Unsupervised Event Outlier Detection in Continuous Time

Somjit Nath
McGill University, Mila
somjitnath@gmail.com

Yik Chau Lui
Borealis AI
yikchau.y.lui@borealisai.com

Siqi Liu
Borealis AI
siqi.x.liu@borealisai.com

Abstract

Event sequence data record the occurrences of events in continuous time. Event sequence forecasting based on temporal point processes (TPPs) has been extensively studied, but outlier or anomaly detection, especially without any supervision from humans, is still underexplored. In this work, we develop, to the best our knowledge, the first unsupervised outlier detection approach to detecting abnormal events. Our novel unsupervised outlier detection framework is based on ideas from generative adversarial networks (GANs) and reinforcement learning (RL). We train a "generator" that corrects outliers in the data with a "discriminator" that learns to discriminate the corrected data from the real data, which may contain outliers. A key insight is that if the generator made a mistake in the correction, it would generate anomalies that are different from the anomalies in the real data, so it serves as data augmentation for the discriminator learning. Different from typical GAN-based outlier detection approaches, our method employs the *generator* to detect outliers in an *online* manner. The experimental results show that our method can detect event outliers more accurately than the state-of-the-art approaches.

1 Introduction

Event sequence data are records of the occurrences of different events in continuous time, e.g., natural disasters in a country, or user actions when using an app. They can be represented as individual points on a timeline, with the location of the points indicating the time of the event occurrences. For event sequence data, forecasting and latent structure inference have been the focuses of most previous research. Methods based on Gaussian processes (e.g., [25, 16, 17, 5, 14]), Hawkes processes (e.g., [39, 11, 34, 29, 9]), and more recently deep neural networks (e.g., [6, 19, 33, 24, 38, 41, 35]), have been widely proposed and evaluated.

In contrast to forecasting, sometimes the event occurrences themselves can be unexpected, i.e. outliers. Liu & Hauskrecht [15] proposed a semi-supervised method, assuming access to clean data to train a model, to detect these outliers. Although their assumption is common in the literature, unsupervised methods without this assumption would have more practical value, since it is usually hard to get clean data without checking and preprocessing.

Inspired by Generative Adversarial Networks (GANs) [7], we propose to solve this problem by modelling a "generator" that tries to find and remove outliers and a "discriminator" that tries to distinguish the "corrected" data from the real data that can be either normal or abnormal. The key insight is that a generator can either correctly remove the outliers in the real data or incorrectly remove real points in the data. If the former is the case, it will be very difficult for the discriminator to separate the "corrected" data from the normal samples in the real data, which, by definition, constitute the majority of the data. Meanwhile, if the latter is the case, it will be relatively easy for the discriminator to separate. This intrinsic contrast between these two cases will be the source of feedback for both the generators and the discriminators to learn. Once learned, the "generators" can be used in an online

manner to detect outliers in unseen event sequences. The discriminator can be trained by standard stochastic gradient descent algorithms. However, gradient descent-based optimization cannot be used for the generator because our case is non-differentiable. There are various ways for handling the non-differentiability, such as Gumbel-softmax [8], cooperative learning [18, 10] and policy gradient methods [31, 23, 27]. We chose the latter approach in this paper, due to its flexibility.
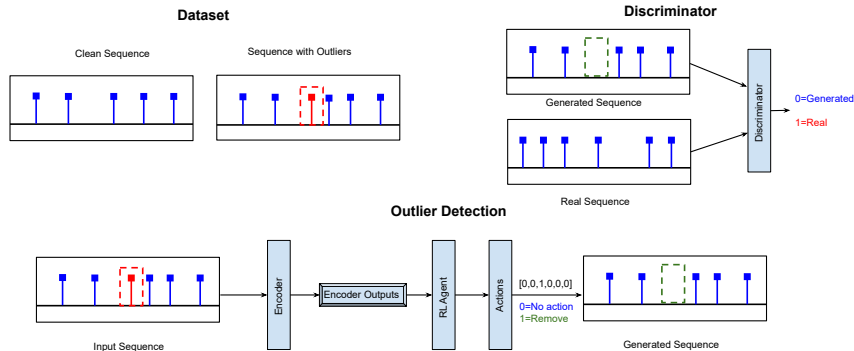
2 Unsupervised Event Outlier Detection

Figure 1: GAN + RL framework for unsupervised event outlier detection in continuous time.

2.1 Problem Formulation

An event sequence is defined as $S = \{t_n : t_n \in \mathcal{T}\}_{n=1}^N$ where t_n is the time of the occurrence of event n, N is the total number of events, and $\mathcal{T}$ denotes the entire time domain. We assume access to a dataset $\mathcal{D} = \{S_i\}_{i=1}^I$, consisting of I event sequences, where some sequences might be corrupted in the form of the addition of abnormal points. Given an event point at t_n in an event sequence S, the goal is to identify whether the event is an outlier or not, so the output would be a label, $y_n \in \{0, 1\}$, assigned to each event point.

In order to detect outliers in an unsupervised and online manner, we develop GAN-based approaches to train outlier detectors without any supervision. More specifically, we model a **generator that produces "corrected" sequences** given input sequences from the potentially corrupted data and a **discriminator that distinguishes between the "real" and "corrected" sequences**.

