is balanced, we have $\gamma_0 = \sum_{k=1}^n \delta[t_0^k]$ and $\gamma_1 = \sum_{k=1}^n \delta[t_1^k]$ are both a collection of n events. We will henceforth describe our procedure for a fixed (but arbitrary) number of events n, and we will later describe how to model the number of events itself. First, we define the interpolant sequence $\gamma_s^z \in \Gamma$ via

$$\gamma_s^z = \sum_{k=1}^n \delta[t_s^k] \qquad t_s^k = (1-s)t_0^k + st_1^k \qquad 0 \leq s \leq 1 \tag{5}$$

where we use the superscript z to denote the dependence on the pair $z = (\gamma_0, \gamma_1)$. In other words, γ_s^z linearly interpolates each corresponding event in γ_0 and γ_1, defining a path $(\gamma_s^z)_{s\in[0,1]} \subset \Gamma$ which evolves the reference sample γ_0 into the data sample γ_1.

We now lift this deterministic path $(\gamma_s^z)_{s\in[0,1]} \subset \Gamma$ to a path of TPP distributions $(\eta_s^z)_{s\in[0,1]} \subset \mathbb{P}(\Gamma)$. We define the point process distribution $\eta_s^z \in \mathbb{P}(\Gamma)$ implicitly by adding independent Gaussian noise to each event in γ_s^z. That is, a draw $\hat{\gamma}_s^z \sim \eta_s^z$ may be simulated via

$$\hat{\gamma}_s^z = \sum_{k=1}^n \delta\left[t_s^k + \epsilon^k\right] \qquad \epsilon^k \sim \mathcal{N}(0, \sigma^2). \tag{6}$$

In principle, using Gaussian noise means that the support of event times in η_s^z is larger than $\mathcal{T}$, but in practice we choose σ^2 sufficiently small such that this is not a concern. The addition of noise ϵ^k is instrumental in obtaining a well-specified model, but in practice we found the noise variance σ^2 to not be a critical hyperparameter. We note that this noising step is typical in flow matching models (Lipman et al., 2023; Tong et al., 2024).

Finally, for $s \in [0, 1]$, we define the marginal TPP measure $\eta_s \in \mathbb{P}(\Gamma)$ via the mixture distribution

$$\eta_s = \int \eta_s^z \, \mathrm{d}\rho(z) \tag{7}$$

which, intuitively, corresponds to the two-stage procedure of drawing a sample $z \sim \rho$ and subsequently following the conditional flow defined in Equation (6).

By construction, the event count distribution $\eta_s(n)$ is given by $\mu_1(n)$ for all $s \in [0, 1]$. This path of TPP distributions η_s approximately interpolates from the reference TPP μ_0 at $s = 0$ to the data TPP μ_1 at $s = 1$, in the sense that at the endpoints, the joint event time distributions $\eta_0^n(t^1, \ldots, t^n)$ and $\eta_1^n(t^1, \ldots, t^n)$ are given by a convolution of $\mu_0^n(t^1, \ldots, t^n)$ and $\mu_1^n(t^1, \ldots, t^n)$ with the Gaussian $\mathcal{N}(0, \sigma^2 I_n)$. As $\sigma^2 \downarrow 0$, it is clear that we recover a genuine interpolant.

Observe that in Equation (6), for each individual event t_s^k, we have constructed a path of Gaussian distributions $\mathcal{N}(t_s^k, \sigma^2)$ centered around said event. This path of Gaussians can be simulated via the vector field $v_s^k := t_1^k - t_0^k$ (Tong et al., 2024). That is, if we draw an initial condition $\tau_0^k \sim \mathcal{N}(t_0^k, \sigma^2)$ and solve the differential equation $\mathrm{d}\tau_s^k = v_s^k \, \mathrm{d}s$ for $s \in [0, 1]$ starting from τ_0^k, we obtain a draw $\tau_1^k \sim \mathcal{N}(t_1^k, \sigma^2)$. Loosely, v_s^k is a velocity associated with the event t_s^k.

Algorithm 1: Training `EventFlow`

Sample $\gamma_1 \sim \mu_1$, $s \sim \mathcal{U}[0,1]$, $\epsilon \sim \mathcal{N}(0,1)$
$e_{\mathcal{H}} = \varnothing$ `/* Null history */`
if *forecast* **then**
  Sample $T_0 \in [\Delta T, T - \Delta T]$
  $\mathcal{H} \leftarrow \{t \in \gamma_1 : t \le T_0\}$
  $e_{\mathcal{H}} \leftarrow f_\theta(\mathcal{H})$
  $\gamma_1 \leftarrow \{t \in \gamma_1 : T_0 < t \le T_0 + \Delta T\}$
$n \leftarrow N(\gamma_1)$
Sample and sort $t_0^k \sim q$ for $k = 1, \dots, n$
$\gamma_0 \leftarrow (t_0^1, \dots, t_0^n)$
$t_s^k \leftarrow (1-s)t_0^k + st_1^k$ for $k = 1, \dots, n$
$\gamma_s^z \leftarrow (t_s^1, \dots, t_s^n)$
Take a gradient step on
$\|\gamma_1 - \gamma_0 - v_\theta(\gamma_s^z + \epsilon, s, e_{\mathcal{H}})\|^2$

Since each event in Equation (6) evolves independently, the path of distributions η_s^z is generated by the constant vector field $v_s^z : \mathcal{T}^n \to \mathbb{R}^n$ defined as

$$v_s^z(\gamma) := \left[v_s^1, \dots, v_s^n\right]^\mathsf{T} = \left[t_1^1 - t_0^1, \quad \dots, \quad t_1^n - t_0^n\right]^\mathsf{T}. \tag{8}$$

In other words, flowing initial samples $\hat{\gamma}_0^z \sim \eta_0^z$ along the vector field $v_s^z(\gamma)$ generates the path of distributions η_s^z, approximately interpolating from the noise sequence γ_0 to the data sequence γ_1.

However, this path is conditioned on $z = (\gamma_0, \gamma_1)$. Since we seek to generate γ_1, this is intractable, and we would instead like to find the vector field v_s that generates the *unconditional* path η_s defined in Equation (6). To this end, we aggregate the conditional vector fields $v_s^z(\gamma)$ across the coupling $z \sim \rho$. In particular, the unconditional vector field $v_s : \mathcal{T}^n \to \mathbb{R}^n$ is given by

$$v_s(\gamma) = \mathbb{E}_{z\sim\rho}[v_s^z(\gamma) \mid \gamma_s^z = \gamma] = \int v_s^z(\gamma) \frac{\mathrm{d}\eta_s^z}{\mathrm{d}\eta_s}(\gamma)\, \mathrm{d}\rho(z) \tag{9}$$

i.e., the average of the conditional vector fields over all pairs z that could have produced the sequence γ at time s. This corresponds to an extension of the standard flow matching construction (Lipman et al., 2023; Tong et al., 2024; Albergo and Vanden-Eijnden, 2023) to the setting of TPPs.

Training If we knew v_s in Equation (9), we could draw a sample from $\gamma_1 \sim \mu_1$ by drawing a sample event sequence $\gamma_0 \sim \mu_0$ from the reference TPP and flowing each event along v_s. More precisely, γ_0 will consist of $N(\gamma_0)$ events and $v_s(\gamma)$ will be a vector field with $N(\gamma_0)$ components, so that we may solve the differential equation $\mathrm{d}\gamma_s = v_s(\gamma)\,\mathrm{d}s$ on $s \in [0,1]$.

Although the marginal vector field in Equation (9) admits an analytical form, it is intractable to compute in practice as the density ratio $\mathrm{d}\eta_s^z / \mathrm{d}\eta_s$ is unknown. To overcome this, we instead regress on the *conditional* vector fields v_s^z. Here, $v_\theta(\gamma_s, s)$ will represent a neural network with parameters θ which takes in a sequence γ_s along with the flow time s. We aim to minimize

$$J(\theta) = \mathbb{E}_{s,(\gamma_0,\gamma_1),\hat{\gamma}_s^z}\left[\|\gamma_1 - \gamma_0 - v_\theta(\hat{\gamma}_s^z, s)\|^2\right] \tag{10}$$

which is equal to the MSE loss on the *unconditional* v_s up to an additive constant not depending on θ (Lipman et al., 2023; Tong et al., 2024). We estimate $J(\theta)$ by uniformly sampling a flow time $s \in [0,1]$, a pair $z = (\gamma_0, \gamma_1) \sim \rho$ from our balanced coupling and drawing a noisy interpolant $\hat{\gamma}_s^z \sim \eta_s^z$ according to Equation (6).

To train the model for forecasting, where the goal is to predict a future sequence of events conditioned on a history $\mathcal{H}$, we embed $\mathcal{H}$ into a fixed-dimensional vector representation $e_{\mathcal{H}} = f_\theta(\mathcal{H})$ via a learned encoder f_θ before providing this to the model $v_\theta(\gamma_s, s, e_{\mathcal{H}})$ and minimizing Equation (10). Note that we jointly train the encoder f_θ and vector field v_θ. See Algorithm 1 for training pseudocode.

Event Count Distributions We have thus far described a procedure for interpolating between a given reference distribution μ_0^n and the data distribution μ_1^n for a given, fixed number of events n. As n was arbitrary, we have successfully constructed a family of interpolants which will enable us to sample from the joint event time distributions $\mu_1^n(t^1, \dots, t^n)$. However, recall that fully characterizing the TPP distribution requires us to also specify the event *count* distribution.

