Appendices

A Discussion on Classical Schemes for Modeling Event Sequence

As shown in Figure 1, a common approach is to use sliding windows to frame the data for model training and prediction. In this setup, we discuss three classical schemes.

- *Pretrained TPP*. A straightforward solution is to pretrain a TPP in the first train set and use it for all the following test periods. Such an approach faces the problem of distribution shift, i.e., the new data is systematically different from the data the model was trained on Snoek et al. (2019). Take the Taobao dataset (Alibaba, 2018) for example, which contains time-stamped user click behaviors [4]. We split the dataset by timestamps into 10 periods sequentially and compute the 3S statistics (Shchur et al., 2021) as a measure of the distribution of event sequences for each period. Figure 7a shows that the 3S statistics differentiate between periods while Figure 7b illustrates an increasing KL divergence of the 3S statistics between the first and later period, implying a pattern shift over time. As a result, this approach may fail to adapt to new data and produce unsatisfactory predictions.
- *Retrained TPP*. Another classical solution is to train a new TPP on the data of sliding windows over again. The TPP can quickly adapt to new data but may suffer from *catastrophic forgetting* (McCloskey & Cohen, 1989): adaptation usually implies that the model loses memory of previously encountered data that may be relevant to future predictions. For example, Figure 7a shows a large overlap in distributions of different periods on the Taobao dataset, indicating the necessity of maintaining the knowledge of existing patterns to improve generalization (Snoek et al., 2019; Wang et al., 2020).
- *Online TPP*. A better solution is an online approach: discretize the time axis into small intervals and then incrementally update the TPP at the end of each interval using an online algorithm. However, online models are generally more difficult to maintain and may also cause *catastrophic forgetting* (Hoi et al., 2021). Besides, to the best of our knowledge, apart from online classical TPPs (Yang et al., 2017; Hall & Willett, 2016), the field of online neural TPPs is much less well-studied.

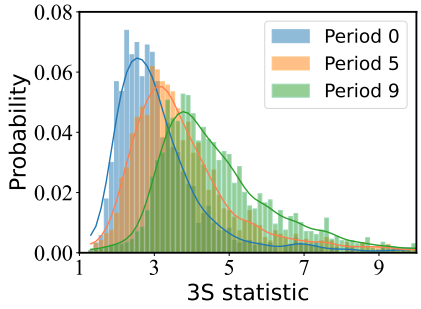
B Related Work Details

Here we draw connections and discuss differences between our method to related works.

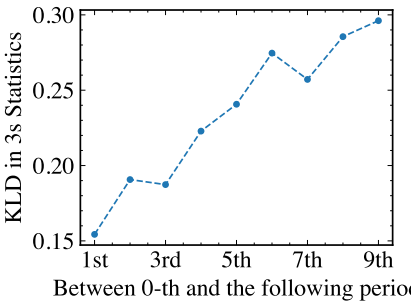
Temporal Point Process. A large variety of Neural TPPs have been proposed over recent decades, aimed at modeling event sequences with varying sorts of properties. Many of them are built on recurrent neural networks (Du et al., 2016; Mei & Eisner, 2017; Xiao et al., 2017; Omi et al., 2019; Shchur et al., 2020; Mei et al., 2020; Boyd et al., 2020). Models of this kind enjoy continuous state spaces and flexible transition functions, thus achieving superior performance on many real-world datasets, compared to classical models such as the Hawkes process Hawkes (1971). To properly capture the long-range dependency in the sequence, the attention mechanism Vaswani et al. (2017) has been adapted to TPPs Zuo et al. (2020); Zhang et al. (2020); Yang et al. (2022) to enhance the predictive performance. However, learning the event sequence under the *stream* setting is largely unexplored. To the best of our knowledge, there exist two prior works Yang et al. (2017); Hall & Willett (2016) that propose online learning algorithms for classical TPPs while that for neural TPP have rarely been studied. We show our method works better than classical online TPPs in practice (see section 4.2).

