LogGPT: Log Anomaly Detection via GPT

Xiao Han
Utah State University
Logan, UT, USA
xiao.han@usu.edu

Shuhan Yuan
Utah State University
Logan, UT, USA
Shuhan.Yuan@usu.edu

Mohamed Trabelsi
Nokia Bell Labs
Murray Hill, NJ, USA
mohamed.trabelsi@nokia-bell-labs.com

2023 IEEE International Conference on Big Data (BigData) | 979-8-3503-2445-7/23/$31.00 ©2023 IEEE | DOI: 10.1109/BigData59044.2023.10386543

***Abstract*—Detecting system anomalies based on log data is important for ensuring the security and reliability of computer systems. Recently, deep learning models have been widely used for log anomaly detection. The core idea is to model the log sequences as natural language and adopt deep sequential models, such as LSTM or Transformer, to encode the normal patterns in log sequences via language modeling. However, there is a gap between language modeling and anomaly detection as the objective of training a sequential model via a language modeling loss is not directly related to anomaly detection. To fill up the gap, we propose LogGPT, a novel framework that employs GPT for log anomaly detection. LogGPT is first trained to predict the next log entry based on the preceding sequence. To further enhance the performance of LogGPT, we propose a novel reinforcement learning strategy to finetune the model specifically for the log anomaly detection task. The experimental results on three datasets show that LogGPT significantly outperforms existing state-of-the-art approaches.**

***Index Terms*—anomaly detection, log data, generative language model**

I. Introduction

Effectively detecting abnormal events in online computer systems is critical to maintaining the security and reliability of the systems. Logs, which are a fundamental component of modern computer systems, serve as a critical source of information for system monitoring and security auditing as they record the system status, offering valuable insights into system performance and potential issues. Anomalies in log data often signify system faults, security breaches, or operational failures, making their detection a crucial task [1]–[4].

However, the task of anomaly detection in log data is challenging due to the nature of high dimensionality, large volume, and complex structure. Recently, deep learning models have emerged for log anomaly detection, such as LSTM-based models like DeepLog [1], LogAnomaly [5], and OC4Seq [6], and BERT-based models like LogBERT [2]. One commonly used strategy is to borrow the idea of language modeling in the natural language processing field to capture the sequential pattern of log data. In this paper, we call this group of log anomaly detection models **log language model**-based approaches. Particularly, the log language model is first trained to predict the next or masked log entries given the normal sequences. Then, the anomalies can be detected if the observed log entry is not in the Top-K list predicted by the log language model. The rationale is that if a log sequence follows normal patterns, the log language model should be able to predict the next or masked log entries. Therefore, when an observed log entry is not in the Top-K list predicted by the log language model, it means that the log entry has a low ratio to be in this specific position given the context, indicating the abnormality.

Although empirical studies have demonstrated the effectiveness of leveraging language models for log anomaly detection, the current models still face some limitations. The traditional LSTM-based log language models, such as DeepLog, often fail to fully capture long-term dependencies in log sequences. Therefore, the recently developed models usually adopt the Transformer structure [7] to model the long log sequences, such as LogBERT [2]. However, the masked log language model may not be able to capture the natural flow in log sequences. More importantly, there is a gap between log language modeling and anomaly detection. Technically, the log language model is usually trained to correctly predict the next log entry, while the current log anomaly detection models label the anomalies if the observed log entry is not in the Top-K list predicted by the log language model. In other words, there is a gap in the objective between the training phase and the testing phase for log anomaly detection.

Inspired by the training strategy for large language models, to fill up the gap, we introduce LogGPT, a novel framework for log anomaly detection that leverages the Generative Pre-trained Transformer (GPT) model. LogGPT still harnesses the power of generative log language models to capture the intricate patterns in log data. Specifically, LogGPT is pre-trained to predict the next log entry given the preceding sequence (prompt). More importantly, we further fine-tune LogGPT via reinforcement learning. LogGPT employs a novel reward mechanism based on whether the observed log entry is within the Top-K predicted log entries from the log language model. If the observed log entry is found within the Top-K predictions, LogGPT will receive a positive reward; otherwise, it will receive a negative reward. Reinforced by this reward signal, we expect that for the normal sequences, LogGPT can ensure the log entry is within the Top-K predictions.