For unconditional generation tasks, this is straightforward: we simply follow the empirical event count distribution see in the training data. When forecasting, though, we must also learn a model of the event count distribution $p_\phi(n \mid \mathcal{H})$, i.e., how many events are likely to occur after observing a history $\mathcal{H}$. Here, ϕ represents the parameters of this model. We train $p_\phi(n \mid \mathcal{H})$ by minimizing a cross-entropy loss. In practice, we found it important to regularize this loss to encourage smoothness in n, which we achieve through an optimal-transport based regularizer. See Appendix C for details.

As both v_θ and p_ϕ must be able to operate on variable-length sequences, we use a transformer-based backbone for both models. However, our overall method is agnostic to this choice.

Sampling Once v_θ is learned, we may sample from the model by drawing a reference sequence $\gamma_0 \sim \mu_0$ and solving the corresponding ODE parametrized by v_θ. We fix a number of events n, which can be sampled from the empirical event count distribution $\mu_1(n)$ for

Algorithm 2: Sampling `EventFlow`

Choose a flow time discretization $0 = s_0 < s_1 < \cdots < s_K = 1$
$e_{\mathcal{H}} = \varnothing$ `/* Null history */`
if *forecast* **then**
  $e_{\mathcal{H}} \leftarrow f_\theta(\mathcal{H})$
  $n \sim p_\phi(n \mid \mathcal{H})$
else
  $n \sim \mu_1(n)$
Sample and sort $t_0^k \sim q$ for $k = 1, \dots, n$
$\gamma_0 \leftarrow (t_0^1, \dots, t_0^n)$
for $k = 1, 2, \dots, K$ **do**
  $h_k \leftarrow s_k - s_{k-1}$
  $\gamma_{s_k} \leftarrow \gamma_{s_{k-1}} + h_k v_\theta(\gamma_{s_{k-1}}, s_{k-1}, e_{\mathcal{H}})$
return γ_0

unconditional generation. For conditional tasks, we draw $n \sim p_\phi(n \mid \mathcal{H})$ from the learned conditional distribution over event counts. Next, we draw n initial events, corresponding to $s = 0$, by sampling and sorting $t_0^1, \dots, t_0^n \sim q$. In practice, we use $q = \mathcal{N}(0, I_n)$ as we normalize our sequences into the range $[-1, 1]$ during training and sampling (followed by renormalization to the data scale). Since we have fixed n, we may view this initial draw as a vector $\gamma_0 = [t_0^1, \dots, t_0^n] \in \mathcal{T}^n$. This event sequence γ_0 then serves as the initial condition for the system of ODEs $d\gamma_s = v_\theta(\gamma_s, s)\, \mathrm{d}s$ which can be solved numerically. In our experiments, we use the forward Euler scheme, i.e., we specify a discretization $\{0 = s_0 < s_1 < \cdots < s_K = 1\}$ of the flow time (in practice, uniform) and recursively compute

$$\gamma_{s_k} = \gamma_{s_{k-1}} + h_k v_\theta(\gamma_{s_{k-1}}, s_{k-1}), \quad k = 1, \dots, K \tag{11}$$

where $h_k = s_k - s_{k-1}$ is a step size. While other choices are possible, we found that this simple scheme was sufficient as the model sample paths are, qualitatively, typically close to linear. See Algorithm 2.

5 EXPERIMENTS

We study our proposed `EventFlow` model under two settings[2]. The first is a conditional task, where we seek to forecast both the number of future events and their times of occurrence, over a particular horizon, given a history. The second is an unconditional task, where we aim to learn a representation of the underlying TPP distribution from empirical observations and generate new sequences from this distribution. This second task can be viewed as a special case of the first with no observed history.

[2]Code for our experiments is available at this URL: https://github.com/GavinKerrigan/eventflow

We evaluate our model across a diverse set of datasets encompassing a wide range of possible point process behaviors. We evaluate our model on seven real-world datasets, which are a standard benchmark for modeling unmarked TPPs (Shchur et al., 2020b; Bosser and Taieb, 2023; Lüdke et al., 2023). We also evaluate on a collection of six synthetic datasets (Omi et al., 2019). See Appendix A for additional information regarding our datasets.

Baseline Models We selected a set of baselines consisting of a set of diverse and highly performant neural TPP models. For an intensity-based method, we compare against the Neural Hawkes Process (NHP) (Mei and Eisner, 2017), a well-known and very widely-used neural TPP model. We additionally compare against two intensity-free methods, namely the flow-based IFTPP model (Shchur et al., 2020a) and the diffusion-based model of Lin et al. (2022). Lastly, our strongest baseline is the Add-and-Thin model of Lüdke et al. (2023), which can be loosely viewed as a non-autoregressive diffusion model. These models use an RNN-based history encoder, with the exception of Add-and-Thin which uses a CNN-based encoder. See Appendix D for additional details on our baselines.

Metrics Evaluating generative TPP models is challenging, as one must take into account both the variable locations and numbers of events. This is particularly challenging for the unconditional setting, where unlike forecasting, we do not have a ground-truth sequence.

Our starting point is a metric (Xiao et al., 2017a) on the space of sequences Γ, allowing us to measure the distance between two sequences $\gamma = \sum_{k=1}^n \delta[t_\gamma^k]$ and $\eta = \sum_{k=1}^m \delta[t_\eta^k]$ with possibly different numbers of events. Without loss of generality, we assume $n \leq m$, in which case the distance is given by

$$d(\gamma, \eta) = \sum_{k=1}^{n} |t_\gamma^k - t_\eta^k| + \sum_{k=n+1}^{m} (T - t_\eta^k) \tag{12}$$

where we recall that sequences are supported on $\mathcal{T} = [0, T]$. This distance can be understood either as an L^1 distance between the counting processes of γ, η or as a generalization of the 1-Wasserstein distance to measures of unequal mass. For our unconditional experiment, we require a metric that will capture the distance between the TPP distributions themselves. To do so we use the distance in Equation (12) to calculate an MMD (Gretton et al., 2012; Shchur et al., 2020b). We use an exponential kernel $k(\gamma, \eta) = \exp\left(-d(\gamma, \eta)/(2\sigma^2)\right)$ with σ chosen to be the median distance between all sequences in a dataset (Shchur et al., 2020b; Lüdke et al., 2023).

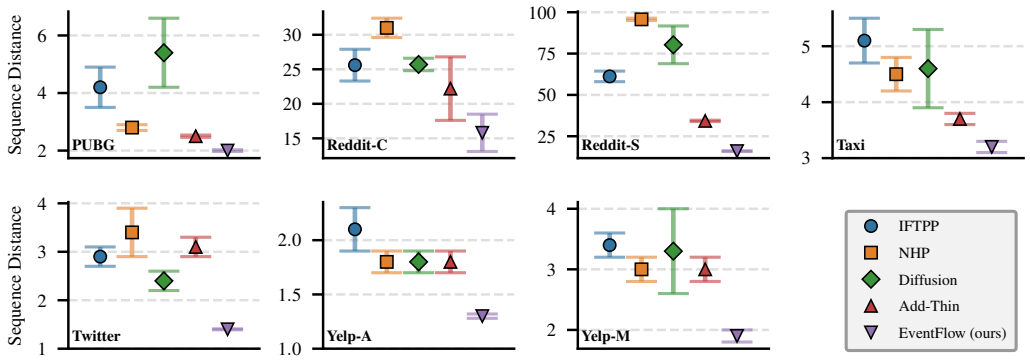

Figure 2: Sequence distance (12) between the forecasted and ground-truth event sequences on a held-out test set. We report the mean ± one standard deviation over five random seeds. `EventFlow` (with 25 NFEs) achieves the lowest mean distance (forecasting error) for each of the 7 datasets.

5.1 Forecasting Event Sequences

We first evaluate our model on a multi-step forecasting task. We set a horizon ΔT for each of our real-world datasets and seek to generate event sequences in the range $[T_0, T_0 + \Delta T]$ for some given T_0, conditioned on the history $\mathcal{H}_{T_0}$. Up to a shift, this means we are taking $\mathcal{T} = [0, \Delta T]$. At training time, we uniformly sample $T_0 \in [\Delta T, T - \Delta T]$. At test time, we sample 50 values of T_0 for each test set sequence. We then generate one forecast for the sequence in $[T_0, T_0 + \Delta T]$ and compute the distance (12) between the ground-truth and generated sequences. We note that the distance in Equation (12) is computed using $T_0 + \Delta T$ rather than T as the maximum event time, as using T would result in a distance which is sensitive to the location of the forecasting window. We further normalize Equation (12) by ΔT. This overall approach is similar to that of Lüdke et al. (2023). For all datasets, the sequences are split into train-validation-test sets, and all metrics are reported on the held-out test set.

We report the results of these experiments in Figure 2. Our proposed `EventFlow` method obtains the lowest average forecasting error across all datasets, achieving an error which is 20%-53% lower than the nearest baseline. Given that the non-autoregressive models (`EventFlow`, Add-and-Thin) typically outperform the autoregressive baselines, we see clear evidence that autoregressive models can struggle on multi-step tasks. This is especially true on Reddit-C and Reddit-S which exhibit long sequence lengths.