We sample sequences from the potentially corrupted data as "real" sequences. Since by definition, outliers are supposed to be *relatively rare* compared to normal data, we can reasonably assume that the majority of the sampled sequences are normal, and if the generator performed an incorrect "correction" on the data, e.g., removing a normal point instead of an outlier point from the sequence, then it would be possible for the discriminator to distinguish it from the sampled sequences. In the following sections, we describe our framework in detail. The final algorithm is in Appendix B.

2.2 Encoder

We use an encoder to summarize the information in the past for each sequence, as the occurrence of a new event can be influenced by the events that occurred before it. Since the event times are continuous and irregular, we use an architecture, the **continuous-time LSTMs (cLSTM)** [19], designed especially for continuous-time event sequences. We can directly use the cLSTM outputs as inputs to the generator. Still, since we wish to detect and remove outliers, information about every point in the past can be crucial for determining the action, so we apply the *attention* mechanism [4] followed by *layer normalization* [2] to the latent outputs from the cLSTM with a causal mask to ensure that there is no information leak from the future. Therefore the learned outlier detector can be applied online. This entire architecture, consisting of the cLSTM and attention layer, forms the encoder as shown in Figure 1.

2.3 Generator

For our problem, each generator is modelled as a Reinforcement Learning (RL) agent that tries to identify and remove outliers in continuous-time event sequences. Each sequence S_i, before being fed into the RL agent, is first passed through an encoder as we described in the previous section. The final encodings are treated by the RL agent as sequential states of the environment, whose goal is to identify and remove outlier points from the input sequence. Each sequence is treated as an episode, in which the agent makes decisions about making changes to the sequence. To achieve this, we use policy gradient methods, which are effective for parameterizing the optimal policy and exhibit good performance in these types of tasks [36]. For our paper, we use the clip version of Proximal Policy Optimization (PPO) algorithm [27].

The RL agent tries to identify and remove outlier points in the sequence. For each point t_n in the sequence, the RL agent takes an action ($a_n \in \{0, 1\}$): either to keep t_n ($a_n = 0$) or to remove it ($a_n = 1$). The RL agent thus learns **a policy π that defines the probability of a point being an outlier**, and if the action sampled from the policy is 1, the point is removed from the sequence. The generated sequence consists of all the points untouched by the RL agent. In this way, by hopefully keeping only the normal points, it generates a new sequence, which is fed into a discriminator to evaluate how "real" it is. The discriminator outputs a reward after the sequence is completed.

2.4 Discriminator

The goal of the discriminator is to distinguish between the generated sequences, S_j^g, obtained from the RL agent, and the real sequences, S_i, sampled from the dataset $\mathcal{D}$. As mentioned earlier, since the proportion of corrupted sequences in the dataset should be low, the majority of the samples are clean sequences. The discriminator tries to **determine whether a given sequence is "real" or "generated"** using a non-linear classifier. To prevent the discriminator from dominating over the RL agent, we also add spectral normalization [22] to each layer of the classifier model. The discriminator has the exact same architecture as the generator except without self-attention and layer normalization. However, no weights are shared between them.

3 Experiments

In the experiments, we study, despite only having access to unlabeled and potentially corrupted data, whether GAN-RL can still learn outlier detectors without supervision and beat the state-of-the-art approach originally designed for semi-supervised settings (Section 3.1). We also add interesting ablation studies and provide explanation of our model and algorithm choices in Appendix E.

Datasets To assess the performance, we use four datasets: two synthetic datasets and two real-world datasets. The synthetic datasets were generated by defining intensity functions for clean data. These intensity functions correspond to either an inhomogeneous **Poisson** process or a **Hawkes** process. For real-world datasets, we include the followings: **MIMIC**, used in prior work in event sequence modelling [6, 19], records the admission times of patients in an Intensive Care Unit over a period of 7 years. **Taxi** [30], used in [35], tracks taxi pick-up and drop-off events in the New York City.

For all the datasets, we also define a parameter β, controlling the percentage of clean sequences in the dataset. If β is 0.7, it means that 70% of the sequences are clean. For all our experiments, we use a β of 0.8 unless otherwise mentioned. Outliers are generated by sampling from a Poisson process with a constant intensity function. The specific intensity functions (Appendix C.1) and outlier generation mechanism (Appendix C.2) can be found in the Appendix.

Baseline As unsupervised event outlier detection without access to clean data has not been previously explored, we adapt the state-of-the-art approach for semi-supervised event outlier detection, PPOD, as our main baseline [15], which has demonstrated strong performance when clean data is available. It also leverages the cLSTM architecture but is trained with a negative log-likelihood loss [19] to generate the intensity functions used for scoring points and intervals in the sequences for outliers. As additional baselines, we also include RND, which generates random outlier scores, and LEN, which is based on the inter-event time interval length, from the same paper.
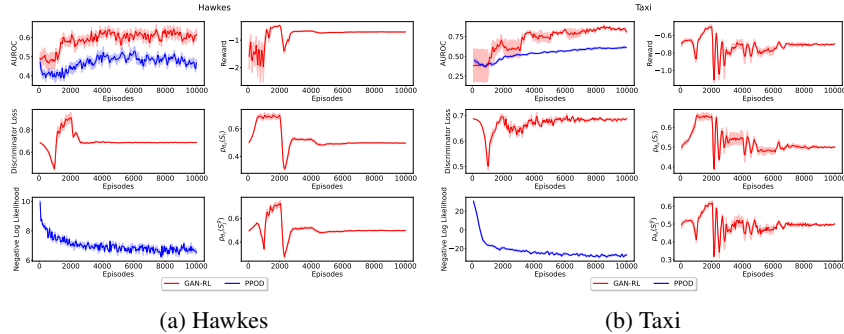
| (a) Hawkes | (b) Taxi |

Figure 2: Training performance of our algorithm on a synthetic and real dataset. We include two more in Appendix D. The curves are plotted across 10 independent runs. The shaded regions represent the standard errors.

