EVENT-FORMER: A SELF-SUPERVISED LEARNING PARADIGM FOR TEMPORAL POINT PROCESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-supervision is one of the hallmarks of representation learning in the increasingly popular suite of foundation models including large language models such as BERT and GPT-3, but it has not been pursued in the context of multivariate event streams, to the best of our knowledge. We introduce a new paradigm for self-supervised learning for temporal point processes using a transformer encoder. Specifically, we design a novel pre-training strategy for the encoder where we not only mask random event epochs but also insert randomly sampled 'void' epochs where an event does not occur; this differs from the typical discrete-time pretext tasks such as word-masking in BERT but expands the effectiveness of masking to better capture continuous-time dynamics. The pre-trained model can subsequently be fine-tuned on a potentially much smaller event dataset, similar to other foundation models. We demonstrate the effectiveness of our proposed paradigm on the next-event prediction task using synthetic datasets and 3 real applications, observing a relative performance boost of as high as up to 15% compared to state-of-the art models.

1 INTRODUCTION

Transfer learning occurs when a model is pre-trained on a task, such as classification on a large labelled dataset such as ImageNet, and the model's 'knowledge' is then applied to another task, such as classification on medical images. In the current era of AI, transfer in domains such as natural language processing and image processing often leverages *self-supervised learning*, where pre-training for representation learning is done using unlabeled data. Although the fundamental ideas of transfer are not new, there is a clear emerging paradigm around foundation models (Bommasani et al., 2021), such as BERT (Devlin et al., 2018) and GTP-3 (Brown et al., 2020), which are trained with diverse unlabeled data at scale using self-supervision. These pre-trained models are then fine-tuned and adapted to different downstream tasks that respectively come with limited labeled data. Recent progress has been possible primarily due to improvements in hardware, development of the attention mechanism (Vaswani et al., 2017), and availability of substantial unlabeled training data.

We extend and pursue self-supervised learning in the context of *multivariate event streams*, i.e. data involving irregular occurrences of different types of events. Event stream datasets are widely available across domains, for instance in the form of social network interactions, customer transactions, system logs, financial events, health episodes, etc. It is well known that *temporal point processes* provide a sound mathematical framework for modeling such datasets (Daley & Jones, 2003). In this paper, we introduce a new paradigm for self-supervised learning for temporal point processes using a transformer encoder. Although self-supervised learning has recently been explored for classical time series data (Zerveas et al., 2021), to the best of our knowledge, self-supervision has not yet been explored in the context of point processes.

Neural models for temporal point processes (e.g. Du et al., 2016; Mei & Eisner, 2016; Xiao et al., 2017) have advanced the state of the art in event modeling, particularly for the task of event prediction. The typical approach in this line of work is to train a neural network on a large amount of event data. Our proposed paradigm differs from current standard practices by taking a transfer learning approach analogous to foundation models: to first pre-train a neural model on a (potentially) large event dataset and then fine-tune the model for prediction on a limited event dataset.

Transfer learning with event models has many potential applications. For instance, there may be abundant data from electronic health records containing information around a particular patient population; this data could potentially be leveraged for another population whose data is either unavailable or harder to obtain. This is an issue relevant to health equity since there may be data related concerns for some under represented populations. Similarly, financial event data from an industrial sector could potentially be transferred to another. Electronic commerce is yet another illustrative domain where transfer learning techniques may help to transfer purchase behaviors across a large pool of user populations and/or product types.

Although there is some related work around multi-task learning with event streams, such as through deploying hierarchical Gaussian process models (Lian et al., 2015) or time-scale graphical event models (Monvoisin & Leray, 2019), this line of research typically considers learning by pooling together disparate data from the same population. In contrast, we tackle the more ambitious effort of transferring from one or multiple event datasets to another. Specifically, we consider the typical setting of homogeneous transfer learning (Zhuang et al., 2021), where all datasets that get pooled for pre-training involve the same set of event types. This allows for potentially leveraging different datasets even though there may be realistic variations with respect to parametric or structural dependencies present in the corresponding event streams across each dataset.

Contributions: We make the following major contributions:

- We introduce a self-supervised paradigm for transfer learning in temporal point processes. A crucial innovation is to explicitly incorporate information about the absence of events, which improves the modeling of temporal dynamics without burdening training efficiency.
- We propose a masked event model, which is a new way to derive a pretext task for self-supervision targets in transformer models for event streams.
- Our empirical evaluation demonstrates improved transfer learning performance for event prediction on synthetic and real datasets relative to state of the art transformer event models.

2 Background and Related Work

2.1 Temporal Point Processes

Multivariate temporal point processes (MTPP) are elegant mathematical models for event streams where event types/labels from some discrete set occur in continuous time (Daley & Jones, 2003). A multi-dimensional MTPP generates sequences with timestamps and associated labels of the form $S = \{(t_i, y_i)\}_{i=1}^{n}$ where t_i is the time of occurrence of i^{th} event and y_i is its label. The cardinality of label set $\mathbb{L}$ is M. A strict temporally ordered event stream assumes a period of events observed within $[0, T]$ for each $t_i \in [0, T]$ for all $i \in [1, 2, ..., n]$. MTPPs are characterized by conditional intensity functions for each label representing the rate at which it occurs at any time t, $\lambda_e(t) = \lim_{\Delta t \to 0} \frac{\mathbb{E}(\mathcal{N}_e(t+\Delta t|h_t) - \mathcal{N}_e(t|h_t))}{\Delta t}$, where $\mathcal{N}_e(t|h_t)$ counts the number of occurrences of label e prior to historical occurrences (or simply history) h_t.

Several MTPP models such as multivariate versions of the classic Hawkes process (Hawkes, 1971) and piecewise-constant models (Gunawardana & Meek, 2016; Bhattacharjya et al., 2018) assume some parametric form of the conditional intensity function. Neural MTPPs are more recent variants that capture the underlying dynamics using neural networks. Recent years have witnessed rising popularity of neural MTPPs due to their state-of-the-art performance on benchmark datasets for predictive tasks (Du et al., 2016; Mei & Eisner, 2016; Xiao et al., 2017; Omi et al., 2019; Shchur et al., 2019; Zuo et al., 2020). A common training objective in neural MTPPs involves minimizing the negative log-likelihood. The log-likelihood of observing a sequence S is the sum of log-likelihood of events and non-events and can be computed as:

$$\log p(S) = \sum_{i=1}^{n} \sum_{e=1}^{M} \log \lambda_e(t_i) - \int_0^T \sum_{e=1}^{M} \lambda_e(t) dt \tag{1}$$

Many of these neural MTPPs assume some form of evolution dynamics between events in order to compute the second term in Eq. 1, such as recurrent neural network (RNN) evolution (Xiao et al.,

2017), exponential decay (Mei & Eisner, 2016), intensity-free modeling of the integral (Omi et al., 2019), or the usage of explicit epochs indicating absence of events (Gao et al., 2020).