In Table 1, we additionally vary the number of function evaluations (NFEs) used at test time, i.e., the number of steps K used in the ODE solver in Algorithm 2. We find that reducing K results in minimal (or no) loss in performance. Even with only a single NFE, `EventFlow` is able to obtain state-of-the-art performance. This is due to the fact that, although the vector fields (Equation (9)) in `EventFlow` are generally non-linear, we find qualitatively that our construction produces paths which are approximately so. In contrast, Add-and-Thin uses 100 NFEs, while the NFEs for the autoregressive models scale linearly with the number of generated events.

Table 1: Mean sequence distance for `EventFlow` as we vary the NFEs used to simulate the ODE.

NFEs	PUBG	Red.C	Red.S	Taxi	Tw.	Yelp-A	Yelp-M
25	2.0	15.8	16.0	3.2	1.4	1.3	1.9
10	2.0	15.8	15.8	3.1	1.4	1.3	1.9
1	2.0	15.8	15.8	3.7	1.4	1.8	1.9

Table 2: MARE values evaluating the predicted number of events when forecasting. Mean values are reported over five random seeds.

	PUBG	Red.C	Red.S	Taxi	Tw.	Yelp-A	Yelp-M
IFTPP	1.05	1.69	0.79	0.60	0.88	0.76	0.76
NHP	1.02	0.95	1.00	0.67	2.48	0.80	1.07
Diffusion	1.95	1.28	1.12	0.49	0.66	0.65	0.72
Add-Thin	0.43	0.99	0.38	0.33	0.60	**0.42**	**0.46**
EventFlow	**0.40**	**0.70**	**0.16**	**0.28**	**0.46**	0.56	0.50

To evaluate the event count predictions, $p_\phi(n \mid \mathcal{H})$, we report the mean absolute relative error (MARE) between the true and predicted counts in Table 2. Our model achieves strong performance on this metric as it decouples the event count prediction from the event time prediction, training explicitly for each task. The baselines, on the other hand, only model n implicitly. See Appendix E for details and standard deviations.

Table 3: MMDs (1e-2) between the test set and 1,000 generated unconditional sequences averaged over five random seeds. `EventFlow` achieves the overall highest mean rank.

	H1	H2	NSP	NSR	SC	SR	PUBG	Red.C	Red.S	Taxi	Twitter	YelpA	YelpM
IFTPP	**1.5**	**1.4**	**2.3**	6.2	**5.8**	**1.3**	5.7	1.3	1.9	5.8	**2.9**	8.2	5.1
NHP	1.9	5.2	3.6	12.6	25.4	5.0	7.2	2.2	22.5	5.0	7.3	6.7	6.1
Diffusion	4.8	5.5	10.8	15.0	9.1	5.1	14.3	3.9	6.2	11.7	12.5	10.9	10.5
Add-Thin	1.9	2.5	2.6	7.4	22.5	2.2	2.8	1.2	2.7	5.2	4.8	**4.5**	**3.0**
EventFlow (25 NFEs)	1.9	2.2	3.8	**4.2**	8.3	1.7	**1.5**	**0.7**	**0.7**	**3.5**	4.9	6.6	**3.0**

5.2 Unconditional Generation of Event Sequences

Finally, we evaluate our model on an unconditional generation task, where we aim to generate new sequences from the underlying data distribution without conditioning on an observed history. In Table 3 we report MMD values for each of the synthetic and real-world datasets. MMDs are calculated by sampling 1,000 sequences from each trained model. The MMD values in Table 3 correspond to the mean test set MMD ($\pm$ one standard deviation) across five random seeds. See Appendix E for results with standard deviations.

`EventFlow` (mean rank: **1.8**) exhibits uniformly strong performance, obtaining either the best or second best MMD on 11 of the 13 datasets. This is particularly pronounced on the real-world datasets, where we obtain the lowest MMD on 5 of the 7 datasets.

Notably, IFTPP (mean rank: **2.1**) is one of the strongest methods on the synthetic data, but is relatively weak on the real-world data. We believe that this can be understood in terms of model flexibility. IFTPP uses a simple parametric form (a mixture of log-normals) to parametrize the subsequent event time distribution (see Section 3). This provides a strong inductive bias and allows it to fit simple synthetic distributions very effectively, but limits its capability to model more realistic data sets. EventFlow, in contrast, makes no parametric assumptions, offering greater flexibility. As a result, it may not outperform simple parametric models on very simple synthetic datasets, but this flexibility is precisely what enables strong performance on complex real-world data, which we consider the more meaningful benchmark.

The Add-and-Thin method (mean rank: **2.4**) is often similarly strong, but struggles on the SC dataset. While the NHP (mean rank: **3.7**) can obtain good fits, this appears to be dataset dependent, with weak results on the NSR, SC, and Reddit-S datasets. The diffusion baseline (mean rank: **4.8**) is our weakest baseline, which is perhaps unsurprising as this model can only be trained to maximize the likelihood of a subsequent event and not the overall sequence likelihood.

6 CONCLUSION

We introduced `EventFlow`, a non-autoregressive generative model for temporal point processes. Our approach extends flow matching to the TPP setting by constructing a continuous transformation between a simple reference distribution and the data distribution over event sequences. Concretely, `EventFlow` generates event sequences by first drawing a sequence of event times from a reference process, and then transporting these events along a learned vector field. Beyond modeling event times, our framework also captures the distribution over the number of events, which we learn via a regularized cross-entropy objective.

Empirically, we demonstrate that `EventFlow` achieves state-of-the-art performance on multi-step forecasting tasks and competitive results on unconditional generation across a range of standard benchmarks. These results highlight the promise of non-autoregressive, flow-based approaches as a flexible alternative to classical intensity-based models for TPPs.

Limitations and Outlook Our work focuses on modeling event times, and we do not consider marked or spatiotemporal point processes. Extending `EventFlow` to incorporate marks or spatial structure is a natural and promising direction for future work, though it may require new coupling and interpolation strategies.

Additionally, our current construction does not explicitly enforce support constraints on $[0, T]$, as it relies on Gaussian noising in Equation (6). Addressing this limitation, e.g., by designing constrained flows, could potentially lead to further gains in performance. Finally, our experimental comparisons adopt the neural architectures used in prior work, which differ across baseline methods. A more controlled study of architectural choices, and their interaction with flow-based objectives, would help isolate the gains attributable to the modeling framework itself.

More broadly, we believe our work opens the door to a new class of generative models for point processes based on flow-based ideas, and we hope it encourages further exploration of non-autoregressive approaches.

Acknowledgements

We thank the reviewers for their feedback on improving the paper. This work was supported by National Science Foundation under awards NSF 2505006 and NSF 2425932, by the National Institutes of Health under awards R01-LM013344 and R01CA297869, by the UK EPSRC grant EP/Y037200/1, by the Hasso Plattner Institute (HPI) Research Center in Machine Learning and Data Science at UCI, and by funding support from Google and from SAP.

References

Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *The Eleventh International Conference on Learning Representations*, 2023.

Sergio Albeverio, Yu G Kondratiev, and Michael Röckner. Analysis and geometry on configuration spaces. *Journal of Functional Analysis*, 154(2):444–500, 1998.

Tanguy Bosser and Souhaib Ben Taieb. On the predictive accuracy of neural temporal point process models for continuous-time event data. *Transactions on Machine Learning Research*, 2023.

Erik Buhmann, Cedric Ewen, Darius A Faroughy, Tobias Golling, Gregor Kasieczka, Matthew Leigh, Guillaume Quétant, John Andrew Raine, Debajyoti Sengupta, and David Shih. Epic-ly fast particle cloud generation with flow-matching and diffusion. *arXiv preprint arXiv:2310.00049*, 2023.

Andrew Campbell, Jason Yim, Regina Barzilay, Tom Rainforth, and Tommi Jaakkola. Generative flows on discrete state-spaces: Enabling multimodal flows with applications to protein co-design. In *Proceedings of the 41st International Conference on Machine Learning*, pages 5453–5512, 2024.

Yuxin Chang, Alex James Boyd, Cao Xiao, Taha Kass-Hout, Parminder Bhatia, Padhraic Smyth, and Andrew Warrington. Deep continuous-time state-space models for marked event sequences. In *Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Daryl J Daley and David Vere-Jones. *An Introduction to the Theory of Point Processes: Volume I: Elementary Theory and Methods*. Springer, 2003.

Quan Dao, Hao Phung, Binh Nguyen, and Anh Tran. Flow matching in latent space. *arXiv preprint arXiv:2307.08698*, 2023.

Felix Draxler, Yang Meng, Kai Nelson, Lukas Laskowski, Yibo Yang, Theofanis Karaletsos, and Stephan Mandt. Transformers for mixed-type event sequences. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Nan Du, Hanjun Dai, Rakshit Trivedi, Utkarsh Upadhyay, Manuel Gomez-Rodriguez, and Le Song. Recurrent marked temporal point processes: Embedding event history to vector. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1555–1564, 2016.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13:723–773, 2012.

Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58 (1):83–90, 1971.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Valerie Isham and Mark Westcott. A self-correcting point process. *Stochastic Processes and Their Applications*, 8(3):335–347, 1979.

Olav Kallenberg. *Random Measures, Theory and Applications*, volume 1. Springer, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Represenations*, 2015.

Peter AW Lewis and Gerald S Shedler. Simulation of nonhomogeneous Poisson processes with degree-two exponential polynomial rate function. *Operations Research*, 27(5):1026–1040, 1979.