Continual Learning. There is also a rich existing literature on CL: the models can be categorized into *regularization-based* methods (Kirkpatrick et al., 2017; Zenke et al., 2017), which regularize important parameters for learned tasks, *architecture-based* methods (Rusu et al., 2016; Mallya & Lazebnik, 2018) which assign isolated parameters for each task and *rehearsal-based* methods (Cha et al., 2021; Buzzega et al., 2020) which save data from learned tasks in a rehearsal buffer to train with the current task. In retrospect, we realize a concurrent work (Dubey et al., 2022) which also augments TPP with CL abilities. Important distinctions of their work from ours include: 1. setup: they use

[4] Please see Appendix D.1 for the details of the Taobao dataset and Appendix D.2 for the explanations on 3S statistics and the procedure of the experiment on distribution shift.



(a) Distributions of 3S statistics for event sequence data in three randomly selected periods.



(b) KL divergence of distributions of 3S statistics between the 0-th and the following period.

Figure 7: Analysis of distribution shift on Taobao dataset.

standard TPP train/valid setting while we formalize more realistic streaming setting to train/validate the models; 2. methodology: they model the event streams with a hypernetwork-based regularizer while we use a trainable prompt pool with more flexibility and generality in CL. 3. task agnostic: they rely on a task descriptor built from meta attributes of events while we do not - our method is task agnostic. As their source code is not available yet, we independently implement it and our method still outperforms it (see section 4.2).

Prompt Learning. Prompt-based learning (or prompting), as an emerging transfer learning technique in NLP, applies a fixed function to condition the model so that the language model gets additional instructions to perform the downstream task. Continuous prompts have also been proposed (Lester et al., 2021; Li & Liang, 2021) to reduce prompt engineering, which directly appends a series of learnable embeddings as prompts into the input sequence, achieving outstanding performance on transfer learning. Wang et al. (2022a,b) connect prompting and CL, which attaches prompts to the pretrained backbone to learn task-invariant and task-specific instructions. Note that it is non-trivial to apply prompting to neural TPPs (See Analysis II and III in section 4.2), and our proposed novel framework reveals its values to event sequence modeling.

C Method Details

C.1 PromptTPP at Training and Test Time

The training and test time Algorithms for PromptTPP are illustrated in Algorithm 1 and Algorithm 2, respectively.

For simplicity of notations, in test time, we show how to sample the next event given one historical event sequence via the thinning algorithm (Mei & Eisner, 2017), which can be easily extended to batch-wise inference.

D Experimental Details

D.1 Dataset Details

We evaluate our methods on two industrial user behavior datasets. We provide details on the preparation and utilization of each below. For both datasets, users are associated with anonymous aliases to remove personally identifiable information (PII).

Taobao (Alibaba, 2018). This dataset contains time-stamped user click behaviors on Taobao shopping pages from November 25 to December 03, 2017. Each user has a sequence of item click events where each event contains the timestamp and the category of the item. Following the previous work (Xue et al., 2022), the categories of all items are first ranked by frequencies, and the top 19 are kept while the rest are merged into one category, with each category corresponding to an event type. We work on a subset of 4800 most active users with an average sequence length of 150 and then end up with $K = 20$ event types.

Algorithm 1 PromptTPP at training time of the $\mathcal{T}$-th task.

Input: Train set $\{s_{train}\}$. CtRetroPromptPool $(\mathcal{K}, \mathcal{V}) = \{(\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}$ (inherited from the previous task) , score function φ, loss weight α and asynchronous refresh frequency C.

Output: Trained base model with a encoder $f_{\phi_{enc}}$ and a decoder $f_{\phi_{dec}}$; trained CtRetroPromptPool $(\mathcal{K}, \mathcal{V}) = \{(\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}$.