2.2 TRANSFORMERS FOR EVENT DATA

Attention (Xiao et al., 2019) and transformer-based event models have shown promising results in recent years, including the self-attentive Hawkes process (SAHP) (Zhang et al., 2020), transformer Hawkes process (THP) (Zuo et al., 2020) and attentive neural point process (ANPP) (Gu, 2021). The self-attention mechanism, in our context, relates different event instances of a single stream in order to compute a representation of the stream. The architecture of transformers for MTPPs generally consists of an embedding layer and a self-attention layer. In the transformer Hawkes process (THP) (Zuo et al., 2020), for example, the embedding layer includes a time embedding and event-type embedding. Time embedding is achieved through:

$$[z(t_j)]_i = \begin{cases} \cos(t_j/10000^{\frac{i-1}{d}}) & \text{if } i \text{ is odd} \\ \sin(t_j/10000^{\frac{i}{d}}) & \text{if } i \text{ is even} \end{cases} \tag{2}$$

where t_j is a timestamp and d is the dimension of encoding. Time embedding and one-hot encoded types are combined to form the embedded input $\mathbf{X}$. For sequence $S = \{t_i, y_i\}_{i=1}^{L}$, time embedding $\mathbf{z_i}$ for each instance is specified in Eq. 2 and for the entire sequence with length L, the embedding is $\mathbf{Z} \in R^{M \times L}$. Type embedding are through the product of a trainable embedding matrix $\mathbf{U} \in \mathbb{R}^{M \times K}$ and one hot encoded vectors $\mathbf{y_i}$'s for all type instances, i.e. $\mathbf{X} = (\mathbf{UY} + \mathbf{Z})^T$ where $\mathbf{Y} = [\mathbf{y_1}, \mathbf{y_2}, ..., \mathbf{y_L}]$. $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ are query, key and value matrix; they are linear transformations of $\mathbf{X}$, i.e. $\mathbf{Q} = \mathbf{X}\mathbf{W}^Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}^K$, $\mathbf{V} = \mathbf{X}\mathbf{W}^V$ where $\mathbf{W}^Q$, $\mathbf{W}^K$, $\mathbf{W}^V$ are trainable weights. Attention output $\mathbf{C}$ is computed by the following:

$$\mathbf{C} = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{M_k}})\mathbf{V} = \mathbf{A}_s\mathbf{V} \tag{3}$$

where $\mathbf{A}_s$ denotes attention score matrix. The output $\mathbf{C}$ is then fed into a pointwise feed forward neural network (FFN) (commonly with residual connection) to learn a high level representation of the sequence for modeling conditional intensity functions.

2.3 SELF-SUPERVISION FOR SEQUENCE DATA

Sequence models such as RNNs have achieved much success in various applications, but more recent methods typically rely on transformer architectures (Vaswani et al., 2017) and the attention mechanism, especially in popular application of natural language process (NLP). With fine-tuning on the downstream tasks, these pre-training models lead to sizable improvement over previous state of the art. However, large-scale transformers are also bulky and resource-hungry, typically with billions of parameters (Brown et al., 2020) and cost millions of USD to train (Floridi & Chiriatti, 2020).

Self-supervision is typically achieved in sequence models by deriving an effective pretext task that is trained through supervised learning with a masking strategy for indicating self-supervision targets. For example, in BERT, about 15% of the words are randomly masked using an independent Bernoulli model of masking, and replaced with a new *[MASK]* label or a random word. In some recent work on self-supervision for time series data (Zerveas et al., 2021), the masking is done to ensure longer lengths of masked values, all replaced with the value 0, to get geometrically distributed run-lengths of masked values. Here we introduce a novel pretext task with a masking strategy specially tailored for asynchronous event data in continuous time. As we show later through experimental ablation studies, straightforward application of prior discrete-time sequence-based masking strategies proves inadequate for event data, because the intensity rates of events that represent continuous-time dynamics may vary in general between any two consecutively observed events. This aspect distinguishes event streams from discrete-time data such as time series and has not been addressed by masking in standard temporal transformers.

3 A SELF-SUPERVISED LEARNING PARADIGM

We introduce a self-supervised modeling paradigm with a transformer-based architecture for multivariate event streams that we refer to as *Event-former*. In particular, we present a novel pretext

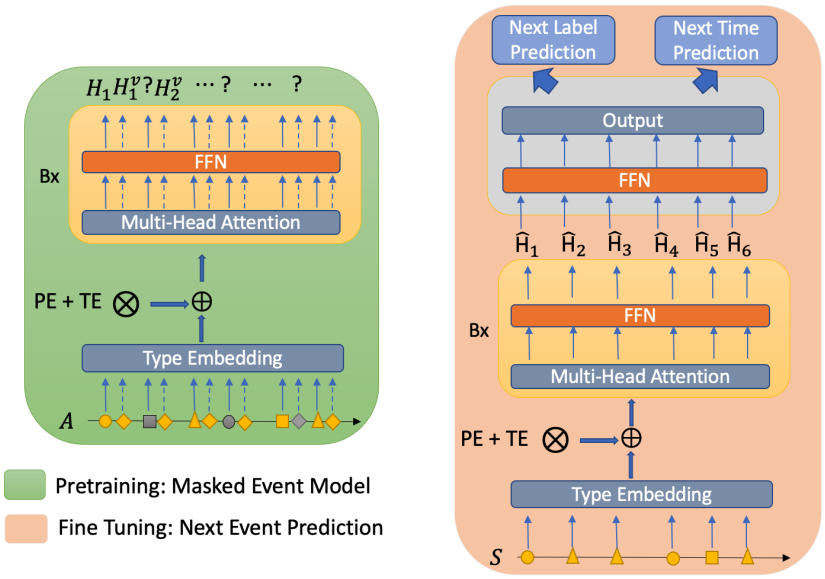

Figure 1: Pre-training and fine-tuning with Event-former.

training task specific to event data to learn a suitable representation for event streams. Such a representation can then be used by a small feed forward network for fine-tuning on a sequential next event prediction task. A high-level figurative scheme is shown in Figure 1; we provide details in the subsections to follow.

Our proposed pre-training paradigm from Figure 1 (left) involves three major aspects that distinguish it from prior work: 1) injecting void events (which are formalized in the next paragraph) to improve the representation learning of event dynamics in continuous time, 2) an effective masking strategy that uses both positional and temporal encoding on the above augmented event stream with void events, and 3) forcing the attention mechanism to adhere to the temporal order of events. We explain each aspect below before prescribing the full pre-training scheme, followed by a brief explanation of the fine-tuning procedure as depicted in Figure 1 (right).

3.1 Void Events in Transformers

Recall that an observed event stream is of the form $S = \{(t_i, y_i)\}_{i=1}^n$ where t_i and y_i are the i^{th} event's time stamp and label, respectively. We consider a modified stream where we inject a predetermined number of *void events* involving epochs where no event occurs; these are of the form (t_i', null) where 'null' is a new label signifying absence of an event occurrence. The modified stream is denoted $S' = \{t_i', y_i'\}_{i=1}^{n'}$ where $S \subset S'$ and $y_i' \in \{\mathbb{L} \cup \text{null}\}\ \forall i$. The role of the void events is to provide additional information about the dynamics of the continuous-time process by explicitly indicating that no event occurs within two consecutively observed events.

Explicitly specifying selected epochs where events do not happen has been used previously in some related work; see for instance the notion of 'fake epochs' in Gao et al. (2020), which was originally developed for RNNs and helped boost the performance of a neural point process on a model fitting task. The main idea is that in point process models, the inter-event duration between two successive events is just as important as the event epochs themselves. This is seen from the integral terms in the conditional intensity based log-likelihood expression for event streams (see Eq. 1). Just as the internal hidden state of a recurrent neural network (such as an LSTM) only changes in discrete steps upon seeing the next token, the transformer based representations too behave in a similar manner. In reality, the conditional intensity rates can evolve continuously. Introducing the void epochs in the inter-event void space provides a convenient way to force the evolution of the transformer based internal representations inside the inter-event interval, and this in turn leads to improvement in both the pre-training and the fine-tuning steps for next event prediction related tasks. We thereby ad-

dress an inherent shortcoming of transformers for event datasets in a non-parametric manner, i.e. without the need of specific parametric or process assumptions such as in the transformer Hawkes process (Zuo et al., 2020). However, to use void events in transformers requires further adaptation, particularly during the training process where masking is also used. Next we present an effective approach for masking with void event labels.