The contributions of this paper are threefold. First, we propose LogGPT, a novel framework for anomaly detection in log data, which utilizes the generative log language model to capture the patterns of normal log sequences by training to predict the next log key given the previous sequence. Second, we introduce a Top-K reward metric specifically designed for fine-

tuning the log language model for anomaly detection. Third, we conduct extensive experiments to validate the effectiveness of LogGPT in detecting anomalies in log data. Experimental results demonstrate that LogGPT outperforms state-of-the-art methods, underscoring its potential as a powerful tool for anomaly detection in log data.

II. Related Work

Log anomaly detection, a critical task for ensuring system security and reliability, has received extensive research. Recently, advanced deep learning models have significantly improved the performance of log anomaly detection. In particular, Long Short-Term Memory Networks (LSTMs) have proven to be effective for log anomaly detection, such as DeepLog [1] and LogAnomaly [5]. However, a primary challenge with LSTM is that the recurrent architecture struggles to encode very long or complex sequences due to its relatively simple structure.

To address the limitations of LSTM-based models, researchers have turned to the use of Transformer [8], which is a more powerful model to capture the long-term dependencies in the sequences, such as LogBERT [2] or CAT [9]. LogBERT is a self-supervised framework that learns the patterns of normal log sequences based on BERT [8]. Specifically, LogBERT takes normal log sequences with random masks as inputs and is trained to predict the randomly masked log entries. After training, LogBERT can encode the patterns of normal log sequences. One limitation is that the masked log language model may not always capture the natural flow of log sequences in some contexts. Moreover, the performance of LogBERT is sensitive to the mask ratio, a hyperparameter controlling how many tokens will be replaced with MASK tokens during both the training and testing phases. In this work, we propose LogGPT, which leverages the GPT model to learn patterns in normal log sequences by predicting the next log entries in a sequence and further proposes a novel reinforcement learning mechanism to enhance the performance for anomaly detection.

III. Preliminary

A. Log Sequence Preprocessing

The first step of log anomaly detection is to preprocess the log messages. The major line of research in log anomaly detection is to first adopt a log parser, such as Drain [10], to extract the template from the log messages, as shown in Figure 1. Each template usually indicates one type of log message, called a log key.

After getting the log keys, the sequence of raw log messages can be transformed into a sequence of log keys. In this case, the log keys are similar to the vocabulary in natural language, while the sequence is like a sentence consisting of a sequence of log keys. Therefore, a language model can be leveraged to model the log sequences.

Formally, after preprocessing, the log messages with the same template are represented by a log key $k \in \mathcal{K}$, where $\mathcal{K}$ indicates the set of log keys extracted from the log messages. Then, a log sequence is organized as ordered log keys, denoted as $S = \{k_1, ..., k_t, ..., k_T\}$, where T indicates the length of the log sequence.

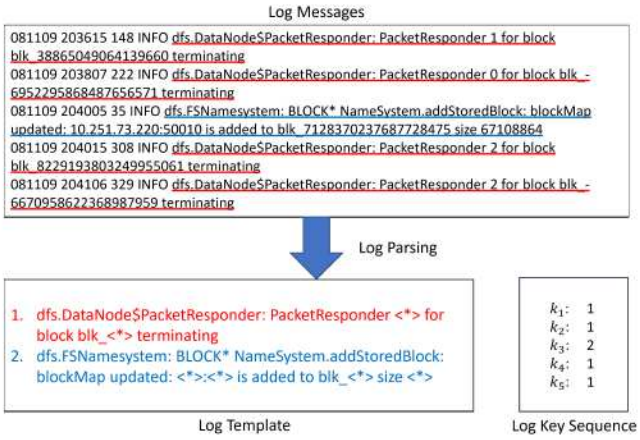


Fig. 1: Log key extraction from HDFS dataset messages via Log Parser. The message with a red/blue underscore indicates the detailed computational event for each log key separately.