3.1 Performance of GAN-RL

In Figure 2, we present a comprehensive analysis of our RL agents' performance over a series of 10,000 episodes, each representing a complete sequence. For the synthetic datasets, these sequences are chosen randomly from a pool of 1000 generated sequences for training. Meanwhile, for Taxi, we sample from the same training data splits as used in previous work. To gauge the effectiveness of our method against the baseline method, we employ AUROC (Area Under the Receiver Operating Characteristic Curve) scores, computed based on the ground-truth labels not accessible by any of the methods during training. This metric provides valuable insights into the RL agents' ability to distinguish outliers from the norm. Over 10 independent runs, our results consistently demonstrate that our RL agents excel at outlier detection when compared to the baseline method. Another interesting observation worth mentioning is the probability that the real sequence is classified correctly and the probability that the *generated* is classified as real seems to be around 0.5 which suggests that the RL agent has done a great job in generating real-looking data by removing outliers.

3.2 Results on Test Data

We also highlight test performance across all the datasets across 10 seeds in Table 1. These are computed over 100 test sequences unseen during training. For the real-world datasets, we simply use the test split in the dataset, and for the synthetic datasets, we generate new sequences using seeds different from the ones used in training. These results demonstrate that the improvement in the performance of GAN-RL is applicable to unseen testing data as well, and demonstrates the generalizability of the proposed method.

Table 1: Evaluation on test sets.

Dataset	RND	LEN	PPOD	GAN-RL
Poisson	0.510 ± 0.029	0.470 ± 0.023	0.55 ± 0.02	0.631 ± 0.03
Hawkes	0.503 ± 0.032	0.481 ± 0.031	0.512 ± 0.01	0.610 ± 0.03
MIMIC	0.495 ± 0.014	0.337 ± 0.024	0.583 ± 0.02	0.778 ± 0.12
Taxi	0.503 ± 0.007	0.587 ± 0.005	0.548 ± 0.01	0.647 ± 0.03

4 Conclusion

In this work, we developed a novel *unsupervised* event outlier detection framework based on ideas from GANs and RL. RL-based generators are learned to correct outliers in the unlabeled dataset through GAN-based training and then applied to unseen sequences to detect event outliers online. We evaluated our method on both synthetic and real-world datasets with simulated outliers. Compared with the state-of-the-art semi-supervised approaches, our method shows similar or better detection accuracy in all the experiments.

4

References

[1] Samet Akcay, Amir Atapour-Abarghouei, and Toby P. Breckon. GANomaly: Semi-supervised anomaly detection via adversarial training. In C. V. Jawahar, Hongdong Li, Greg Mori, and Konrad Schindler (eds.), *Computer Vision – ACCV 2018*, Lecture Notes in Computer Science, pp. 622–637, Cham, 2019. Springer International Publishing. ISBN 978-3-030-20893-6. doi: 10.1007/978-3-030-20893-6_39.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[3] E. Bacry, M. Bompaire, S. Gaïffas, and S. Poulsen. tick: a Python library for statistical learning, with a particular emphasis on time-dependent modeling. *ArXiv e-prints*, July 2017.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, May 2016.

[5] Hongyi Ding, Mohammad Khan, Issei Sato, and Masashi Sugiyama. Bayesian nonparametric Poisson-process allocation for time-sequence modeling. In *International Conference on Artificial Intelligence and Statistics*, pp. 1108–1116, 2018.

[6] 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. ACM, 2016.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[8] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[9] Minyoung Kim. Markov modulated Gaussian Cox processes for semi-stationary intensity modeling of events data. In *International Conference on Machine Learning*, pp. 2640–2648, July 2018.

[10] Sylvain Lamprier, Thomas Scialom, Antoine Chaffin, Vincent Claveau, Ewa Kijak, Jacopo Staiano, and Benjamin Piwowarski. Generative cooperative networks for natural language generation. In *International Conference on Machine Learning*, pp. 11891–11905. PMLR, 2022.

[11] Young Lee, Kar Wai Lim, and Cheng Soon Ong. Hawkes processes with stochastic excitations. In *International Conference on Machine Learning*, pp. 79–88, 2016.

[12] Dan Li, Dacheng Chen, Lei Shi, Baihong Jin, Jonathan Goh, and See-Kiong Ng. MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks. *arXiv:1901.04997 [cs, stat]*, January 2019.

[13] Shuang Li, Shuai Xiao, Shixiang Zhu, Nan Du, Yao Xie, and Le Song. Learning temporal point processes via reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 10781–10791, 2018.