procedure TRAIN($\{s_{train}\}, (\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}, \varphi, \alpha, C$)
  for epoch_id **in** total_epochs :
    Draw a mini batch B
    ▷ *For illustration purposes, we use batch size=1 here. The computation here can be easily extended to batch size* ≥ 1.
    for $s_{[0,T]}$ **in** B :
      Update the loss $\leftarrow$ CALCLOSS($s_{[0,T]}, f_{\phi_{enc}}, f_{\phi_{dec}}, \alpha$)
    Update $f_{\phi_{enc}}, f_{\phi_{dec}}$ by backpropagation.
    if epoch_id $\%C == 0$: Update $(\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}$ by backpropagation.
procedure CALCLOSS($s_{[0,T]}, f_{\phi_{enc}}, f_{\phi_{dec}}, \alpha$)
  $\mathcal{L} \leftarrow 0$.
  ▷ *Recursively compute the loss.*
  for $e@t$ **in** $s_{[0,T]}$:
    ▷ *Compute the likelihood loss; the technical details can be found in Mei & Eisner (2017) and Yang et al. (2022).*
    $\mathcal{L}_{event} \leftarrow$ Take a sum of log intensity at the event time by calling CALCINTENSITY($s_{[0,t)}, e@t, f_{\phi_{enc}}, f_{\phi_{dec}}, (\boldsymbol{k}_i, \boldsymbol{P}_i)_{i=1}^{M}$).
    $\mathcal{L}_{non\_event} \leftarrow$ Integrate log CALCINTENSITY($s_{[0,t)}, e@t$) over inter-event time interval.
    $\mathcal{L}_{nll} \leftarrow \mathcal{L}_{non\_event} - \mathcal{L}_{event}$
    ▷ *Compute the matching loss.*
    $\mathcal{L}_{matching} \leftarrow \sum_{\mathrm{K}_{top-N}} \varphi(f_{\phi_{enc}}(e@t), \boldsymbol{k}_{r_j})$
    $\mathcal{L} \leftarrow \mathcal{L}_{nll} + \alpha \mathcal{L}_{matching}$
  return $\mathcal{L}$
procedure CALCINTENSITY($s_{[0,t)}, e@t, f_{\phi_{enc}}, f_{\phi_{dec}}, (\boldsymbol{k}_i, \boldsymbol{P}_i)_{i=1}^{M}$)
  $s_{[0,t]} \leftarrow$ Append $e@t$ to history $s_{[0,t)}$.
  Encode $s_{[0,t]}$ by $f_{\phi_{enc}}$ to generate the hidden state $\boldsymbol{h}_t$.
  Matching the index ${r_j}_{j=1}^{N}$ based on equation 7.
  Select Top-N prompts $\{\boldsymbol{P}_{r_i}\}_{i=1}^{N}$.
  Prepend $\{\boldsymbol{P}_{r_i}\}_{i=1}^{N}$ to $\boldsymbol{h}_t$ and pass to the decode $f_{\phi_{dec}}$ to generate the intensity $\lambda_e(t), e \in \{1, ..., E\}$.
  return $\lambda_e(t), e \in \{1, ..., E\}$

DATASET	K	# EVENT TOKENS TOTAL	AVG # EVENT TOKENS PER SEQUENCE	AVG # EVENT TOKENS PER TASK	AVG # SEQUENCE PER TASK
TAOBAO	20	720,000	150	80,000	32
AMAZON	16	360,000	70	42,000	10

Table 1: Statistics of each dataset.

Amazon (Ni, 2018). This dataset includes time-stamped user product review behavior from January 2008 to October 2018. Each user has a sequence of produce review events where each event containing the timestamp and category of the reviewed product, with each category corresponding to an event type. We work on a subset of 5200 most active users with an average sequence length of 70 event tokens and then end up with $K = 16$ event types.

For the Taobao dataset, each task includes approximately 1 day of time, while for the Amazon dataset, each task includes approximately 2 years of time.

Table 1 shows statistics about each dataset mentioned above.

Algorithm 2 PromptTPP at test time of the $\mathcal{T}$-th task.

Input: An event sequence $s_{[0,T]} = \{e_i@t_i\}_{i=1}^{I}$. Trained base model with a encoder $f_{\phi_{enc}}$ and a decoder $f_{\phi_{dec}}$; trained CtRetroPromptPool $(\mathcal{K}, \mathcal{V}) = \{(\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}$ and the score function φ.