3.2 MASKING STRATEGY & INPUT ENCODING

We consider a masked event model (MEM) pre-training device that operates on the modified event stream S', where some events (t'_i, y'_i) are randomly masked for the task of prediction given history. Our masking approach is broadly similar to past work but specialized to our model. When an event epoch in the above expanded event stream S' is masked, its timestamp is replaced with the value zero and its label is replaced with the value *[MASK]*. Further, for the choice of which tokens get masked, our model admits both the independent strategy used in BERT (Devlin et al., 2018) as well as the serially correlated temporal strategy used in time series (Zerveas et al., 2021). An ablation study shown later in Table 2 indicates that either of these strategies works well when combined with the proposed MEM model, and leads to improvements through transfer learning in both MSE and accuracy for predicting the next event time and label respectively. We also note that the results are worse without the proposed MEM model's expansion of the event stream, i.e. without the injection of void event epochs. Our model differs from existing literature on masking in that both actual events and void events are admitted as candidates for masking. This combined approach leads to improved representations in experimental evaluation. The MEM model based representations are able to implicitly learn about both the event arrival rates due to masked learning with real epochs, as well as the inter-event empty spaces due to masked learning with void epochs.

In addition to the choice of masking strategy in transformer models, one also needs an encoding for the position information in the input sequence so that the uniqueness of each location is retained to some extent. Traditional positional encoding (PE) (Vaswani et al., 2017) used in transformers is not sufficient by itself for event stream data because events are associated with irregular time stamps, unlike natural language sequences. Similarly, temporal encoding (TE), such as proposed in prior work (Zuo et al., 2020), also proves inadequate by itself in our setting because our masking strategy replaces the time stamps of masked events with zero. Note that this would render indistinguishable any two distinct events (i.e. with distinct time stamps) of the same event type in the input event stream. As seen in Figure 3 in the Appendix, using TE alone leads to early plateauing of the loss function, and this is often a telltale signature of poor end-task performance. To address this issue, we propose the combined encoding strategy of using PE and TE together. We also show that the combined encoding strategy preserves the universal approximation results of standard transformers.

Theorem 1 *Transformers with combined PE and TE are universal approximators for any continuous sequence-to-sequence function with compact domain, i.e. they approximate any continuous functions f:* $\mathbf{X} \rightarrow \mathbf{H}$ *with* ϵ *error w.r.t* p*-norm where* $1 \leq p < \infty$ *and* $\mathbf{X}, \mathbf{H} \in \mathbb{R}^{d \times L}$.

Please refer to the Appendix for a proof of the above result. Yun et al. (2019) establishes that transformers with PE are universal approximators for any continuous sequence-to-sequence function with compact support (Theorem 3 in their paper) and is applicable to language sequences. The afore mentioned result however applies uniquely to event streams. More importantly, it separates two distinct event epoch encodings to (potentially) distinguish representations and establishes the predictability and learnability of a transformer model (with a certain structure) for the MEM.

3.3 TEMPORAL UPPER TRIANGULAR ATTENTION

While masked language models such as BERT leverage contextual information from both prior and post tokens of interest, here we only consider prior tokens. This is because our main task of interest is event prediction given only the past, as typical in most real-world prediction problems, which prohibits us from using post token context. We apply an upper triangular attention so that a current event epoch only attends to prior events. In MEM, any representation in pre-training as shown in Figure 1 (left) for a masked event (regardless of whether the event is observed or void) only attends to history in the past. With these pieces that define the MEM model, we next describe the pre-training and fine-tuning steps that respectively produce and exploit the self-supervised representations.

3.4 PRE-TRAINING SCHEME

Pre-training using the MEM is conducted by first randomly injecting void events into event streams, masking some of the events, and then computing a self-supervised loss determined by predicting the masked events. In this fashion, the MEM is trained to not only predict the time and label of observed events, but also try to be as accurate as possible at determining when events do not happen. In the most general setting, suppose that the sequence of time stamps for void events, denoted τ, is randomly generated from some distribution $\mathbb{P}$. A special case of this random injection is when exactly 1 void event is uniformly generated between each pair of consecutive events in S to create modified event stream S'. After randomly selecting a pre-determined percentage of events to mask, the loss for the self-supervised prediction task can be computed as:

$$\mathcal{L} = \mathbb{E}_{\tau \sim \mathbb{P}}[\mathcal{L}_{event}(\hat{t}'_m, \hat{y}'_m; t'^*_m, y'^*_m)], \tag{4}$$

where m denote the indices of the randomly selected masked events, similar conceptually to Devlin et al. (2018). The hat and star notation for t (y) refer to the model's predicted time (label) and the ground truth time (label), respectively. Note that Eq. 4 will in practice be challenging to optimize, due to the stochastic objective and additional computation complexity from sampling and inserting void events between every two consecutive events in every event stream in a batch when performing stochastic gradient descent. The time complexity for such an insertion during training is $\mathcal{O}(KL)$ where K is the number of event streams and L is the maximum length by merging the two sorted lists.

To reduce the computational cost and improve efficiency, we propose a practical solution by adopting a simpler but just as effective sampling strategy for void events. Specifically, we only sample void events once from the original dataset as an approximation and then merge as a pre-processing step. Thus no additional computing cost occurs during training. Let N_M be the total of number of masked event epochs. We use the following to measure the prediction loss for each masked event, whether it is observed or void:

$$\mathcal{L}_{event} = \frac{1}{N_M} \sum_{i}^{N_M} \text{CE}(\text{softmax}(\mathbf{H}'_m \mathbf{W}^y + \mathbf{b}^y)_{i,:}, y'^*_{m,i}) + \gamma \text{MSE}((\mathbf{H}'_m \mathbf{w}^t + b^t)_i, t'^*_{m,i}), \tag{5}$$

where $\mathbf{H}'_m$ is the masked high level representation from the transformer model of a modified stream and $\mathbf{W}^y \in \mathbb{R}^{d \times M}$, $\mathbf{b}^y \in \mathbb{R}^M$, $\mathbf{w}^t \in \mathbb{R}^d$ and $b^t \in \mathbb{R}$ are trainable weights and biases for label prediction cross entropy (CE) loss and time prediction mean square error (MSE). In addition, index i in the above equation implies a general instance of masked event epoch and $i, :$ corresponds to the i^{th} row of the output matrix. We use γ as the trade-off between the two loss terms. It is worth noting that we use only one hidden layer for masked event prediction; we avoid using deep feed forward networks to force the transformer model to learn a high quality representation $\mathbf{H}'_m$ so that it facilitates the fine-tuning process for downstream tasks.

3.5 FINE-TUNING

After pre-training, MEM can then be applied to model any new event sequence S and be further fine-tuned to obtain a better representation $\hat{\mathbf{H}}_i$. Note that during fine-tuning, we do not include void events, which simplifies the training steps and is compatible with any existing approach. Each learned representation is then fed into a small feed forward neural network for downstream tasks involving event prediction. In other words, our model fine-tunes by consuming each individual event representation $\hat{\mathbf{H}}_i$ and predicting the next label y_{i+1} as well as time t_{i+1}. The power of this approach is primarily through the conversion of sequential prediction into tabular regression and classification. For an event dataset with K event streams, each with length n_k, the loss in fine tuning step $\mathcal{L}_{pred}$ is the following:

$$\mathcal{L}_{pred} = \sum_{l=1}^{K} \sum_{i=1}^{n_k} \text{CE}(\text{softmax}(MLP(\hat{\mathbf{H}}^l_i)), y^l_{i+1}) + \alpha \text{MSE}(MLP(\hat{\mathbf{H}}^l_i), t^l_{i+1}) \tag{6}$$

where α is a similar trade-off between cross entropy and mean square error. Regression and classification share the same multi-layer neural network (MLP) for computational efficiency in our setting.