Haitao Lin, Cheng Tan, Lirong Wu, Zhangyang Gao, and Stan Z Li. An empirical study: extensive deep temporal point process. *arXiv preprint arXiv:2110.09823*, 2021.

Haitao Lin, Lirong Wu, Guojiang Zhao, Liu Pai, and Stan Z Li. Exploring generative neural temporal point process. *Transactions on Machine Learning Research*, 2022.

Yaron Lipman, Ricky T Q Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Yaron Lipman, Marton Havasi, Peter Holderrieth, Neta Shaul, Matt Le, Brian Karrer, Ricky TQ Chen, David Lopez-Paz, Heli Ben-Hamu, and Itai Gat. Flow matching guide and code. *arXiv preprint arXiv:2412.06264*, 2024.

Xingchao Liu, Chengyue Gong, and Qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

David Lüdke, Marin Biloš, Oleksandr Shchur, Marten Lienen, and Stephan Günnemann. Add and thin: Diffusion for temporal point processes. *Advances in Neural Information Processing Systems*, 36:56784–56801, 2023.

David Lüdke, Marten Lienen, Marcel Kollovieh, and Stephan Günnemann. Edit-based flow matching for temporal point processes. In *The Fourteenth International Conference on Learning Representations*, 2026.

Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. *arXiv preprint arXiv:2401.08740*, 2024.

Hongyuan Mei and Jason M Eisner. The neural Hawkes process: A neurally self-modulating multivariate point process. *Advances in Neural Information Processing Systems*, 30, 2017.

Hongyuan Mei, Guanghui Qin, and Jason Eisner. Imputing missing events in continuous-time event streams. In *International Conference on Machine Learning*, pages 4475–4485, 2019.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

Yosihiko Ogata. On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory*, 27(1):23–31, 1981.

Yosihiko Ogata. Space-time point-process models for earthquake occurrences. *Annals of the Institute of Statistical Mathematics*, 50:379–402, 1998.

Takahiro Omi, Kazuyuki Aihara, et al. Fully neural network based model for general temporal point processes. *Advances in Neural Information Processing Systems*, 32, 2019.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021.

Jakob Gulddahl Rasmussen. Temporal point processes: The conditional intensity function. *Lecture Notes*, 2011.

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. In *International Conference on Learning Representations*, 2020a.

Oleksandr Shchur, Nicholas Gao, Marin Biloš, and Stephan Günnemann. Fast and flexible temporal point processes with triangular maps. *Advances in Neural Information Processing Systems*, 33:73–84, 2020b.

Oleksandr Shchur, Ali Caner Türkmen, Tim Januschowski, and Stephan Günnemann. Neural temporal point processes: A review. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4585–4593. International Joint Conferences on Artificial Intelligence Organization, 8 2021.

Hannes Stark, Bowen Jing, Chenyu Wang, Gabriele Corso, Bonnie Berger, Regina Barzilay, and Tommi Jaakkola. Dirichlet flow matching with applications to DNA sequence design. In *Proceedings of the 41st International Conference on Machine Learning*, pages 46495–46513, 2024.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

Lemeng Wu, Dilin Wang, Chengyue Gong, Xingchao Liu, Yunyang Xiong, Rakesh Ranjan, Raghuraman Krishnamoorthi, Vikas Chandra, and Qiang Liu. Fast point cloud generation with straight flows. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9445–9454, 2023.

Jianfeng Xiang, Zelong Lv, Sicheng Xu, Yu Deng, Ruicheng Wang, Bowen Zhang, Dong Chen, Xin Tong, and Jiaolong Yang. Structured 3d latents for scalable and versatile 3d generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 21469–21480, 2025.

Shuai Xiao, Mehrdad Farajtabar, Xiaojing Ye, Junchi Yan, Le Song, and Hongyuan Zha. Wasserstein learning of deep generative point process models. *Advances in Neural Information Processing Systems*, 30, 2017a.

Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen Chu. Modeling the intensity function of point process via recurrent neural networks.

In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017b.

Lizhen Xu, Jason A Duan, and Andrew Whinston. Path to purchase: A mutually exciting point process model for online advertising and conversion. *Management Science*, 60(6):1392–1412, 2014.

Siqiao Xue, Xiaoming Shi, James Zhang, and Hongyuan Mei. Hypro: A hybridly normalized probabilistic model for long-horizon prediction of event sequences. *Advances in Neural Information Processing Systems*, 35:34641–34650, 2022.

Siqiao Xue, Xiaoming Shi, Zhixuan Chu, Yan Wang, Hongyan Hao, Fan Zhou, Caigao Jiang, Chen Pan, James Y. Zhang, Qingsong Wen, Jun Zhou, and Hongyuan Mei. EasyTPP: Towards open benchmarking temporal point processes. In *International Conference on Learning Representations*, 2024.

Chenghao Yang, Hongyuan Mei, and Jason Eisner. Transformer embeddings of irregularly spaced events and their participants. In *International Conference on Learning Representations*, 2022.

Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. Self-attentive Hawkes process. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11183–11193, 2020.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer Hawkes process. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11692–11702, 2020.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes; Section 4 contains a description of our model and pseudocode.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. No; we present a deep learning algorithm for which such analyses are typically not included.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. No; code will be released at a later date.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes; See Section 4.
   (b) Complete proofs of all theoretical results. Yes; see Appendix B.
   (c) Clear explanations of any assumptions. Yes; See Section 4.

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes; a link to our code is provided.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes; this is detailed in the appendix.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes; this is detailed in Appendix C.
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes; this is detailed in Appendix C.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes; citations are provided throughout.
   (b) The license information of the assets, if applicable. Not Applicable
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable; we provide no new assets
   (d) Information about consent from data providers/curators. Not Applicable; we use standard open-source datasets.
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable

EventFlow: Forecasting Temporal Point Processes with Flow Matching: Supplementary Materials

A DATASETS

In this section, we provide some additional details regarding the datasets used in this work. In Table 4, we report the number of sequences in each dataset, some basic statistics regarding the number of events in each sequence, and their support $[0, T]$ and chosen forecast window ΔT. In all datasets, we use 60% of the sequences for training, 20% for validation, and the remaining 20% for testing.

Synthetic Datasets Our synthetic datasets are adopted from those proposed by Omi et al. (2019). Each of these datasets consists of $1,000$ sequences supported on $\mathcal{T} = [0, 100]$. These synthetic datasets are chosen as they exhibit a wide range of behavior, ranging from i.i.d. inter-arrival times to self-correcting processes which discourage rapid bursts of events. We refer to Section 4 of Omi et al. (2019) for details.

Real-World Datasets We use the set of real-world datasets proposed in Shchur et al. (2020b), which constitute a set of standard benchmark datasets for unmarked TPPs. We refer to Appendix D of Shchur et al. (2020b) for additional details. With the exception of PUBG, these datasets are supported on $\mathcal{T} = [0, 24]$, i.e. each sequence corresponds to a single day. For the PUBG dataset, $\mathcal{T} = [0, 38]$ corresponds to the maximum length (in minutes) of an online game of PUBG. We note that PUBG has the largest number of sequences (which can lead to slow training), and the Reddit-C and Reddit-S datasets have long sequences (which can lead to slow training and high memory costs).

For selecting the forecasting horizon ΔT, we follow the choice made in Lüdke et al. (2023) to facilitate comparison and reproducibility. We note the initial times $T_0 \in [\Delta T, T - \Delta T]$ are selected to be at least ΔT to avoid forecasting sequences with no observations in the history.

Table 4: Some basic summary statistics of the datasets we consider in this work.

	Sequences	Mean length	Std length	Range length	Support	ΔT
Hawkes1	1000	95.4	45.8	[14, 300]	[0, 100]	–
Hawkes2	1000	97.2	49.1	[18, 355]	[0, 100]	–
Nonstationary Poisson	1000	100.3	9.8	[71, 134]	[0, 100]	–
Nonstationary Renewal	1000	98	2.9	[86, 100]	[0, 100]	–
Stationary Renewal	1000	109.2	38.1	[1, 219]	[0, 100]	–
Self-Correcting	1000	100.3	0.74	[98, 102]	[0, 100]	–
PUBG	3001	76.5	8.8	[26, 97]	[0, 38]	5
Reddit-C	1356	295.7	317.9	[1, 2137]	[0, 24]	4
Reddit-S	1094	1129	359.5	[363, 2658]	[0, 24]	4
Taxi	182	98.4	20	[12, 140]	[0, 24]	4
Twitter	2019	14.9	14	[1, 169]	[0, 24]	4
Yelp-Airport	319	30.5	7.5	[9, 55]	[0, 24]	4
Yelp-Miss.	319	55.2	15.9	[3, 107]	[0, 24]	4

B PROOFS

In this section, we provide a proof of Proposition 1, showing a necessary and sufficient condition for the existence of balanced couplings.

Proposition 1 (Existence of Balanced Couplings)**.**
Let $\mu_0, \mu_1 \in \mathbb{P}(\Gamma)$ be two TPPs. The set of balanced couplings $\Pi_b(\mu_0, \mu_1)$ is nonempty if and only if have the same distribution over event counts, i.e., $\mu_0(n) = \mu_1(n)$ for every $n \in \mathbb{Z}_{\geq 0}$.