Output: Sampled next event $\widehat{e}_{I+1}@\widehat{t}_{I+1}$.

procedure DRAWNEXTEVENT($s_{[0,T]}, f_{\phi_{enc}}, f_{\phi_{dec}}$)
  $t_0 \leftarrow T$; $\mathcal{H} \leftarrow s_{[0,T]}$
  ▷ *Compute sampling intensity*
  $\{\lambda_e(t_j \mid \mathcal{H})\}_{j=1}^{N} \leftarrow$ SAMPLEINTENSITY($s_{[0,T]}, f_{\phi_{enc}}, f_{\phi_{dec}}, \{(\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}$) for all $t_j \in (t_0, \infty)$
  ▷ *Compute the upper bound λ^*.*
  ▷ *Technical details can be found in Mei & Eisner (2017)*
  find upper bound $\lambda^* \geq \sum_{e=1}^{E} \lambda_e(t_j \mid \mathcal{H})$ for all $t_j \in (t_0, \infty)$
  repeat
    draw $\Delta \sim \text{Exp}(\lambda^*)$; t_0 += Δ ▷ *time of next proposed event $\widehat{t}_{I+1}$*
    $u \sim \text{Unif}(0, 1)$
  until $u\lambda^* \leq \sum_{e=1}^{E} \lambda_e(t_0 \mid \mathcal{H})$
  draw $\widehat{e}_{I+1} \in \{1, \ldots, E\}$ where probability of e is $\propto \lambda_e(t_0 \mid \mathcal{H})$
  return $\widehat{e}_{I+1}@\widehat{t}_{I+1}$
procedure SAMPLEINTENSITY($s_{[0,T]}, f_{\phi_{enc}}, f_{\phi_{dec}}, \{(\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}$)
  Assume the last event in $s_{[0,T]}$ is $e@t$
  Generate a list of sample times $\{t_j\}_{j=1}^{N}, t_j \geq T$.
  Compute the intensity at sample times $\lambda_e t_j \leftarrow$ CALCINTENSITY($s_{[0,t]}, e@t, f_{\phi_{enc}}, f_{\phi_{dec}}, \{(\boldsymbol{k}_i, \boldsymbol{P}_i)\}_{i=1}^{M}$)
  return $\{\lambda_e(t_j \mid \mathcal{H})\}_{j=1}^{N}$

D.2 3S statistics and Experiment on Distribution Shift

We use the 3S (sum-of-squared-spacings) statistics proposed by Shchur et al. (2021) to depict the distribution of an event sequence in continuous time. Compared to the classical KS statistics (Lewis, 1965), it uniformly captures multiple properties of event sequence, such as total event count and distribution of inter-event times. Empirically, replacing the KS score with the 3S statistic consistently leads to a better separation between distributions generated by different TPPs. Please refer to the original paper (Shchur et al., 2021) for a detailed discussion.

For exploring the distribution shift in the Taobao dataset, we randomly sampled a thousand sequences of events and split them into 10 subsets by timestamps: each subset has approximately equal time horizon and is notated sequentially from 0-th to the 9-th subset. Then we follow the procedure in (Shchur et al., 2021) to compute the 3S statistics for each subset and illustrate the results in Figure 7a.

D.3 Evaluation Setup

To set up the training and evaluation process, we partition each dataset into 10 consecutively rolling slides (namely 10 tasks). For the Taobao dataset, each slide covers approximately 1 day of time, and for the Amazon dataset, each slide covers 2 years of time. The subset in each task is split into training, validation, and test sets with a 70%, 10%, 20% ratio by chronological order. Each task has no overlap in the test set. In such a setting, the total test set covers approximately 70% of data.

We train and evaluate each task sequentially. Our evaluation setup is close to that used in real applications: train the model using a fixed length of historical data and evaluate the model using the following window of the data.

D.4 Implementation Details

All models are implemented using the PyTorch framework (Paszke et al., 2017).

For the implementation of NHP, AttNHP, and thinning algorithm, we used the code from the public GitHub repository at https://github.com/yangalan123/anhp-andtt (Yang et al., 2022) with MIT License.