4 EXPERIMENTS

4.1 BASELINES

We use the following baselines for experiments. To focus attention on the potential benefits of self-supervision rather than the choice of neural architecture, we replace non-transformer architectures in baselines with a suitable counterpart transformer.

Recurrent Marked Temporal Point Process (Du et al., 2016) and **Event Recurrent Point Process** (Xiao et al., 2017). We replace the RNNs in both originally proposed models with transformers. We note that the original implementation [1] only predicts the very last event (t_n, y_n) given prior events; for a fair comparison, we therefore modify the code to evaluate next inter-event time d_i where $d_i = t_i - t_{i-1}$ for $i \in \{2, 3, ...n\}$.

Lognormal Mixture (Shchur et al., 2019). We replace the RNN with a transformer and take the expectation of the learned mixture model for next inter-event prediction.

Transformer Hawkes Process [2] (Zuo et al., 2020). This model is representative of the current state-of-the-art for event sequence modeling. It already involves a transformer architecture and therefore does not need any modification.

We use the following acronyms for the afore mentioned models, where the prefix 'T-' clarifies that some of these are transformer-based extensions: T-RMTPP, T-ERPP, T-LNM and THP. Following Zuo et al. (2020), we evaluate model performance on next event time prediction with root mean square error and on next event label prediction with accuracy.

4.2 SYNTHETIC EXPERIMENTS

We conduct experiments using synthetic data generated from two representative parametric families of multivariate temporal point processes: multivariate Hawkes processes (Bacry et al., 2015) and proximal graphical event models (Bhattacharjya et al., 2018). We aim to pre-train a masked event model on a set of datasets and fine-tune the model on a different dataset for the event prediction task.

Hawkes-Exp Dynamics. We generate 400 samples each from 10-dimensional Hawkes' process dynamics for 5 datasets (A, B, C, D, E) with different parameters and combine them to form a pre-training dataset. We further split each into train-dev sets 75-25 and use the dev set for hyper-parameter selection. We also generate 5 folds of a dataset F with different parameters as the target; each fold contains 500 event sequences and is further split into train-dev-test 60-20-20 subsets. Final evaluation is performed on the test subsets.

PGEM Dynamics. We generate 500 samples each from the proximal graphical event model (PGEM) Bhattacharjya et al. (2018) generator with 4 datasets (A, B, C, D) of different parameters where each contains 5 event labels. We combine these to form the pre-training dataset. Similarly, we generate an additional 5 folds of a dataset E with different parameters as the target, each of which contains 500 event sequences. Each fold is further split into train-dev-test 60-20-20 subsets, and as before, final evaluation is performed on the test subsets.

Results. As shown in Table 1, Event-former achieves best results for predicting both the next event time and event type as compared to all baselines. For the Hawkes-Exp generated data, it boosts prediction performance on average 4-5% compared to the best baseline result; on PGEM the increase is around 1-3%. The benefit of our approach along with its efficacy is its efficiency. While we only pre-train once, a typical fine-tuning procedure in this study involves a 20 fold smaller network – $\sim 1M$ trainable parameters compared to say the THP model with $\sim 20M$ trainable parameters at its recommended setting. This suggests our model learns a suitable representation of the event dynamics, specially for Hawkes process. Figure 2 shows the T-SNE projection of learned representations of event data generated by the Hawkes-Exp model datasets A, B, C, D and E onto the 2D plane. Each model generates a unique fragment segment that somewhat overlaps with another model. The linear and curvaceous segment pattern observed here is not uncommon for projecting time-series embedding onto a 2D plane (Wong & Chung, 2019). This overlapping of the representations from

[1] https://github.com/woshiyyya/ERPP-RMTPP

[2] https://github.com/SimiaoZuo/Transformer-Hawkes-Process

Table 1: Next event prediction on synthetic datasets, evaluated by RMSE (Time) and accuracy (Type). Best results (highest accuracy and lowest RMSE) are highlighted. Standard deviations are in parentheses.

Dataset	Prediction	T-RMTPP	T-ERPP	T-LNM	THP	Event-Former
Hawkes-Exp	Time	0.509(0.006)	0.654(0.004)	0.461(0.008)	0.426(0.004)	**0.411(0.003)**
	Type	0.152(0.003)	0.152(0.003)	0.172(0.004)	0.164(0.006)	**0.181(0.003)**
PGEM	Time	1.068 (0.015)	1.234(0.044)	0.861 (0.020)	0.777(0.010)	**0.771(0.014)**
	Type	0.334 (0.005)	0.208(0.005)	0.336(0.013)	0.342(0.010)	**0.352(0.004)**

pre-training data (A, B, C, D and E) with the target data F provides a visual explanation for how various datasets compare in terms of the learned representation.

4.2.1 Ablational Masking Experiments

A typical deployment of MEM involves 3 components: inserted random void epochs, random selection of masks and masking fraction. Our default setting is through the use of void events and uniformly randomly selecting 15% for masking during pre-training. We perform 3 ablation studies on the synthetic datasets: 1) **void vs. no void events**, 2) **geometric vs. random mask** and 3) **mask fraction**. Ablation 1 evaluates the effect of injected random void epochs in MEM on prediction. Clearly from Table 2, we notice a drop of type accuracy and increase of RMSE in time prediction for both cases; in particular, the predictive performance deteriorates to below the baselines on Hawkes-Exp. The injection of void epochs is justified for producing competitive results in random masking. Ablation 2 compares the impact of two masking strategies: geometric and random. We employ the former from Zerveas et al. (2021), along with inserted void epochs. Geometric masks produce slightly deteriorated results particularly on PGEM data, suggesting masking consecutive segments may not aid in learning dynamics in the continuous-time setting. Ablation 3 compares the choice of fraction of randomly masked epochs. In general, we find no significant difference between 15% and 30% for event prediction.

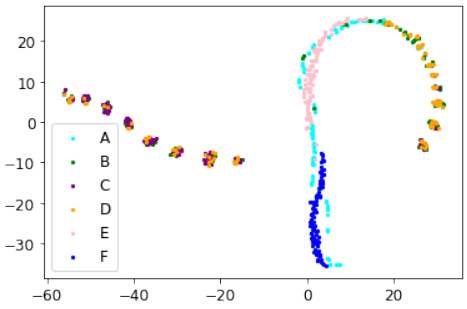

Figure 2: T-SNE projection of learned representations of Hawkes-Exp streams with pre-training on models A, B, C, D and E together.

Table 2: Ablation study on the effect of void events and masking percentage, evaluated by RMSE (Time) and accuracy (Type). Standard deviations are in parentheses.

	No-void Injection		Geometric Mask		Mask Fraction 30%	
Dataset	Time	Type	Time	Type	Time	Type
Hawkes-Exp	0.516(0.085) ↑	0.130(0.007) ↓	0.411(0.002) -	0.178(0.003) ↓	0.411(0.003) -	0.179(0.001) ↓
PGEM	0.773(0.016) ↑	0.350(0.004) ↓	0.776(0.023) ↑	0.339(0.027) ↓	0.773(0.011) ↑	0.344(0.007) ↓

4.3 Real Applications

Datasets. We perform transfer learning experiments on 3 real applications, as listed below. A descriptive summary of the 6 datasets used are shown in Table 3.

- **Financial**: *Defi-Mainnet* and *Defi-Polygon* are privately curated datasets involving user-level crypto currency transactions from the Aave website[3]. Mainnet and Polygon represent two different protocols/deployments on the platform. We test the algorithms on Mainnet after pre-training on Polygon.