Proof. Let $A_1, A_2 \subseteq \Gamma$ be Borel measurable (Daley and Vere-Jones, 2003, Prop. 5.3) subsets of the configuration space Γ, i.e. each of A_1, A_2 is a measurable collection of event sequences. Observe that for $i = 1, 2$, each A_i can be written as a disjoint union

$$A_i^n = \bigcup_{n=0}^{\infty} \mathcal{T}^n \cap A_i \tag{13}$$

i.e. $A_i^n \subseteq A_i$ is the subset of A_i containing only sequences with n events. Note each A_i^n is a Borel measurable subset of $\mathcal{T}^n$.

Now, suppose that $\mu(n) = \nu(n)$ have equal event count distributions. We define the coupling $\rho \in \mathbb{P}(\Gamma \times \Gamma)$ by

$$\rho(A_1 \times A_2) = \sum_{n=0}^{\infty} \mu(n) \mu^n(A_1^n) \nu^n(A_2^n). \tag{14}$$

Here, in a slight abuse of notation, we use μ^n, ν^n to denote the corresponding joint probability measures over n events, i.e., both are Borel probability measures on $\mathcal{T}^n$. Since the n-dimensional projection of Γ in Equation (13) is simply $\mathcal{T}^n$, it is immediate that $\rho(A_1 \times \Gamma) = \mu(A_1)$ and $\rho(\Gamma \times A_2) = \nu(A_2)$, so that ρ is indeed a coupling. Moreover, it is clear that the coupling is balanced.

Conversely, suppose $\rho \in \Pi_b(\mu_0, \mu_1)$ is a balanced coupling. Let $N : \Gamma \to \mathbb{Z}_{\geq 0}$ be the event counting functional and let $\pi^1, \pi^2 : \Gamma \times \Gamma \to \Gamma$ denote the canonical projections of $\Gamma \times \Gamma$ onto its components. That is, $\pi^1 : (\gamma_0, \gamma_1) \mapsto \gamma_0$ and $\pi^2 : (\gamma_0, \gamma_1) \mapsto \gamma_1$. Furthermore, let $(N, N) : \Gamma \times \Gamma \to \mathbb{Z}_{\geq 0} \times \mathbb{Z}_{\geq 0}$ denote the product of the counting functional, i.e. $(N, N)(\gamma_0, \gamma_1) = (N(\gamma_0), N(\gamma_1))$. Note that the pushforward $N_{\#}\mu$ yields the event count distribution $\mu(n)$ of μ (and analogously for ν).

Now, observe that composing the projections and counting functionals yields

$$\pi^1 \circ (N, N) = N \circ \pi^1 \qquad \pi^2 \circ (N, N) = N \circ \pi^2. \tag{15}$$

As ρ is a coupling, we have that $\mu = \pi^1_{\#}\rho$ and $\nu = \pi^2_{\#}\rho$. From these observations, it follows that

$$\begin{aligned}
N_{\#}\mu &= N_{\#}\left(\pi^1_{\#}\rho\right) && (16)\\
&= (N \circ \pi^1)_{\#}\rho && (17)\\
&= (\pi^1 \circ (N, N))_{\#}\rho && (18)\\
&= \pi^1_{\#}\left((N, N)_{\#}\rho\right) && (19)\\
&= \pi^2_{\#}\left((N, N)_{\#}\rho\right) && (20)\\
&= N_{\#}\nu && (21)
\end{aligned}$$

where the equality in the penultimate line follows from the fact that ρ is balanced. Thus, we have shown that the existence of a balanced coupling implies that $N_{\#}\mu = N_{\#}\nu$, i.e. the event count distributions are equal. □

C EVENTFLOW ARCHITECTURE AND TRAINING DETAILS

Here, we provide additional details regarding the parametrization and training of our `EventFlow` model. Our model is based on the transformer architecture (Vaswani et al., 2017; Yang et al., 2022), due to its general ability to handle variable length inputs and outputs, high flexibility, and ability to incorporate long-range

interactions. We emphasize that this is an effective architecture choice, but our method is not necessarily tied to this architecture. In all settings, our reference measure μ_0 is specified with $q = \mathcal{N}(0, I)$. All models (including the baselines) were trained on a cluster of six NVIDIA A6000 GPUs with 24Gb of VRAM. No hardware parallelization was used, i.e., each model was only trained on a single GPU.

Model Parametrization Our unconditional model takes in a sequence $\gamma_s = (t_s^1, \dots, t_s^n)$ and flow time s. We first embed the sequence times t^k, the flow-time s, and the sequence position indices. These position indices are handled by sinusoidal embeddings followed by an additional linear layer. There are three linear layers in total: one for the flow time, one shared across the sequence times, and one for the position indices. These embeddings are added together to create a representation for each element of the sequence, and we apply a standard transformer to this sequence to produce a sequence of vectors of length $N(\gamma_s)$. Finally, each of these vectors is projected to one dimension via a final linear layer with shared weights to produce the vector field $v_\theta(\gamma_s, s)$. See Figure 3.

For the conditional model, we use a standard transformer encoder-decoder architecture. We first embed the history sequence times $\mathcal{H}$ and the sequence position indices in a manner analogous to the above. The model was provided the start of the prediction window T_0 by concatenating it as the final event in $\mathcal{H}$. This yielded better results than encoding the start of the prediction window separately. We feed these embeddings through the transformer encoder produce an intermediate representation $e_{\mathcal{H}}$.

For the decoder, we provide the model with the current state γ_s (corresponding to the generated event times at flow-time s), the flow-time s, and the corresponding positional indices. These are embedded as previously described, before being passed into the transformer decoder. The history encoding $e_{\mathcal{H}}$ is provided to the decoder via cross-attention in the intermediate layer. This produces a sequence of $N(\gamma_s)$ vectors, which we again pass through a final linear layer to produce the final conditional vector field $v_\theta(\gamma_s, s, e_{\mathcal{H}})$. See Figure 4.

Our architecture for predicting the number of future events given a history, i.e. $p_\phi(n \mid \mathcal{H})$, is again based on the transformer decoder, sharing the same overall architecture as our unconditional model. The key difference is that we instead take a mean of the final sequence embeddings before passing this through a small MLP to produce the final logit. See Figure 5. We note that this requires specifying a maximum value $N_{\max}$, which we set as the maximum sequence length seen in the training data. In early experiments, we also parametrized the model not through a logit but rather as a mixture of Poissons or a mixture of Negative Binomials. While this no longer has the limitation of assuming a value for $N_{\max}$, we found that this approach yielded worse results.

Training and Tuning We normalize all sequences to the range $[-1, 1]$, using the overall min/max event time seen in the training data. All sequences are generated on this normalized scale, prior to re-scaling the sequence back to the original data range before evaluation. Our models are trained with the Adam (Kingma and Ba, 2015) optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for $30,000$ steps with a cosine scheduler (Loshchilov and Hutter, 2017), which cycled every $10,000$ steps. Final hyperparameters were selected by best performance on the validation dataset achieved at any point during the training, where models were evaluated 10 times throughout their training.

To tune our model, we performed a grid search over learning rates in $\{5 \times 10^{-3}, 10^{-3}, 5 \times 10^{-4}\}$ and dropout probabilities in $\{0, 0.1, 0.2\}$. Overall, we found that learning rates of 10^{-2} or larger often caused the model to diverge. We use 6 transformer layers, 8 attention heads, and an embedding dimension of 512 across all settings, except for the Reddit-C and Reddit-S datasets where we use 4 heads and an embedding dimension of 128 due to the increased memory cost of these datasets.

To train the event count model $p_\phi(n \mid \mathcal{H})$, we seek to minimize the regularized cross-entropy loss

$$\mathcal{L}(\phi) = \mathbb{E}_{n,\mathcal{H}}\left[-\log p_\phi(n \mid \mathcal{H}) + \frac{\alpha}{N_{\max}} \sum_{k=1}^{N_{\max}} p_\phi(k \mid \mathcal{H})(n-k)^2\right] \tag{22}$$

where $\mathcal{H}$ is a given history and n is the number of events in $\mathcal{T}$ following this history. Note that this is a cross-entropy loss where the regularization term can be viewed as a squared optimal transport distance between the distribution $p_\phi(n \mid \mathcal{H})$ and a delta distribution $\delta[n]$ at the true n. This regularization term encourages the values of $p_\phi(n \mid \mathcal{H})$ to be smooth in n, as it penalizes predictions which are far fronm the true n more heavily than those which are nearby. We searched for values of α, the weight associated with the OT-loss, over the set $\{0, 1/N_{\max}, 10/N_{\max}, 100/N_{\max}, 1000/N_{\max}\}$.

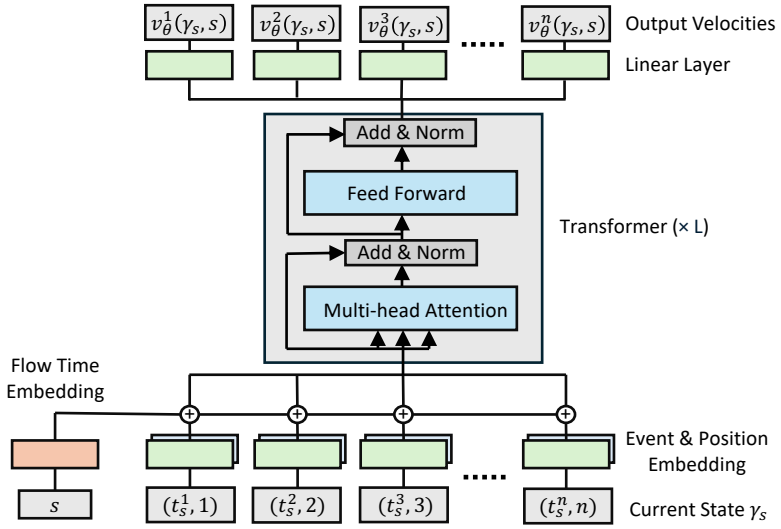

Figure 3: Overview of our model architecture for unconditional generation. The model takes as input the flow time s and current sequence state $\gamma_s = \sum_{k=1}^{n} \delta[t_s^k]$. Each input is projected to a fixed-length vector via a learnable embedding. The resulting embeddings are added and passed to the transformer model, which produces a sequence of output velocities $v_\theta(\gamma_s, s)$ with $N(\gamma_s)$ components.