B. Log Language Model

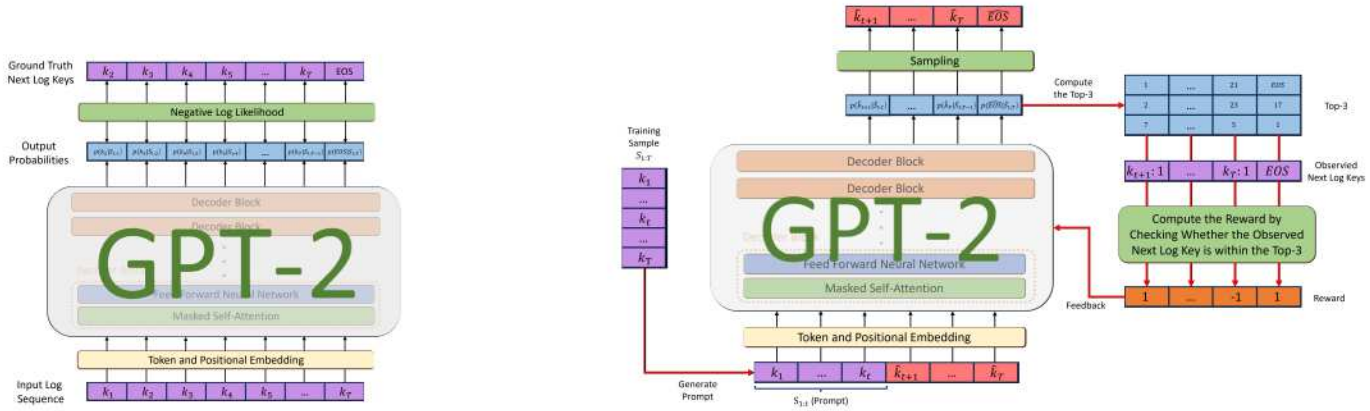
We use DeepLog [1] to illustrate the concept of the log language model. DeepLog leverages Long Short-Term Memory networks (LSTMs) for log language modeling. The primary objective of DeepLog is to learn a probabilistic model of normal execution from log data and then detect anomalies as significant deviations from normal patterns.

DeepLog is trained on $\mathcal{D} = \{S^i\}_{i=1}^N$ consisting of N normal log sequences. The LSTM in DeepLog is trained to predict the next log key in a sequence based on the preceding sequence. Formally, given a sequence of log keys $S_{1:T} = \{k_1, ..., k_t, ..., k_T\}$, where k_t indicates the log key at the t-th position. DeepLog trains an LSTM to model the conditional probability $p(k_{t+m+1}|S_{t:t+m})$ for $t = 1, 2, ..., T - m - 1$, where m indicates the window size. In other words, DeepLog adopts a sliding window with size m to split the sequences into a set of small windows and predict the next log key given the previous m log keys. The LSTM is trained to maximize the likelihood of the next log key given the preceding sequence, which can be formulated as the following objective function:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T-m-1} \log p(k^i_{t+m+1}|S^i_{t:t+m}), \quad (1)$$

where θ denotes the parameters of LSTM.

During the anomaly detection phase, given a new sequence, DeepLog still splits the sequences into small windows and employs the trained LSTM model to predict the next log key. The LSTM model predicts a probability distribution over all possible log keys in $\mathcal{K}$, ranking them based on their likelihood of being the next key in the sequence. Then, a sequence will be labeled as abnormal if the observed log key does not appear in the Top-K prediction list multiple times across all sliding windows in that sequence.



(a) Pre-training (b) Fine-tuning

Fig. 2: Framework of LogGPT.

IV. LogGPT

In this section, we introduce LogGPT, a novel log anomaly detection model based on GPT. Similar to DeepLog, LogGPT detects the log anomaly by examining whether the observed log key is in the Top-K prediction list. Because GPT is a more powerful structure compared to LSTM used by DeepLog, LogGPT does not need to further split the sequence into multiple small windows. Instead, LogGPT is trained to predict the next log key given the previous sequence, which intrinsically can capture the long-term dependence of log sequences. Moreover, besides leveraging the powerful GPT structure, we also propose a novel reinforcement learning strategy to further improve the performance of log anomaly detection.

The design of LogGPT is inspired by the training process of large language models, where the training process consists of two primary stages: pre-training and fine-tuning.