[14] Siqi Liu and Milos Hauskrecht. Nonparametric regressive point processes based on conditional Gaussian processes. In *Advances in Neural Information Processing Systems*, pp. 1062–1072, 2019.

[15] Siqi Liu and Milos Hauskrecht. Event outlier detection in continuous time. In *International Conference on Machine Learning*, pp. 6793–6803. PMLR, July 2021.

[16] Chris Lloyd, Tom Gunter, Michael Osborne, and Stephen Roberts. Variational inference for Gaussian process modulated Poisson processes. In *International Conference on Machine Learning*, pp. 1814–1822, 2015.

[17] Chris Lloyd, Tom Gunter, Michael Osborne, Stephen Roberts, and Tom Nickson. Latent point process allocation. In *Artificial Intelligence and Statistics*, pp. 389–397, May 2016.

[18] Sidi Lu, Lantao Yu, Siyuan Feng, Yaoming Zhu, and Weinan Zhang. Cot: Cooperative training for generative modeling of discrete data. In *International Conference on Machine Learning*, pp. 4164–4172. PMLR, 2019.

[19] Hongyuan Mei and Jason M. Eisner. The neural Hawkes process: A neurally self-modulating multivariate point process. In *Advances in Neural Information Processing Systems*, pp. 6757–6767, 2017.

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

[21] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pp. 3481–3490. PMLR, 2018.

[22] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks, 2018.

[23] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL `http://arxiv.org/abs/1602.01783`.

[24] Takahiro Omi, Naonori Ueda, and Kazuyuki Aihara. Fully neural network based model for general temporal point processes. In *Advances in Neural Information Processing Systems*, pp. 2120–2129, 2019.

[25] Vinayak Rao and Yee W. Teh. Gaussian process modulated renewal processes. In *Advances in Neural Information Processing Systems*, pp. 2474–2482, 2011.

[26] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In Marc Niethammer, Martin Styner, Stephen Aylward, Hongtu Zhu, Ipek Oguz, Pew-Thian Yap, and Dinggang Shen (eds.), *Information Processing in Medical Imaging*, Lecture Notes in Computer Science, pp. 146–157, Cham, 2017. Springer International Publishing. ISBN 978-3-319-59050-9. doi: 10.1007/978-3-319-59050-9_12.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL `http://arxiv.org/abs/1707.06347`.

[28] Oleksandr Shchur, Ali Caner Turkmen, Tim Januschowski, Jan Gasthaus, and Stephan Günnemann. Detecting anomalous event sequences with temporal point processes. In *Advances in Neural Information Processing Systems*, volume 34, pp. 13419–13431, 2021.

[29] Yichen Wang, Bo Xie, Nan Du, and Le Song. Isotonic Hawkes processes. In *International Conference on Machine Learning*, pp. 2226–2234, 2016.

[30] C Whong. Foiling nyc's taxi trip data, 2014. URL `https://chriswhong.com/open-data/foil_nyc_taxi/`.

[31] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256, may 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL `https://doi.org/10.1007/BF00992696`.

[32] Shuai Xiao, Mehrdad Farajtabar, Xiaojing Ye, Junchi Yan, Le Song, and Hongyuan Zha. Wasserstein learning of deep generative point process models. In *Advances in Neural Information Processing Systems*, pp. 3247–3257, 2017.

[33] Shuai Xiao, Junchi Yan, Xiaokang Yang, Hongyuan Zha, and Stephen M. Chu. Modeling the intensity function of point process via recurrent neural networks. In *AAAI*, pp. 1597–1603, 2017.

[34] Hongteng Xu, Mehrdad Farajtabar, and Hongyuan Zha. Learning Granger causality for Hawkes processes. In *International Conference on Machine Learning*, pp. 1717–1726, 2016.

[35] 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.

[36] Jinsung Yoon, Sercan O. Arik, and Tomas Pfister. Data valuation using reinforcement learning, 2019.

[37] Ping Zhang, Rishabh Iyer, Ashish Tendulkar, Gaurav Aggarwal, and Abir De. Learning to select exogenous events for marked temporal point process. In *Advances in Neural Information Processing Systems*, volume 34, pp. 347–361. Curran Associates, Inc., 2021.

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

[39] Ke Zhou, Hongyuan Zha, and Le Song. Learning triggering kernels for multi-dimensional Hawkes processes. In *International Conference on Machine Learning*, pp. 1301–1309, 2013.

[40] Shixiang Zhu, Henry Shaowu Yuchi, Minghe Zhang, and Yao Xie. Sequential Adversarial Anomaly Detection for One-Class Event Data, April 2023.

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

A   Related Work

Generative model based outlier detection   Although outlier detection methods using deep generative models are not new (e.g., [1, 26, 12, 40]), none of them detect abnormal occurrences and absence of events, and almost all of them take a semi-supervised approach, assuming availability of training data consisting of samples from a single class (usually normal data but sometimes abnormal data). Note that some authors [26, 12] just use a different terminology. In contrast, our method detects event outliers in continuous time and does it in a completely unsupervised setting, where the training event sequences contain both normal and abnormal occurrences and absence of events. Once trained, the models can be used for detecting outliers online on unseen data.