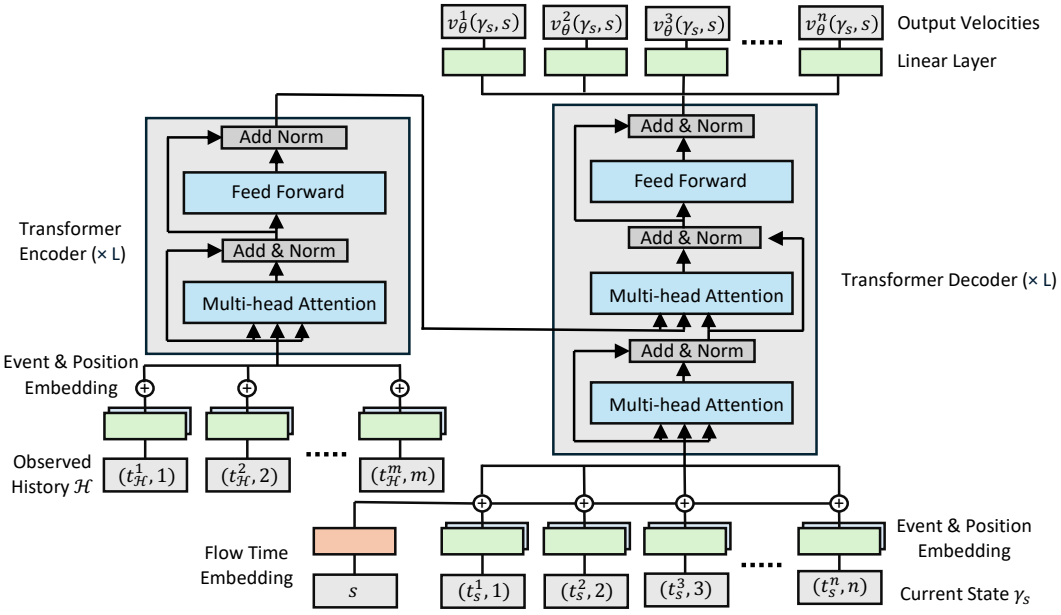

Figure 4: Overview of our model architecture for conditional generation. The encoder (left) takes as input the observed history $\mathcal{H}$, which is embedded in a fashion analogous to our unconditional model. The decoder (right) takes as input the flow time s and current state $\gamma_s = \sum_{k=1} \delta[t_s^k]$. These are embedded and passed through the decoder, which applies cross attention to produce the conditional velocities $v_\theta(\gamma_s, s, e_\mathcal{H})$.

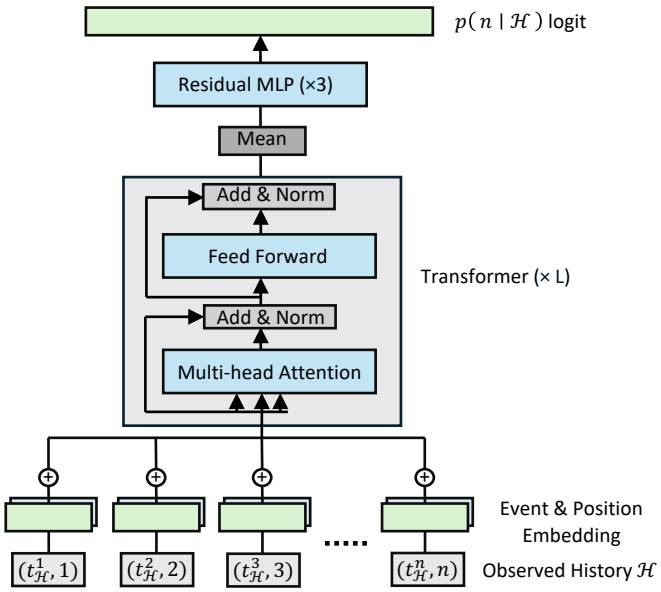

Figure 5: Overview of our architecture modeling the event count distribution $p_\phi(n \mid \mathcal{H})$. The model takes as input an observed history $\mathcal{H}$. As in our other architectures, the events are embedded and passed through a transformer. Here, the final sequence embedding output by the transformer is averaged and passed through an additional residual MLP with three layers to produce the logit corresponding to $p_\phi(n \mid \mathcal{H})$.

Table 5: The best hyperparameter settings found for the vector field v_θ in our `EventFlow` method on the unconditional generation task.

	Learning Rate	Emb. Dim.	MLP Dim	Heads	Transformer Layers	Dropout
Hawkes1	10^{-3}	512	2048	8	6	0.1
Hawkes2	10^{-3}	512	2048	8	6	0.1
Nonstationary Poisson	10^{-3}	512	2048	8	6	0.1
Nonstationary Renewal	10^{-3}	512	2048	8	6	0.1
Stationary Renewal	10^{-3}	512	2048	8	6	0.1
Self-Correcting	10^{-3}	512	2048	8	6	0.1
PUBG	5×10^{-4}	512	2048	8	6	0.1
Reddit-C	10^{-3}	128	256	4	6	0.1
Reddit-S	5×10^{-3}	128	256	4	6	0.1
Taxi	5×10^{-4}	512	2048	8	6	0.1
Twitter	10^{-3}	512	2048	8	6	0.1
Yelp-Airport	5×10^{-4}	512	2048	8	6	0.1
Yelp-Miss.	10^{-3}	512	2048	8	6	0.1

Table 6: The best hyperparameter settings found for the vector field v_θ in our `EventFlow` method on the forecasting task.

	Learning Rate	Emb. Dim.	MLP Dim.	Heads	Transformer Layers	Dropout
PUBG	5×10^{-3}	512	2048	8	6	0.1
Reddit-C	10^{-3}	128	256	4	6	0.2
Reddit-S	10^{-3}	128	256	4	6	0.2
Taxi	5×10^{-4}	512	2048	8	6	0.1
Twitter	5×10^{-4}	512	2048	8	6	0.1
Yelp-Airport	10^{-3}	512	2048	8	6	0.2
Yelp-Miss.	5×10^{-4}	512	2048	8	6	0.1

Table 7: The best hyperparameter settings found for the event count predictor $p_\phi(n \mid \mathcal{H})$ in our `EventFlow` method on the forecasting task.

	Learning Rate	α/N_{max}	Emb. Dim.	MLP Dim.	Heads	Transformer Layers	Dropout
PUBG	10^{-3}	1.0	512	2048	8	6	0.2
Reddit-C	10^{-3}	1000.0	128	256	4	6	0.2
Reddit-S	10^{-3}	1000.0	128	256	4	6	0.0
Taxi	10^{-3}	1000.0	512	2048	8	6	0.2
Twitter	10^{-3}	1000.0	512	2048	8	6	0.2
Yelp-Airport	10^{-3}	1000.0	512	2048	8	6	0.2
Yelp-Miss.	10^{-3}	100.0	512	2048	8	6	0.2

D ADDITIONAL DETAILS ON BASELINES

In this section, we provide additional details regarding our baseline methods. All methods are trained at a batch size of 64 for $1,000$ epochs, using early stopping on the validation set loss. In early experiments, we also evaluated AttNHP (Zuo et al., 2020), a variant of the NHP which uses an attention-based encoder, but found it to be prohibitively expensive in terms of memory cost (requiring more than 24 GB of VRAM) and, as a result, do not include it in our results.

IFTPP Our first baseline is the intensity-free TPP model of Shchur et al. (2020a). This model uses an RNN encoder and a mixture of log-normal distributions to parametrize the decoder. We directly use the implementation provided by the authors.[3]. We train for $1,000$ epochs with early stopping based on the validation set loss. To tune this baseline, we performed a grid search over learning rates in $\{10^{-4}, 10^{-3}, 10^{-2}\}$, weight decays in $\{0, 10^{-6}, 10^{-5}, 10^{-4}\}$, history embedding dimensions $\{32, 64, 128\}$, and mixture component counts $\{8, 16, 32, 64\}$. Our best hyperparameters can be found in Table 8 and Table 9.

Table 8: The best hyperparameter settings found for IFTPP on the unconditional generation task.

	Learning Rate	Weight Decay	Embedding Dimension	Mixture Components
Hawkes1	10^{-3}	10^{-4}	32	8
Hawkes2	10^{-2}	0	32	8
Nonstationary Poisson	10^{-3}	10^{-6}	128	8
Nonstationary Renewal	10^{-2}	10^{-6}	64	16
Stationary Renewal	10^{-3}	10^{-4}	32	8
Self-Correcting	10^{-3}	10^{-6}	32	64
PUBG	10^{-2}	0	128	32
Reddit-C	10^{-3}	10^{-4}	64	16
Reddit-S	10^{-2}	10^{-4}	64	16
Taxi	10^{-2}	10^{-5}	128	64
Twitter	10^{-3}	10^{-4}	64	6
Yelp-Airport	10^{-2}	10^{-6}	64	64
Yelp-Miss.	10^{-3}	10^{-4}	32	8

Table 9: The best hyperparameter settings found for IFTPP on the forecasting task.