In the pre-training stage (Figure 2a), a generative log language model $f_\theta(\cdot)$ is trained on a corpus of normal log sequences $\mathcal{D}$, which allows the model to learn the underlying patterns and structures of normal system behavior. After pre-training, LogGPT is capable of generating log sequences based on a given part of the log sequences.

The fine-tuning stage (Figure 2b) is designed to further refine the model's ability to distinguish between normal and abnormal log sequences. In this stage, we employ reinforcement learning techniques to finetune the pre-trained LogGPT. Borrowing the terminology from the large language model, we define a set of prompts $\mathcal{P} = \{S^i_{1:t}\}^N_{i=1}$, where $S^i_{1:t} \subseteq S^i_{1:T}$ and $S^i_{1:T} \in \mathcal{D}$. These prompts are fed into the LogGPT to generate the following sequence $\hat{S}^i_{t:T}$ step by step. We propose a novel reward, called the Top-K metric, to fine-tune LogGPT for anomaly detection.

A. Generative Log Language Model

LogGPT utilizes GPT-2 [11] for modeling the log sequences, which is based on Transformer decoder [7]. LogGPT is trained to predict the next log key given the preceding log keys. The objective function for pretraining the LogGPT is defined as follows:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T-1}\log p(k^i_{t+1}|S^i_{1:t}), \tag{2}$$

where θ denotes the parameters of LogGPT, N is the number of log sequences and T is the length of each sequence, $p(k^i_{t+1}|S^i_{1:t})$ indicates the probability of log key at the $t+1$-th position predicted by LogGPT given the sequence $S^i_{1:t}$.

Specifically, to derive $p(k^i_{t+1}|S^i_{1:t})$, the structure of LogGPT can be defined as:

$$\mathbf{h}^i_t = \text{Transformer\_Decoder}(S^i_{1:t}) \tag{3a}$$

$$p(k^i_{t+1}|S^i_{1:t}) = \text{Softmax}(\mathbf{h}^i_t\mathbf{W}), \tag{3b}$$

where $\mathbf{h}^i_t \in \mathbb{R}^d$ indicates the hidden representation, and $\mathbf{W} \in \mathbb{R}^{d\times|\mathcal{K}|}$ is the parameter of the log language model head that maps the hidden representation to a probability distribution of all log keys in $\mathcal{K}$.

After pre-training, GPT-2 is capable of generating a log sequence $\hat{S}^i_{t+1:T} = \{\hat{k}^i_{t+1}, ..., \hat{k}^i_T\}$ based on a given part of the log sequence $S^i_{1:t}$. This capability is crucial for the subsequent fine-tuning stage, where the model is further refined to distinguish between normal and anomalous log sequences.

B. Reinforcement Learning for Log Anomaly Detection

We employ reinforcement learning to fine-tune the pre-trained GPT-2 model for log anomaly detection. In the context of our framework, we define the following elements.

State: The state, denoted as $\tilde{S}^i_{1:t} = S^i_{1:t}$, is initially defined as the given part of a log sequence. As the model generates the log sequence $\hat{S}^i_{t+1:T}$ based on the given part, the state evolves dynamically. Specifically, for each step j where $t+1 \le j \le T-1$, the state $\tilde{S}^i_{1:j}$ becomes the concatenation of the given part of the log sequence $S^i_{1:t}$ and the generated part of the log sequence $\hat{S}^i_{t+1:j}$, denoted as $\tilde{S}^i_{1:j} = \{S^i_{1:t}, \hat{S}^i_{t+1:j}\}$. The sequence $\tilde{S}^i_{1:j}$ is further transformed to a hidden representation $\tilde{\mathbf{h}}^i_j$ by the Transformer decoder shown in Equation 3a.

Action: An action is defined as sampling a log key from the K log keys with the highest probabilities predicted by LogGPT, denoted as $a^i_{j+1} \sim \text{Top-K}(p(\hat{k}^i_{j+1}|\tilde{S}^i_{1:j}))$.

Policy: A policy takes the form of LogGPT and is defined by its parameters. Specifically, given the current part of the sequence until the j-th position, the policy outputs a probability distribution over the action space, represented as $\pi_\theta(a^i_{j+1}|\tilde{\mathbf{h}}^i_j)$, where θ indicates the parameters of LogGPT.