[3] aave.com

- **E-Commerce**: *Electronics* and *Cosmetics* contain user-level online transactions for electronic [4] and cosmetic products [5]. We test the algorithms on Electronics after pre-training on Cosmetics.
- **Political**: *ACLED-Bangladesh* [6] and *ACLED-India* [7] are political conflicts datasets; each involves streams of conflict related actions (i.e. riots and protests) in the corresponding country. We test the algorithms on Bangladesh after pre-training on India.

Table 3: Properties of 6 real datasets.

Dataset	# classes	# seqs.	Avg. length	# events	Data Type
Defi-Mainnet	6	20539	32	654844	Financial
Defi-Polygon	6	33597	85	2856453	Financial
Electronics	4	9993	20	195726	E-Commerce
Cosmetics	4	19301	39	752109	E-Commerce
ACLED-India	6	111	17	1934	Political
ACLED-Bangladesh	4	97	17	1697	Political

Results. As demonstrated by its highest accuracy and lowest RMSE in Table 4, transfer learning with Event-former in these datasets consistently improves upon all baselines, and the improvement ranges from 3% to 15%. The most impressive improvement of 15% is on time prediction in Defi-Mainnet where using a different protocol appears to be sufficient to help with learning the dynamics. This likely suggests that the dynamics of real applications have noticeable shared similarities that the state-of-the-art transformer model approaches are unable to exploit. In addition, we observe that the transfer performance in general increases with increasing size of pre-training data. As shown from Table 3, the 3 pairs of datasets have different amounts of data; the Defi datasets have the highest number of samples, while the ACLED datasets have the lowest number. This is an indication that Event-former may be able to generalize better with more pre-training data.

Table 4: Next event prediction for 3 real applications.

Dataset	Prediction	T-RMTPP	T-ERPP	T-LNM	THP	Event-former	Improvement
Defi-Mainnet	Time	1.711	0.989	0.056	0.055	**0.047**	15% ↓
	Type	0.507	0.480	0.486	0.494	**0.565**	12% ↑
Electronics	Time	0.055	0.984	0.011	0.012	**0.010**	7% ↓
	Type	0.821	0.823	0.820	0.809	**0.887**	8% ↑
ACLED-Bangladesh	Time	2.200	0.966	0.092	0.101	**0.088**	4% ↓
	Type	0.673	0.676	0.647	0.630	**0.700**	3% ↑

5 CONCLUSION

In this work, we propose a novel self-supervised paradigm for transfer learning in multivariate temporal point processes. We introduce the usage of void events for transformer architectures, which is unique in continuous-time event models, and design a masking strategy for predicting masked event epochs and void spaces in-between. We empirically demonstrate the potential of our approach using synthetic as well as various real-world datasets. In particular, improvement of prediction performance is noticeably significant on transferring tasks over many existing competitive transformer-based approaches. While this study focuses on the homogeneous transfer setting, our approach could potentially be extended to other more complex transfer settings, such as out-of-domain heterogeneous transfer (Zhuang et al., 2021) with datasets that contain non-overlapping event labels. We leave these more complex cases to future work.

[4] https://www.kaggle.com/datasets/mkechinov/ecommerce-events-history-in-electronics-store

[5] https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop

[6] https://www.kaggle.com/datasets/saimasharleen/acled-bangladesh

[7] https://www.kaggle.com/datasets/shivkumarganesh/riots-in-india-19972022-acled-dataset-50k

REFERENCES

Emmanuel Bacry, Iacopo Mastromatteo, and Jean-François Muzy. Hawkes processes in finance. *Market Microstructure and Liquidity*, 1(01):1550005, 2015.

Debarun Bhattacharjya, Dharmashankar Subramanian, and Tian Gao. Proximal graphical event models. *Advances in Neural Information Processing Systems*, 31, 2018.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Daryl J Daley and D Vere Jones. *An Introduction to the Theory of Point Processes: Elementary Theory of Point Processes*. Springer, 2003.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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*, pp. 1555–1564, 2016.

Luciano Floridi and Massimo Chiriatti. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.

Tian Gao, Dharmashankar Subramanian, Karthikeyan Shanmugam, Debarun Bhattacharjya, and Nicholas Mattei. A multi-channel neural graphical event model with negative evidence. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3946–3953, 2020.

Yulong Gu. Attentive neural point processes for event forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7592–7600, 2021.

Asela Gunawardana and Chris Meek. Universal models of multivariate temporal point processes. In *Artificial Intelligence and Statistics*, pp. 556–563. PMLR, 2016.

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

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Wenzhao Lian, Ricardo Henao, Vinayak Rao, Joseph Lucas, and Lawrence Carin. A multitask point process predictive model. In *International Conference on Machine Learning*, pp. 2030–2038. PMLR, 2015.

Hongyuan Mei and Jason Eisner. The neural Hawkes process: A neurally self-modulating multivariate point process. *arXiv preprint arXiv:1612.09328*, 2016.

Mathilde Monvoisin and Philippe Leray. Multi-task transfer learning for timescale graphical event models. In *European Conference on Symbolic and Quantitative Approaches with Uncertainty*, pp. 313–323. Springer, 2019.

Takahiro Omi, Naonori Ueda, and Kazuyuki Aihara. Fully neural network based model for general temporal point processes. *arXiv preprint arXiv:1905.09690*, 2019.

Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. *arXiv preprint arXiv:1909.12127*, 2019.

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*, 30, 2017.

Kwan Yeung Wong and Fu-lai Chung. Visualizing time series data with temporal matching based t-sne. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 1–8. IEEE, 2019.

Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M Chu. Modeling the intensity function of point process via recurrent neural networks. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 1597–1603, 2017.

Shuai Xiao, Junchi Yan, Mehrdad Farajtabar, Le Song, Xiaokang Yang, and Hongyuan Zha. Learning time series associated event sequences with recurrent point process networks. *IEEE transactions on neural networks and learning systems*, 30(10):3124–3136, 2019.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*, 2019.

George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2114–2124, 2021.

Qiang Zhang, Aldo Lipani, Omer Kirnap, and Emine Yilmaz. Self-attentive Hawkes process. In *International Conference on Machine Learning*, pp. 11183–11193. PMLR, 2020.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1): 43–76, 2021. doi: 10.1109/JPROC.2020.3004555.

Simiao Zuo, Haoming Jiang, Zichong Li, Tuo Zhao, and Hongyuan Zha. Transformer Hawkes process. In *International Conference on Machine Learning*, pp. 11692–11702. PMLR, 2020.

A Appendix

A.1 Synthetic Generators

We generated datasets from Hawkes process and Proximal Graphical Event Model. We describe the parameters used in our experiments to generate event datasets.