Deep generative models for event sequences   Most deep learning models for event sequence data are based on combining deep architectures (such as RNNs [6, 19] and attention-based [38, 41] with TPPs, and learned by maximizing the likelihood. Different from these models, Xiao et al. [32] developed a Wasserstein distance for TPPs and a Wasserstein GAN to generate samples from the learned TPP. Li et al. [13] propose to use reinforcement learning to learn a generative model utilizing inverse reinforcement learning to learn the reward from the training data. Our work is also inspired by ideas from GANs and reinforcement learning, but our goal is outlier detection instead of sequence generation, which results in different model architectures and algorithms for learning and inference.

Outlier detection for event sequences   Liu & Hauskrecht [15] aim to detect event outliers as in our work, but their approach requires training a point process model to learn the distribution of the data, which, in theory, would require the training data to be clean without any outliers. In contrast, our method does not require clean data and can learn directly on polluted data in a completely unsupervised fashion. Zhang et al. [37] develop a greedy algorithm to separate exogenous events from an event sequence, which can be considered as a special case of unexpected event occurrences. Similarly Mei et al. [20] propose a particle-smoothing algorithm for imputing missing events in event sequences. These methods focus on offline data processing and analysis, while our work focuses on unsupervised learning of models for online outlier detection. Zhu et al. [40] propose a GAN-based approach to detecting anomalous *sequences* assuming the availability of only anomalous data, while we focus on detecting anomalous occurrences and absence of *events* assuming the unlabeled data contain both normal and abnormal events. Similarly [28] propose a new statistic for goodness-of-fit testing and detecting anomalous event *sequences* instead of events.

B   Algorithm Details

Algorithm 1 describes how the generator and discriminator are trained and what the inputs and outputs of the models look like. One additional thing to note is that, while each sequence is treated as an episode for the RL agent, the generator is trained only when an episode has been completed i.e. we always wait for the sequence to get completed before updating the generator. The same is true for the discriminator because it would need complete sequences as inputs anyway. Another crucial thing to keep in mind is that although this entire network can be trained end-to-end, we opt for an *iterative* training regime in the spirit of alternating gradient descent in the GAN literature [21]. The difference is that our generator is trained by PPO instead of gradient descent. That means for the first few episodes only the discriminator is trained, and then only the generator, and so on. This is to ensure that the training dynamics are sufficiently smooth and that both the discriminator and the generator can learn gradually. In Section B.1, we also illustrate the entire model architecture and provide intuition on the training dynamics with more details on the training framework.

Algorithm 1 GAN-RL

Require: Unlabeled Event Sequences $\mathcal{D} = \{S_i\}_{i=1}^{I}$, `Generator` ($\theta_g$), `Discriminator` ($\theta_d$), Update Frequency F, Number of Episodes K

1: **for** $k = 0 \ldots K - 1$ **do**
2:      Sample $S_j = \{t_n\}_{n=1}^{N_j} \sim \mathcal{D}$
3:      $\{\phi_n^j\}_{n=1}^{N_j} = \texttt{Encoder}_{\theta_g}(S_j)$
4:      **for** $n = 1 \ldots N_j$ **do**
5:          $a_n \leftarrow \pi_{\theta_g}(\phi_n^j)$
6:          $r_n = 0$
7:      **end for**
8:      $S_j^g = \texttt{generate}(S_j, \{a_n\}_{n=1}^{N_j})$ ▷ remove points
9:      **if** $k \mod F < F/2$ **then**
10:        Sample $S_i \sim \mathcal{D}$ and compute $p_{\theta_d}(S_i)$ and $p_{\theta_d}(S_j^g)$
11:        Update θ_d using Cross Entropy Loss
12:      **else**
13:        Compute $r_{N_j} = p_{\theta_d}(S_j^g)$
14:        Update θ_g using RL Loss
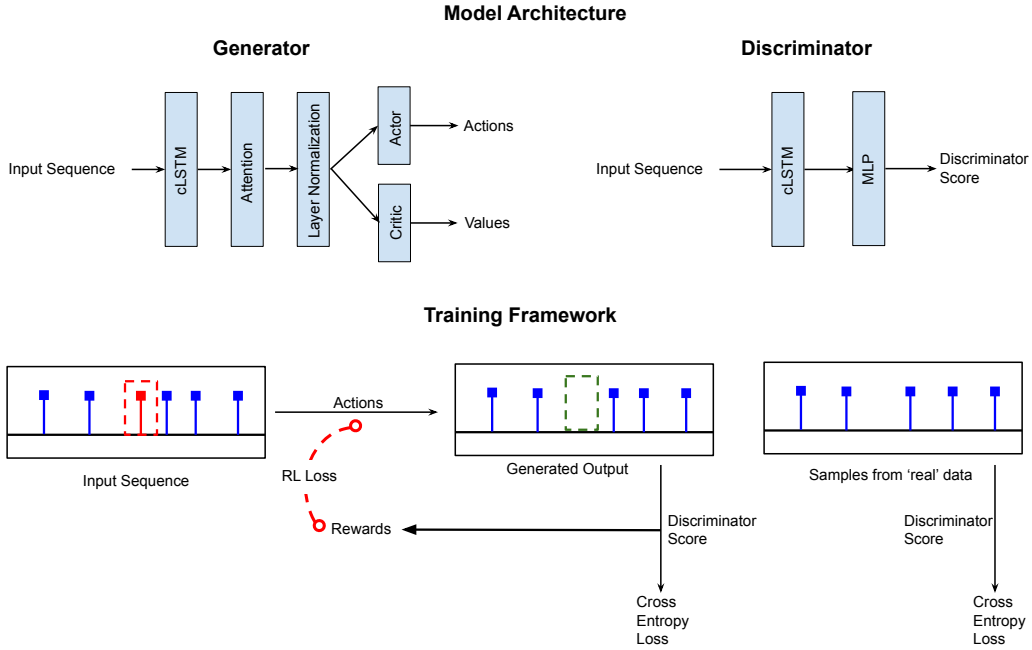15:   **end if**
16: **end for**