Reward: The reward function provides feedback to the policy based on the quality of its actions. We propose a novel reward function to evaluate the predicted log key for anomaly detection, called the Top-K metric.

At each step, the Top-K metric checks whether the observed next log key is within the Top-K predicted log keys. If this is the case, the model receives a reward of 1; otherwise, it receives a reward of -1. Given a part of log sequence $S^i_{1:t}$, after an action is taken, the reward function is formulated as:

$$r_{j+1} = \begin{cases} 1, & \text{if } k^i_{j+1} \in \text{Top-K}(p(\hat{k}^i_{j+1}|\tilde{S}^i_{1:j})) \\ -1, & \text{if } k^i_{j+1} \notin \text{Top-K}(p(\hat{k}^i_{j+1}|\tilde{S}^i_{1:j})) \end{cases}. \quad (4)$$

Here, k^i_{j+1} refers to the actual next log key, and $p(\hat{k}^i_{j+1}|\tilde{S}^i_{1:j})$ denotes the probability distribution predicted by LogGPT over the action space given the current state.

The Top-K metric promotes better generalization and robustness of LogGPT in anomaly detection. By encouraging the model to predict a set of likely next log keys rather than a single most likely log key, the Top-K metric helps LogGPT learn a more nuanced representation of the normal log patterns. This approach recognizes that log data may contain inherent variability even for the normal log sequences, and a broader range of acceptable candidates can still reflect normal system behavior. The Top-K metric, therefore, enhances the precision of anomaly detection by aligning the model's predictions with the complex nature of log data.

C. Policy Update

We adopt Proximal Policy Optimization (PPO) [12] for the policy update. PPO is a type of policy gradient method that optimizes the policy directly by maximizing the expected reward and can further maintain the stability of the learning process and prevent harmful updates. The objective function of PPO is defined as follows:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{i=1}^{N}\sum_{j=t}^{T-1}\frac{\pi_\theta(a^i_{j+1}|\mathbf{h}^i_j)}{\pi_{\theta_{\text{old}}}(a^i_{j+1}|\mathbf{h}^i_j)}r_{j+1}\right], \quad (5)$$

where π_θ is the new policy, $\pi_{\theta_{\text{old}}}$ is the old policy, and r_{j+1} is the reward for an action.

The policy π_θ is updated by performing gradient ascent on the objective function $J(\theta)$:

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta), \quad (6)$$

where α is the learning rate.

The policy update process is repeated for a number of iterations until the policy converges or a maximum number of iterations is reached. The Top-K metric encourages the model to recognize the inherent variability in normal log data by rewarding predictions that include the actual next log key within a broader set.

TABLE I: Statistics of the Datasets. The number in the parentheses indicates the unique log keys in the training set.

Dataset	# of Unique Log Keys	# of Log Sequences	Avg. Seq. Length	Training Data	Testing Data	
					Normal	Anomalous
HDFS	48 (15)	575,061	19	5,000	553,223	16,838
BGL	396 (160)	36,927	58	5,000	28,631	3,296
Thunderbird	7,703 (904)	112,959	166	5,000	67,039	40,920

D. Anomaly Detection

After fine-tuning, LogGPT is deployed to detect abnormal log sequences. Given a new log sequence $S_{1:T}$, LogGPT iteratively predicts the next log key k_{t+1} given the preceding subsequence $S_{1:t}$ for $1 \le t \le T-1$.

For each predicted log key, the model generates a set of Top-K predicted log keys. This set represents the K most likely log keys at the current position. The actual next log key is then compared to this set. As long as one actual log key is not in the set of Top-K predicted log keys, the whole log sequence will be flagged as anomalous.

V. Experiments

A. Experimental Setup

Datasets. We evaluate LogGPT on three system log datasets: 1) **HDFS** (Hadoop Distributed File System) [13] consists of 575,061 log sequences, out of which 16,838 have been labeled as anomalous; 2) **BGL** (BlueGene/L Supercomputer System) [14] contains 36,927 log sequences, with 3,296 of them classified as anomalous; 3) **Thunderbird** [14] 112,959 log sequences, with 40,920 of them marked as anomalous.