Hawkes-Exp. We use a standard library to generate datasets from the Hawkes dynamics [8].The parameters are baseline rate, decay coefficent, adjacency (infectivity matrix) and end time. We describe the four for models A, B, C, D, E and F in our study.
A. baseline = [0.1097627 , 0.14303787, 0.12055268, 0.10897664, 0.08473096, 0.12917882, 0.08751744, 0.1783546 , 0.19273255, 0.0766883], decay = 2.5, infectivity = [[0.15037453, 0.10045448, 0.10789028, 0.17580114, 0.01349208, 0.01654871, 0.00384014, 0.1581418 , 0.14779747, 0.16524382], [0.1858717 , 0.1517864 , 0.08765006, 0.14824807, 0.02246419, 0.12154197, 0.02722749, 0.17942359, 0.0991161 , 0.07875789], [0.05024778, 0.14705235, 0.0866379 , 0.10796424, 0.0035688 , 0.11730922, 0.11625704, 0.11717598, 0.17924869, 0.12950002], [0.06828233, 0.08300669, 0.13250303, 0.01143879, 0.12664085, 0.12737611, 0.03995854, 0.02448733, 0.05991018, 0.0690806], [0.10829905, 0.0833048 , 0.18772458, 0.01938165, 0.03967254, 0.03063796, 0.12404668, 0.04810838, 0.0885677 , 0.04642443], [0.03019353, 0.02096386, 0.1246585 , 0.02624547, 0.03733743, 0.07003299, 0.15593352, 0.01844271, 0.1591532 , 0.01825224], [0.18546165, 0.08901222, 0.18551894, 0.11487999, 0.14041038, 0.00744305, 0.05371431, 0.02282927, 0.05624673, 0.02255028], [0.06039543, 0.07868212, 0.01218371, 0.13152315, 0.10761619, 0.05040616, 0.09938195, 0.01784238, 0.10939112, 0.1765038], [0.06050668, 0.1267631 , 0.02503273, 0.13605401, 0.0549677 , 0.03479404, 0.11139803, 0.00381908, 0.15744288, 0.00089182], [0.12873957, 0.05128336, 0.13963744, 0.18275114, 0.04724637, 0.10943116, 0.11244817, 0.10868939, 0.04237051, 0.18095826]] and end-time = 10.

B. Baseline = [8.34044009e-02, 1.44064899e-01, 2.28749635e-05, 6.04665145e-02, 2.93511782e-02, 1.84677190e-02, 3.72520423e-02, 6.91121454e-02, 7.93534948e-02, 1.07763347e-01], decay = 2.5, adjacency = [[0. , 0. , 0.03514877, 0.15096311, 0. , 0.11526461, 0.07174169, 0.09604815, 0.02413487, 0.], [0. , 0. , 0. , 0. , 0.15066599, 0.15379789, 0.01462053, 0.00671417, 0. , 0.], [0.01690747, 0.07239546, 0.16467728, 0. , 0.11894528, 0. , 0.11802102, 0.14348615, 0.00314406, 0.12896239], [0.17000181, 0. , 0. , 0. , 0. , 0. , 0. , 0.0504772 , 0.04947341, 0.], [0. , 0.11670321, 0.03638242, 0. , 0. , 0.00917392, 0. , 0.0252251 , 0.10131151, 0.1203002], [0. , 0.07118317, 0.11937903, 0.07120436, 0. , 0. , 0. , 0.08851807, 0.16239168, 0.10083865], [0. , 0.02363421, 0.02394394, 0.1388041 , 0.06836732, 0.02842716, 0.15945428, 0. , 0. , 0.12481123], [0.15185513, 0. , 0.1290996 , 0. , 0. , 0.15401787, 0. , 0.16587219, 0.11405672, 0.10687992], [0. , 0. , 0.07734744, 0.09943488, 0.07016556, 0.04074891, 0. , 0. , 0.00049346, 0.10609756], [0.05615574, 0.09061013, 0.15230831, 0.06142066, 0.15619243, 0.10716606, 0.00271994, 0.15978585, 0.11877677, 0.17145652]] and end-time = 20.

C. baseline = [0.08719898, 0.00518525, 0.1099325 , 0.08706448, 0.08407356, 0.06606696, 0.04092973, 0.12385419, 0.05993093, 0.05336546], decay = 2.5, adjacency = [[0. , 0.09623737, 0. , 0. , 0. , 0.14283231, 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.14434229, 0.10548788, 0. , 0.], [0. , 0. , 0.16178088, 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0.14554646, 0. , 0. , 0. , 0. , 0.09531432, 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.04899491], [0. , 0. , 0. , 0.07048057, 0. , 0.07546073, 0. , 0. , 0. , 0.], [0.05697369, 0. , 0. , 0. , 0. , 0. , 0.11709833, 0. , 0.03100542, 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0.04016475, 0. , 0.10768499, 0.06297179]] and end-time = 20.

D. Baseline = [0.11015958, 0.14162956, 0.05818095, 0.10216552, 0.17858939, 0.17925862, 0.02511706, 0.04144858, 0.01029344, 0.08816197], decay = [[8., 9., 2., 7., 3., 3., 2., 4., 6., 9.], [2., 9., 8., 9., 2., 1., 6., 5., 2., 6.], [5., 8., 7., 1., 1., 3., 5., 6., 9., 9.], [8., 6., 2., 2., 2., 6., 6., 8., 5., 4.], [1., 1., 1., 1., 3., 3., 8., 1., 6., 1.], [2., 5., 2., 3., 3., 5., 9., 1., 7., 1.], [5., 2., 6., 2., 9., 9., 8., 1., 1., 2.], [8., 9., 8., 5., 1., 1., 5., 4., 1., 9.], [3., 8., 3., 2., 4., 3., 5., 2., 3., 3.], [8., 4., 5., 2., 7., 8., 2., 1., 1., 6.]], adjacency = [[0. , 0.14343731, 0. , 0.11247478, 0. , 0. , 0. , 0.09416725, 0. , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0.15805746, 0. , 0.08262718], [0. , 0. , 0.03018515, 0. , 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0.11090719, 0.08158544, 0. , 0.11109462, 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0.14264684, 0. , 0. , 0.11786607, 0. , 0. , 0.01101593, 0.], [0.04954495, 0. , 0. , 0. , 0. , 0.12385743, 0. , 0. , 0.02375575, 0.05345351], [0. , 0.14941748, 0.02618691, 0. , 0.13608937, 0. , 0.06263167, 0. , 0.04097688, 0.14101171], [0. , 0.11902986, 0. , 0.04889382, 0. , 0. , 0.01569298, 0.03678315, 0. , 0.], [0.05359555, 0. , 0. , 0.09188512, 0. , 0.14255311, 0. , 0. , 0. , 0.], [0.12792415, 0.05843994, 0.16156482, 0.11931973, 0. , 0.00774966, 0.00947755, 0. , 0. , 0.]], and end-time = 10.

[8]https://x-datainitiative.github.io/tick/modules/generated/tick.hawkes.SimuHawkesExpKernels.html

E. Baseline = [0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2], decay = [[9., 4., 9., 9., 1., 6., 4., 6., 8., 7.], [1., 5., 8., 9., 2., 7., 3., 3., 2., 4.], [6., 9., 2., 9., 8., 9., 2., 1., 6., 5.], [2., 6., 5., 8., 7., 1., 1., 3., 5., 6.], [9., 9., 8., 6., 2., 2., 2., 6., 6., 8.], [5., 4., 1., 1., 1., 1., 3., 3., 8., 1.], [6., 1., 2., 5., 2., 3., 3., 5., 9., 1.], [7., 1., 5., 2., 6., 2., 9., 9., 8., 1.], [1., 2., 8., 9., 8., 5., 1., 1., 5., 4.], [1., 9., 3., 8., 3., 2., 4., 3., 5., 2.]], adjacency= [[0.02186539, 0.09356695, 0. , 0.16101355, 0.11527002, 0.09149395, 0. , 0. , 0.15672219, 0.], [0. , 0. , 0.14241135, 0. , 0.11167029, 0. , 0. , 0. , 0.0934937 , 0.], [0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.15692693, 0.], [0.08203618, 0. , 0. , 0.02996925, 0. , 0. , 0. , 0. , 0. , 0.], [0. , 0. , 0.11011391, 0.08100189, 0. , 0.11029999, 0. , 0. , 0. , 0.], [0. , 0. , 0. , 0.14162654, 0. , 0. , 0.11702301, 0. , 0. , 0.01093714], [0. , 0.04919057, 0. , 0. , 0. , 0. , 0.12297152, 0. , 0. , 0.02358583], [0.05307117, 0. , 0.14834875, 0.02599961, 0. , 0.13511597, 0. , 0.06218368, 0. , 0.04068378], [0.1400031 , 0. , 0.11817848, 0. , 0.0485441 , 0. , 0. , 0.01558073, 0.03652006, 0.], [0. , 0.05321219, 0. , 0. , 0.0912279 , 0. , 0.14153347, 0. , 0. , 0.]] and end-time = 50.