B.1 Training Dynamics

Figure 3: Model Architectures and Training Framework

In Figure 3, we illustrate the individual model architectures for both the generator and discriminator. Both the generator and the discriminator encode sequences with separate cLSTMs. The generator additionally has a causal attention layer, followed by layer normalization. These encodings are then fed to the actor and critic which output actions and the value functions respectively. The discriminator architecture is much simpler, and the cLSTM outputs are just fed to a neural network to output the scores for each of the individual sequences.

Figure 3 also describes the training framework and how the RL and Discriminator interact with each other. The RL agent generates new sequences from inputs by taking actions. The discriminator score

is fed as the reward to the input sequence. The discrimination score is the probability that the input sequence is a 'real' sequence, so the RL agent should try to improve this score, hence this score is an ideal candidate to be chosen as the reward. The discriminator is trained by its separate cross-entropy loss.

At the core of the training methodology for this framework lies the concept of leveraging a single scalar reward function for each sequential dataset. During discriminator training, a generated sequence has a target label 0 and a real sequence 1. In parallel, the RL agent takes on a distinct objective: to enhance the quality of the generated sequences, striving to render them as devoid of outliers as possible, thereby aligning them closely with the characteristics of clean sequences. The incentive guiding the RL agent's actions is grounded in the discriminator's output for classifying sequences as 0. This output is repurposed as a reward signal that propels the RL agent to optimize its approach. This interplay between the RL agent and the discriminator creates a dynamic wherein the generated sequences evolve to closely mimic the clean sequences, blurring the boundaries between the two gradually.

C Datasets

C.1 Intensities for Synthetic Datasets

The synthetic datasets are each characterized by their own intensity functions. For simulating these Point Processes, we use Tick [3]. The details are as follows:

- **Poisson**: The intensity function, $\lambda(t) = 1 + \sin(2t)$.
- **Hawkes**: we use a kernel with a sum of U exponential decays with intensities $\alpha = [0.01, 0.02, 0.01]$ and decays $\beta = [1.0, 3.0, 7.0]$. The intensity function is defined as:

$$\lambda(t) = \sum_{u=1}^{U} \alpha_u \beta_u \exp\left(-\beta_u t\right) 1_{t>0}$$

C.2 Outlier Simulation

We use a Poisson process with a constant intensity function (α) to generate outliers, and then we merge them with the clean sequence to create a corrupted sequence. The value of the intensity function is defined based on the type of dataset being used.

As a rule of thumb, we choose α such that the number of outliers in a sequence is around $20 - 30\%$ of the average clean sequence length. The exact values for every dataset are in Table 3.

D Additional Training Curves on Poisson and MIMIC
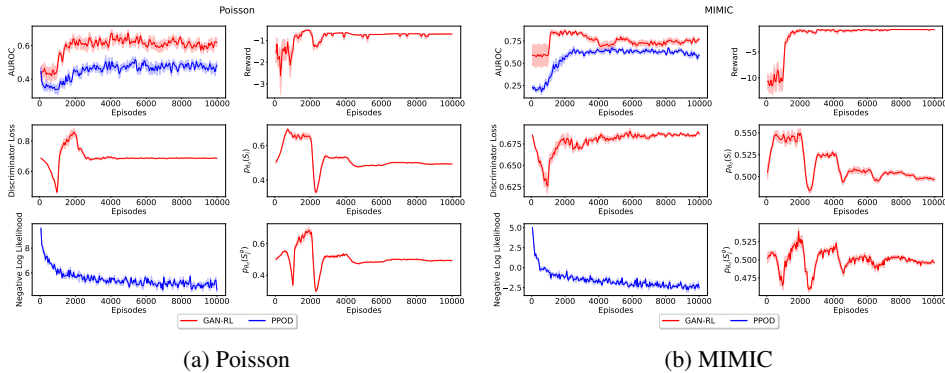

(a) Poisson   (b) MIMIC

Figure 4: Training performance of our algorithm on one synthetic and real dataset. The curves are plotted across 10 independent runs. The shaded regions represent the standard errors.