Table I shows the statistics of three datasets. For all the datasets, we randomly select 5000 normal log sequences as the training dataset.

Baselines. We compare LogGPT with a variety of baseline methods, consisting of both traditional machine learning models and deep learning models: 1) *PCA* (Principal Component Analysis) [15], which reduces the dimension of a counting matrix based on the frequency of log key sequences to identify anomalies; 2) *iForest* (Isolation Forest) [16], which detects anomalies as instances with short average path lengths on the constructed isolation trees; 3) *OCSVM* (One-Class Support Vector Machine) [17], which is a variant of the Support Vector Machine algorithm that is designed for anomaly detection tasks; 4) *LogCluster* [18], which detects anomalies that are not in any clusters; 5) *DeepLog* [1], which is a log language model-based approach for anomaly detection; 6) *LogAnomaly* [5], which can detect both sequential and quantitative log anomalies simultaneously; 7) *OC4Seq* (Multi-Scale One-Class Recurrent Neural Networks) [6], which detect anomalies based on the idea of the deep one-class classification model; 8) *LogBERT* [2], which is another log language model-based approach based on BERT architecture; 9) *CAT* (Content-Aware Transformer) [9], which is a self-attentive encoder-decoder

TABLE II: Experimental Results on HDFS, BGL, and Thunderbird Datasets.

Method	HDFS			BGL			Thunderbird		
	Precision	Recall	F-1 score	Precision	Recall	F-1 score	Precision	Recall	F-1 score
PCA	$0.166_{\pm 0.008}$	$0.059_{\pm 0.003}$	$0.087_{\pm 0.002}$	$0.117_{\pm 0.023}$	$0.035_{\pm 0.007}$	$0.054_{\pm 0.010}$	$0.953_{\pm 0.004}$	$0.980_{\pm 0.005}$	$0.966_{\pm 0.003}$
iForest	$0.043_{\pm 0.010}$	$0.422_{\pm 0.224}$	$0.078_{\pm 0.021}$	$0.491_{\pm 0.364}$	$0.037_{\pm 0.052}$	$0.063_{\pm 0.090}$	$0.338_{\pm 0.128}$	$0.015_{\pm 0.011}$	$0.028_{\pm 0.020}$
OCSVM	$0.058_{\pm 0.012}$	$0.910_{\pm 0.089}$	$0.108_{\pm 0.021}$	$0.073_{\pm 0.003}$	$0.345_{\pm 0.010}$	$0.121_{\pm 0.004}$	$0.550_{\pm 0.004}$	$0.998_{\pm 0.000}$	$0.709_{\pm 0.003}$
LogCluster	$\mathbf{0.996_{\pm 0.003}}$	$0.368_{\pm 0.001}$	$0.538_{\pm 0.001}$	$\mathbf{0.941_{\pm 0.015}}$	$0.641_{\pm 0.033}$	$0.762_{\pm 0.021}$	$\mathbf{0.977_{\pm 0.005}}$	$0.291_{\pm 0.063}$	$0.445_{\pm 0.067}$
DeepLog	$0.793_{\pm 0.092}$	$0.863_{\pm 0.031}$	$0.824_{\pm 0.060}$	$0.792_{\pm 0.048}$	$0.946_{\pm 0.012}$	$0.861_{\pm 0.028}$	$0.864_{\pm 0.005}$	$0.997_{\pm 0.000}$	$0.926_{\pm 0.003}$
LogAnomaly	$0.907_{\pm 0.017}$	$0.369_{\pm 0.014}$	$0.524_{\pm 0.017}$	$0.884_{\pm 0.002}$	$0.850_{\pm 0.009}$	$0.867_{\pm 0.003}$	$0.873_{\pm 0.005}$	$0.996_{\pm 0.000}$	$0.931_{\pm 0.003}$
OC4Seq	$0.922_{\pm 0.059}$	$0.758_{\pm 0.227}$	$0.808_{\pm 0.157}$	$0.441_{\pm 0.045}$	$0.352_{\pm 0.044}$	$0.391_{\pm 0.041}$	$0.901_{\pm 0.046}$	$0.823_{\pm 0.232}$	$0.845_{\pm 0.177}$
LogBERT	$0.754_{\pm 0.142}$	$0.749_{\pm 0.037}$	$0.745_{\pm 0.082}$	$0.917_{\pm 0.006}$	$0.892_{\pm 0.006}$	$0.905_{\pm 0.005}$	$0.962_{\pm 0.019}$	$0.965_{\pm 0.008}$	$0.963_{\pm 0.007}$
CAT	$0.102_{\pm 0.022}$	$0.422_{\pm 0.082}$	$0.164_{\pm 0.034}$	$0.177_{\pm 0.122}$	$0.210_{\pm 0.184}$	$0.190_{\pm 0.148}$	$0.751_{\pm 0.072}$	$0.516_{\pm 0.124}$	$0.607_{\pm 0.120}$
LogGPT	$0.884_{\pm 0.030}$	$\mathbf{0.921_{\pm 0.066}}$	$\mathbf{0.901^{*}_{\pm 0.036}}$	$0.940_{\pm 0.010}$	$\mathbf{0.977_{\pm 0.018}}$	$\mathbf{0.958^{*}_{\pm 0.011}}$	$0.973_{\pm 0.004}$	$\mathbf{1.000_{\pm 0.000}}$	$\mathbf{0.986^{*}_{\pm 0.002}}$