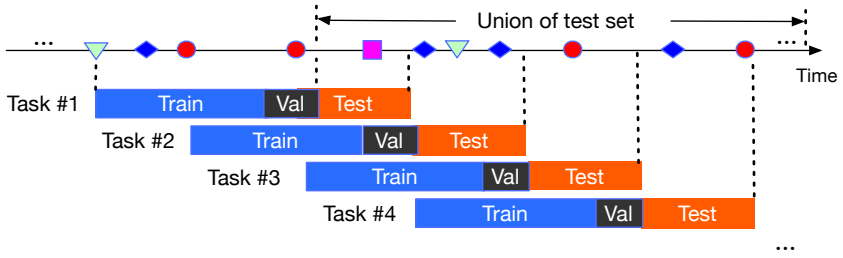


Figure 8: A demonstration of the sliding window validation method.

MODEL	# PARAMETERS	
	TAOBAO	AMAZON
PRE-NHP	23.3K	23.4K
PRE-ANHP	25.4K	25.6K
RE-NHP	23.3K	23.4K
RE-ANHP	25.4K	25.6K
O-TPP	<1K	<1K
CL-NHP	27.6K	27.7K
CL-ANHP	29.5K	29.6K
PT-NHP	26.2K	26.3K
PT-ANHP	27.8K	27.0K

Table 2: Total number of parameters for models trained on the two datasets.

For O-TPP, as the authors Yang et al. (2017) have not published the code, we implement it using the tick (Bacry et al., 2017) library.

For CL-NHP and CL-ANHP, without the public code, by following the main idea of the authors Dubey et al. (2022), we develop the hypernetwork with an MLP layer and add apply regularizer to the hypernetwork parameters while learning a new event sequence, which prevents adaptation of the hypernetworks parameters completely to the new event sequence. Note that, the base models used are NHP and ANHP, respectively, instead of FullyRNN (Omi et al., 2019) applied in the original paper.

We implemented our methods with PyTorch (Paszke et al., 2017). We submit the code to the Google Drive[5] and will release it formally upon the acceptance of the paper.

D.5 Training and Testing Details

D.5.1 Training and Hyperparameters Selection

Training base TPP model. To train the parameters for a given neural TPP, we performed early stopping based on log-likelihood on the held-out dev set.

- For NHP, the main hyperparameters to tune are the hidden dimension D of the neural network. In practice, the optimal D for a model was usually $32, 64, 128$, and we search for the optimal value among them for different datasets.
- For AttNHP, in spite of D, another important hyperparameter to tune is the number of layers L of the attention structure. In practice, the optimal L was usually $1, 2, 3, 4$. In the experiment, we choose the hyperparameter based on the held-out dev set while keeping AttNHP to have a similar size to that of NHP.

Training PromptTPP. We find α in equation 11 is not sensitive and works well in a large range, so we set $\alpha = 0.1$ consistently for both datasets. For the prompts, we set $M = 10, N = 4, L_p = 10$ for both datasets. For the asynchronous training parameter C, we choose $C = 2$ for Taobao and Amazon datasets by default.

[5]https://drive.google.com/drive/folders/103jzYhKgbhqbGB68jK7fYGmYvzapfTcw

Model	Description	Value Used	
		Taobao	Amazon
Pre-NHP	RNN Hidden Size	76	76
Pre-ANHP	Temporal Embedding	64	32
	Hidden Size	64	64
	Layer Number	3	3
Re-NHP	RNN Hidden Size	76	76
Re-ANHP	Temporal Embedding	64	32
	Hidden Size	64	64
	Layer Number	3	3
O-TPP	Kernel Size	20×20	16×16
CL-NHP	RNN Hidden Size	64	64
CL-ANHP	Temporal Embedding	64	32
	Hidden Size	64	64
	Layer Number	3	3
Pt-NHP	RNN Hidden Size	64	64
	M(Retrieval Prompt Pool Size)	10	10
	N(Top-N Selected)	4	4
	L_p(Prompt Length)	10	10
	C(Asynchronous Refresh Frequency)	2	2
Pt-ANHP	Temporal Embedding	64	32
	Hidden Size	64	64
	Layer Number	2	2
	M(Retrieval Prompt Pool Size)	10	10
	N(Top-N Selected)	4	4
	L_p(Prompt Length)	10	10
	C(Asynchronous Refresh Frequency)	2	2

Table 3: Descriptions and values of hyperparameters used for models trained on the two datasets.

Chosen Optimizer and Hyperparameters. All models are optimized using Adam (Kingma & Ba, 2015). Table 3 contains descriptions that list all of the hyperparameters set throughout our experiments.