	Learning Rate	Weight Decay	Embedding Dimension	Mixture Components
PUBG	10^{-4}	10^{-6}	32	32
Reddit-C	10^{-2}	0	64	8
Reddit-S	10^{-2}	0	64	16
Taxi	10^{-3}	10^{-6}	128	8
Twitter	10^{-2}	10^{-5}	32	8
Yelp-Airport	10^{-2}	10^{-6}	128	32
Yelp-Miss.	10^{-2}	10^{-6}	32	8

[3]URL: https://github.com/shchur/ifl-tpp

NHP We additionally compare against the Neural Hawkes Process of Mei and Eisner (2017). This model uses an LSTM encoder and a parametric form, whose weights are modeled by a neural network, to model the conditional intensity function. In practice, we use the implementation proved by the EasyTPP benchmark (Xue et al., 2024), as this version implements the necessary thinning algorithm for sampling.[4] We perform a grid search over learning rates in $\{10^{-4}, 10^{-3}, 10^{-2}\}$ and embedding dimensions in $\{32, 64, 128\}$. These hyperparameters are chosen as the EasyTPP implementation allows these to be configured easily. Our best hyperparameters are reported in Table 10 and Table 11.

Table 10: The best hyperparameter settings found for NHP on the unconditional generation task.

	Learning Rate	Embedding Dimension
Hawkes1	10^{-3}	64
Hawkes2	10^{-3}	64
Nonstationary Poisson	10^{-3}	64
Nonstationary Renewal	10^{-4}	64
Stationary Renewal	10^{-3}	64
Self-Correcting	10^{-3}	64
PUBG	10^{-4}	64
Reddit-C	10^{-2}	64
Reddit-S	10^{-2}	64
Taxi	10^{-2}	64
Twitter	10^{-4}	64
Yelp-Airport	10^{-3}	128
Yelp-Miss.	10^{-2}	64

Table 11: The best hyperparameter settings found for NHP on the forecasting task.

	Learning Rate	Embedding Dimension
PUBG	10^{-3}	128
Reddit-C	10^{-2}	64
Reddit-S	10^{-2}	64
Taxi	10^{-2}	128
Twitter	10^{-2}	128
Yelp-Airport	10^{-3}	64
Yelp-Miss.	10^{-2}	64

Diffusion Our diffusion baseline is based on the implementation of Lin et al. (2022), and our decoder model architecture is taken directly from the code of Lin et al. (2022).[5] At a high level, this model is a discrete-time diffusion model (Ho et al., 2020) trained to generate a single inter-arrival time given a history embedding. Note that as the likelihood is not available in diffusion models, the CDF in the likelihood in Equation (3) is not tractable. Instead, the model is trained by maximizing an ELBO of only the subsequent inter-arrival time.

In preliminary experiments, we found that the codebase provided by Lin et al. (2022) often produced `NaN` values during sampling, prompting us to make several changes. First, we use the RNN encoder from Shchur et al. (2020a), i.e. the same encoder as the IFTPP baseline, to reduce the memory requirements of the model. Second, we do not log-scale the inter-arrival times as suggested by Lin et al. (2022), as we found that this often led to overflow and underflow issues at sampling time. Third, we do not normalize the data via standardization (i.e., subtracting off the mean inter-arrival time and dividing by the standard deviation), but rather, we scale the inter-arrival times so that they are in the bounded range $[-1, 1]$. This is aligned with standard diffusion implementations (Ho et al., 2020), and allows us to perform clipping at sampling time to avoid the accumulation of errors. With these changes, our diffusion baseline is competitive, and able to obtain stronger results than previous work has reported (Lüdke et al., 2023).

[4]URL: https://github.com/ant-research/EasyTemporalPointProcess
[5]URL: https://github.com/EDAPINENUT/GNTPP

We use 1000 diffusion steps and the cosine beta schedule (Nichol and Dhariwal, 2021), and we train the model on the simplified ϵ-prediction loss of Ho et al. (2020). We train for $1,000$ epochs with early stopping based on the validation set loss. To tune this baseline, we performed a grid search over learning rates in $\{10^{-4}, 10^{-3}, 10^{-2}\}$, weight decays in $\{0, 10^{-6}, 10^{-5}, 10^{-4}\}$, history embedding dimensions $\{32, 64, 128\}$, and layer numbers $\{2, 4, 6\}$. Our best hyperparameters can be found in Table 12 and Table 13.

Table 12: The best hyperparameter settings found for diffusion on the unconditional generation task.

	Learning Rate	Weight Decay	Embedding Dimension	Layers
Hawkes1	10^{-3}	10^{-6}	64	2
Hawkes2	10^{-2}	10^{-5}	64	4
Nonstationary Poisson	10^{-3}	10^{-5}	128	2
Nonstationary Renewal	10^{-3}	10^{-4}	64	2
Stationary Renewal	10^{-2}	0	32	6
Self-Correcting	10^{-3}	0	32	6
PUBG	10^{-3}	0	64	2
Reddit-C	10^{-3}	10^{-6}	128	4
Reddit-S	10^{-3}	0	64	4
Taxi	10^{-2}	0	128	4
Twitter	10^{-3}	10^{-4}	64	6
Yelp-Airport	10^{-2}	0	32	2
Yelp-Miss.	10^{-2}	10^{-5}	128	2

Table 13: The best hyperparameter settings found for diffusion on the forecasting task.

	Learning Rate	Weight Decay	Embedding Dimension	Layers
PUBG	10^{-4}	10^{-5}	32	6
Reddit-C	10^{-2}	10^{-6}	64	6
Reddit-S	10^{-3}	0	64	4
Taxi	10^{-3}	10^{-6}	32	2
Twitter	10^{-4}	10^{-5}	64	6
Yelp-Airport	10^{-4}	10^{-5}	64	6
Yelp-Miss.	10^{-3}	10^{-5}	32	4

Add-and-Thin We compare to the `Add-and-Thin` model of Lüdke et al. (2023) as a recently proposed non-autoregressive baseline. We directly run the code provided by the authors without additional modifications.[6] We do, however, perform a slightly larger hyperparameter sweep than Lüdke et al. (2023), in order to ensure a fair comparison between the methods considered. We train for $1,000$ epochs with early stopping on the validation loss. Tuning is performed via a grid search over learning rates in $\{10^{-4}, 10^{-3}, 10^{-2}\}$ and number of mixture components in $\{8, 16, 32, 64\}$. We choose to tune only these hyperparameters in order to follow the implementation provided by the authors. Our best hyperparameters can be found in Table 14 and Table 15.

Table 14: The best hyperparameter settings found for Add-and-Thin on the unconditional generation task.

	Learning Rate	Mixture Components
Hawkes1	10^{-3}	32
Hawkes2	10^{-2}	32
Nonstationary Poisson	10^{-2}	16
Nonstationary Renewal	10^{-2}	8
Stationary Renewal	10^{-2}	8
Self-Correcting	10^{-4}	8
PUBG	10^{-3}	8
Reddit-C	10^{-2}	32
Reddit-S	10^{-2}	16
Taxi	10^{-2}	8
Twitter	10^{-4}	32
Yelp-Airport	10^{-4}	8
Yelp-Miss.	10^{-2}	64

Table 15: The best hyperparameter settings found for Add-and-Thin on the forecasting task.

	Learning Rate	Mixture Components
PUBG	10^{-2}	64
Reddit-C	10^{-2}	16
Reddit-S	10^{-2}	64
Taxi	10^{-2}	8
Twitter	10^{-3}	8
Yelp-Airport	10^{-2}	32
Yelp-Miss.	10^{-3}	16

[6] URL: https://github.com/davecasp/add-thin

E ADDITIONAL EXPERIMENTS

This section contains additional empirical evaluations, supplementing Section 5.

E.1 MARE

In Table 16, we evaluate the performance of the various models only in terms of the predicted number of events in the forecast. To do so, we measure the mean absolute relative error (MARE) given by

$$\text{MARE} = \mathbb{E}_{n,\mathcal{H},\hat{n}\sim p_\phi(n|\mathcal{H})} \left| \frac{\hat{n} - n}{n} \right| \tag{23}$$

where n represents the true number of points in a sequence following a history $\mathcal{H}$ and $\hat{n} \sim p_\phi(n \mid \mathcal{H})$ represents the predicted number of points. The expectation is estimated empirically on the testing set. As our method directly predicts the number of events n by sampling from the learned distribution $p_\phi(n \mid \mathcal{H})$, this serves as a direct evaluation of this component of our model. We note that the baselines only model n implicitly. For example, the autoregressive models sample new events until an event is generated outside the support $\mathcal{T}$. As an ablation, we also report the MARE values for our method without regularization, i.e., setting $\alpha = 0$. We see that the regularization significantly improves the event count predictions.

Table 16: MARE values evaluating the predicted number of events when forecasting. Mean values ± one standard deviation are reported over five random seeds. The lowest MARE on each dataset is indicated and bold, and the second lowest is indicated by an underline.