The asterisk indicates that LogGPT significantly outperforms the best baseline at the 0.05 level, according to the paired t-test.

transformer framework designed for anomaly detection in event sequences.

Implementation Details. We first employ Drain [10] to parse raw log messages into log keys. For the baseline models, we utilize the Loglizer [19] package to evaluate PCA, OCSVM, iForest, and LogCluster for anomaly detection. DeepLog and LogAnomaly are evaluated using the Deep-loglizer [20] package. For OC4Seq[1], LogBERT[2], and CAT[3], we use the open-source code provided by the authors separately.

As for LogGPT, we use a GPT model with 6 layers and 6 heads. The dimensions of the embeddings and hidden states are set to 60. The learning rate is set to 1e-4 for the pre-training phase and 1e-6 for the fine-tuning phase. To accommodate different datasets, we set the K in Top-K to 50% of the training log keys. It means during the test phase if an observed log key is not in the top 50% of the prediction list from the GPT, the sequence will be labeled as an anomaly. This allows us to maintain a high level of flexibility when dealing with datasets of varying sizes and characteristics. The batch size for the pre-training phase is set to 16, and we train the model for 100 epochs. The episode is set to 20 with early stop criteria to prevent overfitting and ensure efficient training.

B. Experimental Results

Performance on Log Anomaly Detection. Table II illustrates the results and standard deviation of LogGPT and various baselines over 10 runs on the HDFS, BGL, and Thunderbird datasets. The asterisk in the table indicates that LogGPT significantly outperforms the best baseline for each dataset at the 0.05 level, according to the paired t-test.

First, we can observe that PCA, iForest, and OCSVM perform poorly on the HDFS and BGL datasets, as indicated by their low F-1 scores. However, PCA's performance is notably better on the Thunderbird dataset, achieving a high F-1 score. This inconsistency in performance across datasets highlights the sensitivity of PCA to datasets.

LogCluster, specifically designed for log anomaly detection, shows improved performance over other traditional machine learning models, i.e., PCA, iForest, and OCSVM, on the HDFS and BGL datasets but is outperformed by PCA on the Thunderbird dataset. This pattern further emphasizes the importance of dataset-specific characteristics in determining the effectiveness of different methods.

Deep learning-based approaches, such as DeepLog, LogAnomaly, OC4seq, LogBERT, and CAT, outperform traditional methods across all three datasets, which shows the advantages of utilizing deep learning to capture complex patterns in log sequences.

Our proposed model, LogGPT, stands out by consistently achieving the highest F-1 scores across all three datasets, with significant margins over all baselines.

TABLE III: Performance of LogGPT with or without reinforcement learning.