E   Ablation Studies

E.1   Sensitivity to the Amount of Corruption in the Data

The unsupervised nature of the problem setting necessitates the presence of *some* normal sequences (or sequence segments if we split complete sequences into shorter segments) in the dataset, and this usually should not be an issue as the majority of the data should be clean by definition of outliers. However, if that is not the case we expect GAN-RL to fail because the discriminator would perceive noisy sequences as real. To study the effect of the "cleanness" of the datasets, we plot in Figure 5 the average performance on 100 test sequences. The X-axis is the parameter β and increasing β means fewer corrupted sequences. As β increases, we notice an improvement in the performance of GAN-RL as the discriminator gets more real sequences and is able to distinguish them better from generated data. However, when $\beta = 0.0$, we notice that GAN-RL gets an AUROC of around 0.5 suggesting it is purely random. These results demonstrate how much we can relax the assumption of most of the data being clean. From Fig. 5, we see GAN-RL can perform well even with 60% of clean data across all the datasets.
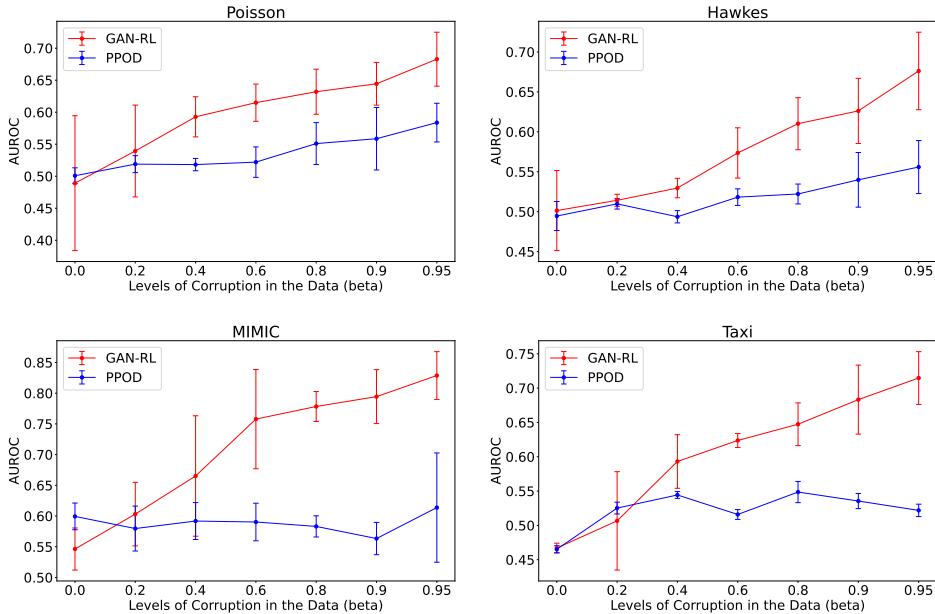

Figure 5: Sensitivity to corruption in the data.

E.2   Importance of Attention

Table 2: Comparison of asymptotic training performance with and without attention.

Dataset	GAN-RL w/o Attention	GAN-RL	PPOD	PPOD with Attention
Poisson	0.542 ± 0.02	0.651 ± 0.03	0.506 ± 0.01	0.503 ± 0.03
Hawkes	0.522 ± 0.01	0.654 ± 0.02	0.515 ± 0.04	0.523 ± 0.08
MIMIC	0.592 ± 0.05	0.601 ± 0.08	0.593 ± 0.06	0.582 ± 0.04
Taxi	0.795 ± 0.01	0.832 ± 0.02	0.690 ± 0.01	0.710 ± 0.01

The addition of the attention layer was really crucial to the good performance of GAN-RL. We believe this is because the generator needs to be able to focus on particular time points in the past before being able to make decisions on outliers and this is where adding attention can help. We run an ablation study where we remove the attention layer from the encoder and plot the training performance across

the last 10% episodes. We do notice a drop in performance of GAN-RLwithout attention in Table 2. It is interesting to note that, although removing attention results in worse performance of GAN-RL in general, the amount of decrease can vary from datasets to datasets. Importantly, even without attention, GAN-RLcan still outperform the baseline across all datasets, especially on Taxi, suggesting there are merits to our overall framework. Meanwhile, just adding an attention layer on top of cLSTM to the baseline does not improve performance much, if at all.

E.3 Utility of GANs

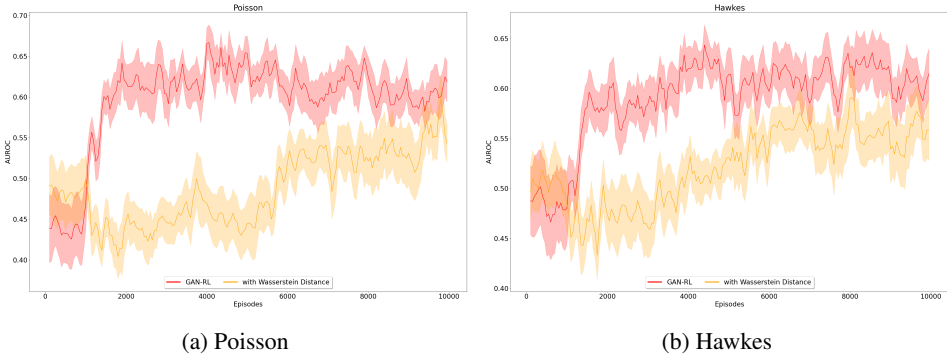|	
(a) Poisson	(b) Hawkes

Figure 6: Comparison of GAN-RL with a static reward function that operates on the Wasserstein Distance between the generated and real sequences. These results are across 10 independent runs.

We look at a simple RL agent that takes actions to generate clean sequences. Instead of using a Discriminator to evaluate the quality of the generated sequence, here, we use the Wasserstein distance between the real and generated sequence as a reward signal. So, essentially, we have a non-learnable reward signal. Figure 6 portrays GAN-RL along with new variation without the discriminator. From the difference in AUROC during training, we can conclude that having a learnable discriminator is very crucial to our method. This further justifies our decision to introduce a GAN based framework for outlier detection.