	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp-A	Yelp-M
IFTPP	1.05±0.14	1.69±0.39	0.79±0.20	0.60±0.11	0.88±0.08	0.76±0.07	0.76±0.05
NHP	1.02±0.08	<u>0.95</u>±0.01	1.00±0.0004	0.67±0.11	2.48±0.40	0.80±0.22	1.07±0.34
Diffusion	1.95±0.48	1.28±0.09	1.12±0.56	0.49±0.07	0.66±0.04	0.65±0.07	0.72±0.07
Add-and-Thin	<u>0.43</u>±0.01	0.99±0.10	<u>0.38</u>±0.01	<u>0.33</u>±0.02	<u>0.60</u>±0.02	**0.42**±0.01	**0.46**±0.03
EventFlow (ours)	**0.40**±0.01	**0.70**±0.07	**0.16**±0.01	**0.28**±0.01	**0.46**±0.03	<u>0.56</u>±0.06	<u>0.50</u>±0.02
EventFlow ($\alpha = 0$)	0.69±0.17	2.01±0.40	0.26±0.01	0.47±0.03	1.23±0.07	0.66±0.03	0.80±0.05

E.2 MMDs

In Tables 17 and 18, we report the MMD values appearing in the unconditional experiment (i.e., Table 3 in the main paper) with standard deviations. These are omitted from the main paper for the sake of space.

Table 17: MMDs (1e-2) between the test set and 1,000 generated sequences on our real-world datasets. Lower is better. We report the mean ± one standard deviation over five random seeds. The lowest MMD distance on each dataset is indicated in bold, and the second lowest is indicated by an underline.

	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp-A	Yelp-M
Data	1.3	0.6	0.4	3.1	2.6	3.6	3.1
IFTPP	5.7±1.8	1.3±1.2	<u>1.9</u>±0.6	5.8±0.9	**2.9**±0.6	8.2±4.7	<u>5.1</u>±0.7
NHP	7.2±0.2	2.2±1.6	22.5±0.3	<u>5.0</u>±0.1	7.3±0.7	6.7±1.5	6.1±2.3
Diffusion	14.3±6.5	3.9±1.2	6.2±3.3	11.7±1.8	12.5±1.9	10.9±3.8	10.5±5.2
Add-and-Thin	<u>2.8</u>±0.5	<u>1.2</u>±0.27	2.7±0.3	5.2±0.6	<u>4.8</u>±0.4	**4.5**±0.2	**3.0**±0.5
EventFlow (ours)	**1.5**±0.2	**0.7**±0.1	**0.7**±0.1	**3.5**±0.1	4.9±0.7	<u>6.6</u>±1.2	**3.0**±0.5

Table 18: MMDs (1e-2) between the test set and 1,000 generated sequences on our synthetic datasets. Lower is better. We report the mean ± one standard deviation over five random seeds. The lowest MMD distance on each dataset is indicated in bold, and the second lowest is indicated by an underline.

	Hawkes1	Hawkes2	NSP	NSR	SC	SR
Data	1.3	1.3	1.8	3.0	5.7	1.1
IFTPP	**1.5**±0.4	**1.4**±0.5	**2.3**±0.7	6.2±2.1	**5.8**±0.5	**1.3**±0.3
NHP	1.9±0.3	5.2±1.6	3.6±1.3	12.6±1.8	25.4±11.5	5.0±0.7
Diffusion	4.8±2.7	5.5±3.3	10.8±7.5	15.0±3.6	9.1±1.8	5.1±2.8
Add-and-Thin	1.9±0.5	2.5±0.3	2.6±0.5	7.4±1.2	22.5±0.5	2.2±0.8
EventFlow (ours)	1.9±0.2	2.2±0.1	3.8±1.2	**4.2**±0.5	8.3±0.4	1.7±0.3

E.3 MSEs

In Table 19, we evaluate the performance of our model when forecasting only a single subsequent event. That is, given a history $\mathcal{H}$, we evaluate the MSE between the first true event time following this history and the first event time generated by each model conditioned on $\mathcal{H}$. The results are reported in Table 19. Generally, all of the methods show similar results on this metric, despite there being clear differences between methods on the multi-step task. We believe this serves to further highlight the necessity of moving beyond single-step prediction tasks.

Table 19: MSE values evaluating one-step-ahead forecasting performance. Mean values ± one standard deviation are reported over five random seeds. The lowest MSE on each dataset is indicated and bold, and the second lowest is indicated by an underline.

	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp-A	Yelp-M
IFTPP	0.85±0.05	0.32±0.03	0.0047±0.0006	0.22±0.03	1.74±0.10	1.24±0.16	1.11±0.17
NHP	0.89±0.09	0.53±0.24	**0.0022**±0.0007	0.31±0.12	2.00±0.30	1.30±0.26	1.03±0.35
Diffusion	**0.61**±0.10	0.33±0.04	0.0037±0.0012	0.23±0.14	**1.30**±0.21	**0.86**±0.18	**0.92**±0.20
Add-and-Thin	0.86±0.05	**0.30**±0.04	0.0043±0.0007	0.21±0.03	1.53±0.14	1.16±0.16	1.20±0.14
EventFlow (25 NFEs)	0.68±0.02	1.06±0.09	0.028±0.0015	**0.19**±0.01	2.30±0.22	0.90±0.05	1.30±0.03
EventFlow (10 NFEs)	0.60±0.02	1.01±0.31	0.0113±0.0016	0.16±0.01	1.93±0.07	0.77±0.05	1.14±0.07
EventFlow (1 NFE)	0.65±0.16	1.30±0.61	0.0445±0.0104	0.18±0.04	1.52±0.17	0.95±0.17	1.03±0.24

E.4 Forecasting TPPs

In Table 20, we report the sequence distance values appearing in the forecasting experiment (i.e., Figure 2 and Table 1 in the main paper) with standard deviations. We additionally report an ablation where we use the true value of n, rather than sampling $n \sim p_\phi(n \mid \mathcal{H})$. This serves to measure how much room for improvement remains from the event count predictor. In general, we see that this benchmark is still not saturated, with further gains being possible especially on datasets with long sequences (e.g., the Reddit-C and Reddit-S datasets).

Table 20: Sequence distance (12) between the forecasted and ground-truth event sequences on a held-out test set. Lower is better. We report the mean ± one standard deviation over five random seeds. The lowest mean distance on each dataset is indicated in bold, and the second lowest by an underline.

	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp-A	Yelp-M
IFTPP	4.2±0.7	25.6±2.3	61.2±3.2	5.1±0.4	2.9±0.2	2.1±0.2	3.4±0.2
NHP	2.8±0.1	31.0±1.4	95.7±0.7	4.5±0.3	3.4±0.5	1.8±0.1	3.0±0.2
Diffusion	5.4±1.2	25.7±0.9	80.3±11.4	4.6±0.7	2.4±0.2	1.8±0.1	3.3±0.7
Add-and-Thin	2.5±0.04	22.2±4.6	34.3±0.4	3.7±0.1	3.1±0.2	1.8±0.1	3.0±0.2
EventFlow (25 NFEs)	**2.0**±0.03	**15.8**±2.7	**16.0**±0.2	**3.2**±0.1	**1.4**±0.01	**1.3**±0.02	**1.9**±0.1
EventFlow (10 NFEs)	2.0±0.03	15.8±2.7	15.8±0.2	3.1±0.1	1.3±0.02	1.3±0.1	1.9±0.1
EventFlow (1 NFE)	2.0±0.01	15.8±2.7	15.8±0.2	3.2±0.2	1.4±0.03	1.8±0.3	1.9±0.1
EventFlow (25 NFEs, true n)	1.2±0.01	5.5±0.3	8.8±0.2	1.8±0.02	0.7±0.01	0.7±0.02	1.1±0.02

E.5 Runtimes

In Table 21, we report the wall-clock time required to generate 1 000 sequences for each method. As expected, NHP and IFTPP achieve the fastest generation times, though this comes at the cost of substantially lower forecast quality (see Figure 2). Both Diffusion and Add-and-Thin are relatively slower due to their iterative refinement procedures. EventFlow (with 1 NFE) attains state-of-the-art forecasting accuracy while maintaining generation times comparable to IFTPP on datasets with moderate sequence lengths (Yelp-A, Yelp-M, Taxi). On datasets with longer sequences (e.g., Reddit-S), the transformer-based architecture of EventFlow naturally incurs higher computational cost. Although EventFlow with 25 NFEs is slower, Table 1 shows that a single NFE already suffices to reach SOTA performance. We note that these runtime comparisons are meant as approximate indicators, since implementation and architectural details can substantially influence wall-clock times. Overall, EventFlow achieves competitive generation efficiency despite not being explicitly optimized for speed.

Table 21: Wall-clock runtimes (seconds) required to generate 1 000 sequences from each model, at the largest batch size which fit in memory.

	PUBG	Reddit-C	Reddit-S	Taxi	Twitter	Yelp-A	Yelp-M
IFTPP	16.15	20.77	69.14	22.62	1.85	6.55	9.60
NHP	0.99	6.13	11.94	1.19	0.83	0.74	0.89
Diffusion	43.77	99.82	262.26	78.77	43.41	29.82	32.85
Add-and-Thin	149.09	261.53	292.51	42.60	16.71	34.73	34.87
EventFlow (1 NFE)	35	739	885	8	42	6	10
EventFlow (25 NFEs)	769	17889	19624	187	960	142	233