F. Baseline = [0.21736198, 0.11134775, 0.16980704, 0.33791045, 0.00188754, 0.04862765, 0.26829963, 0.3303411 , 0.05468264, 0.23003733], decay = [[5., 1., 7., 3., 5., 2., 6., 4., 5., 5.], [4., 8., 2., 2., 8., 8., 1., 3., 4., 3.], [6., 9., 2., 1., 8., 7., 3., 1., 9., 3.], [6., 2., 9., 2., 6., 5., 3., 9., 4., 6.], [1., 4., 7., 4., 5., 8., 7., 4., 1., 5.], [5., 6., 8., 7., 7., 3., 5., 3., 8., 2.], [7., 7., 1., 8., 3., 4., 6., 5., 3., 5.], [4., 8., 1., 1., 6., 7., 7., 6., 7., 5.], [8., 4., 3., 4., 9., 8., 2., 6., 4., 1.], [7., 3., 4., 5., 9., 9., 6., 3., 8., 6.]], adjacency = [[0.15693854, 0.04896059, 0.0400508 , 0. , 0. , 0.13021228, 0. , 0.10699903, 0. , 0.15329807], [0.0784283 , 0. , 0. , 0. , 0. , 0. , 0.00310706, 0.0090892 , 0.07758874, 0.], [0.01672489, 0. , 0. , 0.07851303, 0. , 0. , 0.12848331, 0.08859293, 0. , 0.], [0.09984995, 0. , 0. , 0.10541925, 0. , 0.08032527, 0. , 0. , 0. , 0.], [0.14642469, 0.06629365, 0. , 0. , 0. , 0. , 0.11891738, 0.04166225, 0.09808829, 0.17638655], [0.00976324, 0.1100343 , 0.02003261, 0. , 0. , 0.05993539, 0.09739541, 0. , 0. , 0.], [0. , 0. , 0. , 0.04672133, 0.16916 , 0. , 0.17341419, 0.12078975, 0.14441602, 0.], [0. , 0.17305542, 0.06927975, 0. , 0.03408974, 0. , 0.08457162, 0.03787486, 0.10863292, 0.], [0.07186225, 0.05760593, 0. , 0. , 0.08042031, 0.04403479, 0.1033595 , 0.17046747, 0. , 0.05083523], [0. , 0.10029222, 0. , 0.1022067 , 0. , 0. , 0.0588527 , 0. , 0.03530513, 0.]], and end-time = 40.

PGEM. We implement PGEM generator (Bhattacharjya et al., 2018) to generate 5-dimensional event datasets governed by the PGEM dynamics. The parameters are the conditional intensity (lambdas) for each event type given parental states, parental configuration (parents), windows for each parental state (windows) and end time. We describe the 5 for models A, B, C, D and E in our study. End time is 100 across all models.

A has the following parameters: 'parents': 'A': [], 'B': [], 'C': ['B'], 'D': ['A', 'B'], 'E': ['C'], 'windows': 'A': [], 'B': [], 'C': [15], 'D': [15, 30], 'E': [15], 'lambdas': 'A': (): 0.2, 'B': (): 0.05, 'C': (0,): 0.2, (1,): 0.3, 'D': (0, 0): 0.1, (0, 1): 0.05, (1, 0): 0.3, (1, 1): 0.2, 'E': (0,): 0.1, (1,): 0.3

B has the follow parameters: 'parents': 'A': ['B'], 'B': ['B'], 'C': ['B'], 'D': ['A'], 'E': ['C'], 'windows': 'A': [15], 'B': [30], 'C': [15], 'D': [30], 'E': [30], 'lambdas': 'A': (0,): 0.3, (1,): 0.2, 'B': (0,): 0.2, (1,): 0.4, 'C': (0,): 0.4, (1,): 0.1, 'D': (0,): 0.05, (1,): 0.2, 'E': (0,): 0.1, (1,): 0.3.

C has the follow parameters: 'parents': 'A': ['B', 'D'], 'B': [], 'C': ['B', 'E'], 'D': ['B'], 'E': ['B'], 'windows': 'A': [15, 30], 'B': [], 'C': [15, 30], 'D': [30], 'E': [30], 'lambdas': 'A': (0, 0): 0.1, (0, 1): 0.05, (1, 0): 0.3, (1, 1): 0.2, 'B': (): 0.2, 'C': (0, 0): 0.2, (0, 1): 0.05, (1, 0): 0.4, (1, 1): 0.3, 'D': (0,): 0.1, (1,): 0.2, 'E': (0,): 0.1, (1,): 0.4.

D has the follow parameters: 'parents': 'A': ['B'], 'B': ['C'], 'C': ['A'], 'D': ['A', 'B'], 'E': ['B', 'C'], 'windows': 'A': [15], 'B': [30], 'C': [15], 'D': [15, 30], 'E': [30, 15], 'lambdas': 'A': (0,): 0.05, (1,): 0.2, 'B': (0,): 0.1, (1,): 0.3, 'C': (0,): 0.4, (1,): 0.2, 'D': (0, 0): 0.1, (0, 1): 0.3, (1, 0): 0.05, (1, 1): 0.2, 'E': (0, 0): 0.1, (0, 1): 0.02, (1, 0): 0.4, (1, 1): 0.1.

E has the follow parameters: 'parents': 'A': ['A'], 'B': ['A', 'C'], 'C': ['C'], 'D': ['A', 'E'], 'E': ['C', 'D'], 'windows': 'A': [15], 'B': [30, 30], 'C': [15], 'D': [15, 30], 'E': [15, 30], 'lambdas': 'A': (0,): 0.1, (1,): 0.3, 'B': (0, 0): 0.01, (0, 1): 0.05, (1, 0): 0.1, (1, 1): 0.5, 'C': (0,): 0.2, (1,): 0.4, 'D': (0, 0): 0.05, (0, 1): 0.02, (1, 0): 0.2, (1, 1): 0.1, 'E': (0, 0): 0.1, (0, 1): 0.01, (1, 0): 0.3, (1, 1): 0.1.

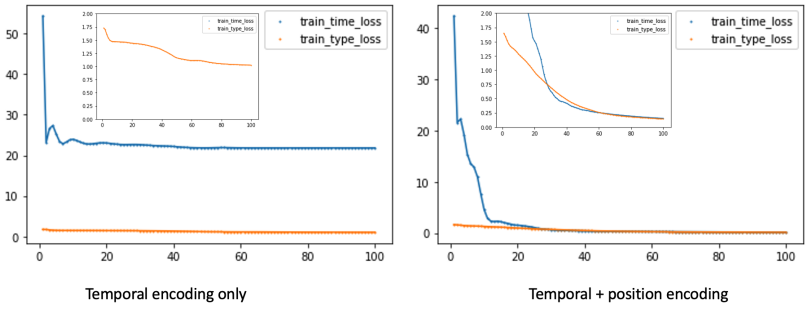

Figure 3: The effect of combined TE + PE in training.

A.2 Real Datasets

Cosmetics and **Electronics** contain user-level online transactions in an electronics and cosmetics store respectively. While the original cosmetics dataset from Kaggle contain multiple months of transactions, we used the one from Dec, 2019. Similarly, we also used electronics dataset from Dec, 2019. Both share the same four types of events: 'view', 'cart','remove-from-cart' and 'purchase. The two datasets involve transaction events in seconds. To optimize computation for transformer models, we filtered out sequences longer than 300 events and shorter than 30 and scaled the timestamps into [0,1] to avoid numerical issues.