E.4 Learning End-to-End

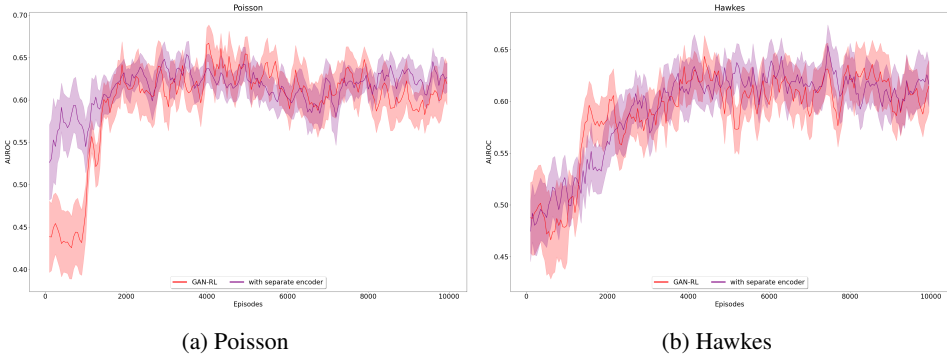|	
(a) Poisson	(b) Hawkes

Figure 7: Comparison of GAN-RL with a variant of GAN-RL that uses a separately trained encoder cLSTM that is trained on the Negative Log Likelihood loss similar to PPOD. These results across 10 independent runs.

For the second ablation, we try to evaluate if we lose any performance by training the encoder directly with the generator or discriminator losses. In Figure 7, we show a variation of GAN-RL with a separate encoder that is trained exactly like the PPOD baseline. Our method is applied on top of it, and hence the method is not end-to-end. When we try to compare it with our method, we see that we do not give up much performance when compared with the method that has a separately trained encoder.

11

F   Hyper-Parameter Sensitivity

(a) Update Frequency (b) Discriminator Learning Rate (c) Generator Learning Rate
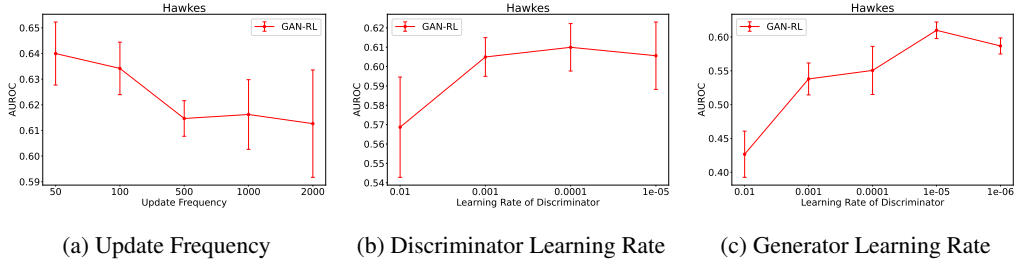
Figure 8: Sensitivity curves across 10 runs for Hawkes dataset.

In this section, we evaluate the sensitivity to the key hyper-parameters of our method. We believe there are three key hyper-parameters that dictate the learning dynamics. The first is the update frequency which is the number of episodes after which the generator or the discriminator training is switched. This is highlighted in Fig 8 (a), where we see relatively good performance across different values, suggesting our algorithm is quite robust to this hyper-parameter.

Additionally, the learning rate of the generator and the discriminator also influence the training dynamics and these are shown in Fig 8 (a) a& (b). Here, we notice a very similar trend. Higher values of the learning rate of both the generator and discriminator are quite bad, and we believe this is because of the inter-dependency of the generator and discriminator. For higher values, the discrimiantor for example can take bigger gradient steps in the wrong direction thus messing up the reward structure for the RL agent and this can lead to bad performance overall.

G Hyperparameters

For all the experiments, the hyper-parameters were tuned on final AUROC scores on the training set.

Table 3: Hyperparameters for all the experiments

Dataset	Generator	Discriminator
Maximum Time Length=10 Poisson: α=0.5 Hawkes: α=0.5 MIMIC: α=0.1 Taxi: α=0.3 Seeds: Training: $[100, ..., 109]$ Testing: $[1000]$	cLSTM hidden size=64 Self Attention Layer Layer Normalization learning rate for encoder = 0.001 Actor Arch: Linear(64, 64), Tanh(), Linear(64, 64), Tanh(), Linear(64, 2), Softmax() Critic Arch: Linear(64, 64), Tanh(), Linear(64, 64), Tanh(), Linear(64, 2), update policy every 10 sequences update policy for 10 epochs in one update clip parameter for PPO, $\epsilon = 0.2$ $c_1 = 0.5, c_2 = 0.01$ discount factor = 0.99 learning rate for actor network = 0.00001 learning rate for critic network = 0.00001	cLSTM hidden size=64 learning rate for encoder = 0.001 Arch: Linear(64, 64) Tanh(), Linear(64, 1) Linear Layers have Spectral Normalization update discriminator every 50 sequences learning rate for discriminator = 0.001 Number of Episodes=10000 Update Frequency=1000