Testing. As described in Mei & Eisner (2017), we minimized the Bayes risk via the thinning algorithm to determine decisions for what a predicted next event time $\widehat{t}_{i+1}$ and type $\widehat{e}_{i+1}$ would be after conditioning on a portion of a sequence $s_{[0,t_i]} = [e_1@t_1, \ldots, e_i@t_i]$. All experimental results are averaged over 5 runs, and the corresponding standard deviation is reported as well. In the experiment, we report the metrics on the test set for each task as well as the average metrics over all the tasks.

Integral Approximations. During training and testing, there are a number of integrals (e.g., log-likelihood in equation 11) that need to be computed, which are not feasible in closed form. Thus, we must approximate them. All integrals and expectations are approximated via Monte Carlo (MC) estimates with certain amounts of samples used. $\mathcal{L}_{non\_event}$ in equation 1 uses 100 MC samples during training and testing. When evaluating the integrals used for next event predictions in thinning algorithm, we used 100 samples where the sample points were shared across integrals for a single set of predictions in order to save on computation. The exact approximation procedure for the log-likelihood can be found in Mei & Eisner (2017).

Environment. All the experiments were conducted on a server with 256G RAM, a 64 logical cores CPU (Intel(R) Xeon(R) Platinum 8163 CPU @ 2.50GHz) and one NVIDIA Tesla P100 GPU for acceleration.

D.6 Analysis II Details: Statistical Significance

We performed the paired permutation test to validate the significance of our proposed temporal prompt. Particularly, for each model variant (Pt-NHP and Pt-ANHP), we split the test data into ten folds and collected the paired test results with temporal prompt and with the standard prompt,

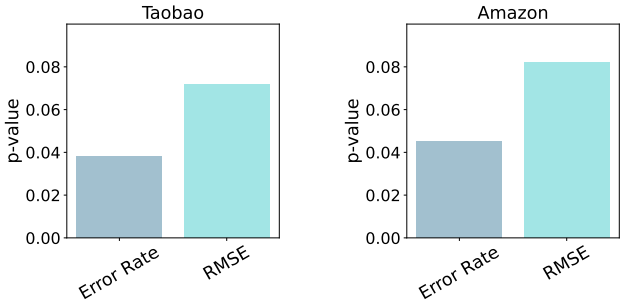
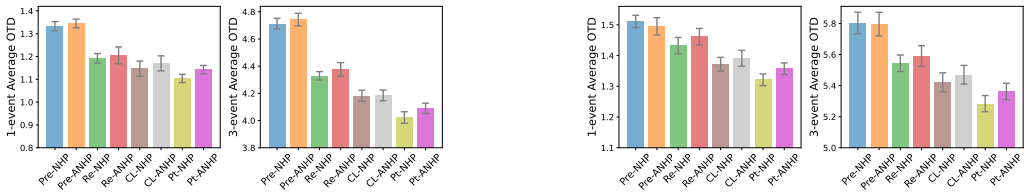


Figure 9: Statistical significance of the improvements from a temporal prompt on Taobao and Amazon datasets.



(a) Evaluation results on Task 9.

(b) Evaluation results on Task 10.

Figure 10: OTD distance comparison of generated sequence of all models.

respectively, for each fold. Then we performed the test and computed the p-value following the recipe at https://axon.cs.byu.edu/Dan/478/assignments/permutation_test.php.

The results are in Figure 9. It turns out that, on both datasets, the performance differences are strongly significant for the error rate metric (p-value < 0.05) and weakly significant for the RMSE metric (p-value ≈ 0.08).

D.7 More Result: Generation Ability Comparison

We investigated the generative ability of the models empirically on the Taobao dataset. Given a trained model, we fixed the prefix event sequence and performed the 1-event sampling and 3-event sampling (autoregressively) on the test set of task 9 and task 10. We followed (Xue et al., 2022) to compute the average optimal transport distance (OTD) to measure the distance between the generated sequence and the ground truth. Seen from Figure 10, our proposed models Pt-NHP and Pt-AttNHP achieves the best results. This is consistent with the findings in Main Results in the paper.