DeFi-Mainnet and **DeFi-Polygon**. DeFi Mainnet is built on the more widely used ethereum blockchain. Polygon is a scalable sidechain of Ethereum that allows for much faster and low fee transactions than the original Ethereum Blockchain. The difference in fee structure produces quite different dynamics in the two different AAVE lending protocols. DeFi-Polygon has many more users and transactions per user, but much less total value locked than DeFi Mainnet. The polygon users are much more likely to engage in risky but potentially profitable "Yield Farming" transactions. Foundation methods would be very useful in modeling the many other lending protocols in AAVE or other lending platforms which are new or less popular and thus have less transactions. The 6 types of actions user performs are : 'borrow', 'collateral', 'deposit', 'liquidation', 'redeem', 'repay'. The origin timestamps are mined in Unix Timestamp. We filtered out sequences longer than 300 events and shorter than 30 and scaled the timestamps into [0,1] to avoid numerical issues.

ACLED-India and **ACLED-Bangladesh** contains sequences where each sequence involves an actor involving some armed conflict (i.e. riots and protests) in the respective country. The former involves events happening from 2016 to 2022, and the latter 2010-2021. The unit of each timestamp is 'days'. There are 6 types of events in the ACLED-India which are: 'Battles', 'Explosions/Remote violence', 'Riots', 'Violence against civilians','Protests',and 'Strategic developments'. The 4 overlapping types in ACLED-Bangladesh are 'Battles', 'Explosions/Remote violence', 'Riots',and 'Violence against civilians'. We filtered out sequences longer than 300 events and shorter than 2 and scaled the timestamps into [0,1] to avoid numerical issues.

A.3 Model Implementation and (Pre)Training

Pretraining. Our pretrain model adapts codes from Zuo et al. (2020) [9]. A full repo will be given upon acceptance. The procedure is fully described by Algorithm 1. We train our model via stochastic gradient descent and Adam Kingma & Ba (2014) optimizer is used for optimization. The default transformer architecture we employed are the following for pretraining: the number of blocks for multi-headed self-attention module is 4; the dimension of the value vector after attention has been applied is 512; the number attention heads is 4; the dimension of the hidden layer of the feed forward neural network 1024; the dimension of the value vector 512; the dimension of the key vector 512; dropout is 0.1. We train 100 epochs with a learning rate of 0.0001. γ is set to 1 for all experiments other than one on ACLED-india where we use 10.

[9] https://github.com/SimiaoZuo/Transformer-Hawkes-Process/tree/master/transformer

Fine-tuning. The fine tuning model consists of a feed forward network with 3 hidden layers of dimension 512. We train with Adam optimizer with learning rate of 0.001 with 100 epochs. α is set 0.01 for all experiments.

PE+TE. The combined TE + PE improves optimization when trained with only 2 samples. With TE only results in a compromised optimization (see Figure 3).

Algorithm 1: Pretraining of Event-former

Given dataset S with D sequences, each with length d_l, $\{(t_i, y_i)\}_{i=1}^{d_l}$, batch size b
Insertion void epochs:
$S' = []$
for $d \leftarrow 1$ **to** D **do**
  seq' = []
  for $i \leftarrow 1$ **to** $d_l - 1$ **do**
    t'$_i \sim Unif(t_i, t_{i+1})$
    seq'.append((t'$_i, null$))
  end
  seq_{new} = **Merge**(seq, seq')
  S'.**append**(seq_{new})
end
Masking to obtain S'_m:
for $d \leftarrow 1$ **to** D **do**
  for $i \leftarrow 1$ **to** $d'_l - 1$ **do**
    mask $\sim Bern(0.15)$
    if $mask == 1$ **then**
      (0, null) $\leftarrow (t'_i, y'_i)$
    end
  end
end
split(S'$_m$) :=S'$_{m,tr} = \{S'_{m,k}\}_{k=1}^{K}$, $S'_{m,dev}$
for $epoch \leftarrow 1$ **to** N **do**
  for $iteration \leftarrow 1$ **to** $\lceil \frac{K}{b} \rceil$ **do**
    Sample a batch of sequences B' from $S'_{m,tr}$
    compute $\mathcal{L}_{event}(B')$ (Eq. 5)
    back-propagate with gradient $\nabla_{\theta,\phi}\mathcal{L}_{event}(B')$ (θ: Transformer model parameters, ϕ: Weights and bias for regression and classification.)
    update parameters of network θ, ϕ
  end
  evaluate $L_{event}(S'_{m,dev})$, stop training if not improving in 5 epochs
end
Return: Optimal parameters ϕ^*, θ^*

A.4 Baselines Models and Implementation

T-RMTPP and **T-ERPP**. The original RMTPP and ERPP models are LSTM based. The transformer architecture we used for T-RMTPP and T-ERPP is adapted from Zuo et al. (2020). We used recommended set of parameters for training these two models.

T-LNM. The original LNM is RNN/LSTM/GRU based. The transformer architecture we used for T-LNM is adapted from Zuo et al. (2020). We used 64 mixtures of lognormal components to model that density of log (inter-event) times.

THP. We used the recommended set of parameters to train the model as it is.

A.5 PROOF OF THEOREM 1: TRANSFORMER WITH COMBINED TEMPORAL ENCODING AND POSITION ENCODING

Consider a general type of transformer architecture described by Yun et al. (2019). Our proof for a transformer with combined temporal encoding and position encoding as a universal approximating function for any continuous function for sequence-to-sequence with compact support follows similarly as proof of transformer with position encoding (Theorem 3 in Yun et al. (2019)). Without loss of generality, consider a sequence with timestamps $\{t_1, t_2, ..., t_n\}$ and let t_i be integer-valued for $i \in \{1, 2, 3..., n\}$ and $t_i < t_j$ for all $i < j$. If t_i are in decimals, we multiply by a constant to transform each given timestamp to an integer without affecting the dynamics of the event sequence. We choose a d-dimensional Temporal encoding for the n event epochs to be the following:

$$\mathbf{T} = \begin{bmatrix} 0 & t_2 - t_1 & \dots & t_n - t_1 \\ 0 & t_2 - t_1 & \dots & t_n - t_1 \\ \vdots & \vdots & & \vdots \\ 0 & t_2 - t_1 & \dots & t_n - t_1 \end{bmatrix}$$

Similarly a d-dimensional position encoding for the n event epochs to be the following:

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & \dots & n \\ 0 & 1 & \dots & n \\ \vdots & \vdots & & \vdots \\ 0 & 1 & \dots & n \end{bmatrix}$$

The combined encoding is :

$$\mathbf{PE} + \mathbf{TE} = \begin{bmatrix} 0 & t_2 - t_1 + 1 & \dots & t_n - t_1 + 1 \\ 0 & t_2 - t_1 + 1 & \dots & t_n - t_1 + 1 \\ \vdots & \vdots & & \vdots \\ 0 & t_2 - t_1 + 1 & \dots & t_n - t_1 + 1 \end{bmatrix}$$

The strict temporal order of timestamps $t_i - t_1 + 1 < t_{i+1} - t_1 + 1$for all $i \in 1, 2, 3, ..., n - 1$. This guarantees for all rows the coordinates are monotonically increasing and input values can be partitioned into cubes. The rest of the proof follows directly from the proof of Theorem 3 in Yun et al. (2019) by replacing n with t_n by performing quantization by feed-forward layers, contextual mapping by attention layers and function value mapping by feed-forward layers.