Metric	Approach	HDFS	BGL	Thunderbird
Precision	LogGPT w/o RL	$0.932_{\pm 0.015}$	$0.936_{\pm 0.011}$	$0.971_{\pm 0.004}$
	LogGPT	$0.884_{\pm 0.030}$	$0.940_{\pm 0.010}$	$0.973_{\pm 0.004}$
Recall	LogGPT w/o RL	$0.790_{\pm 0.101}$	$0.975_{\pm 0.018}$	$1.000_{\pm 0.000}$
	LogGPT	$0.921_{\pm 0.066}$	$0.977_{\pm 0.018}$	$1.000_{\pm 0.000}$
F-1 score	LogGPT w/o RL	$0.853_{\pm 0.065}$	$0.955_{\pm 0.010}$	$0.985_{\pm 0.002}$
	LogGPT	$0.901^{*}_{\pm 0.036}$	$0.958_{\pm 0.011}$	$0.986^{*}_{\pm 0.002}$

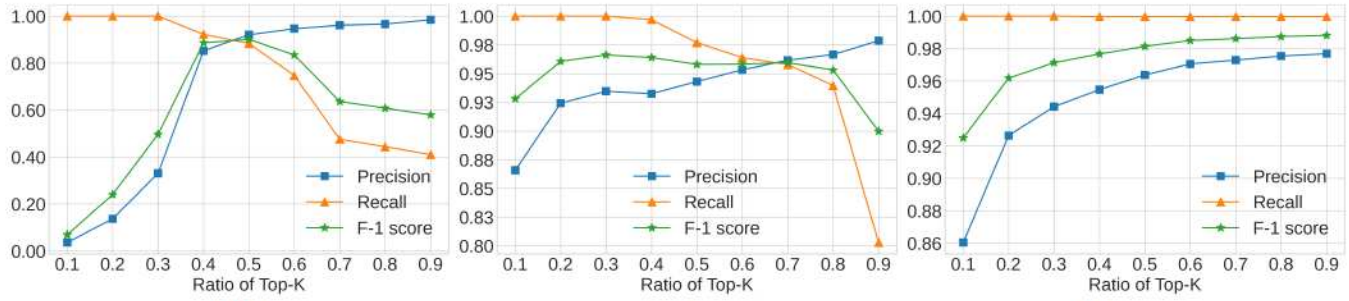
Significantly outperforms LogGPT w/o RL at the 0.05 level (paired t-test).

Ablation Studies. To investigate the contribution of reinforcement learning to the performance of LogGPT, we conduct an ablation study, comparing the performance of LogGPT with and without the RL component, shown in Table III.

First, on both HDFS and Thunderbird datasets, LogGPT significantly outperforms LogGPT without the RL component, which demonstrates that the RL component enhances the overall performance of LogGPT. Especially, on the HDFS dataset, by finetuning the GPT model with RL reward, the recall achieved by LogGPT is improved by a large margin with a little sacrifice on precision, leading to extensive improvement in the F-1 score. It also shows that fine-tuning the log language model with Top-K reward can identify more log anomalies. Meanwhile, on the BGL dataset, we can also notice a slight improvement in F-1 of LogGPT compared to the one without the RL component. Another interesting finding is that even the LogGPT without the RL component already outperforms all baselines (shown in Table II) in three datasets, which also shows the advantage of leveraging the GPT model to capture the patterns of log sequences.

Parameter Analysis: Ratio of Top-K. LogGPT detects the anomalies by examining whether the observed log key is in the Top-K list predicted by GPT. We analyze the difference in the

[1]https://github.com/KnowledgeDiscovery/OC4Seq
[2]https://github.com/HelenGuohx/logbert
[3]https://github.com/mmichaelzhang/CAT



(a) HDFS (b) BGL (c) Thunderbird

Fig. 3: Impact of the ratio of Top-K log keys.

performance by tuning K for anomaly detection. By default, K is set as 50% of unique log keys. It means if the next log key falls into the top 50% of unique log keys predicted by GPT, the sequence is normal.

The impact of different Top-K ratios on the precision, recall, and F-1 score for the HDFS, BGL, and Thunderbird datasets is illustrated in Figure 3. On both HDFS and BGL datasets, we have similar observations. With the increasing of ratios as normal log keys, the recall keeps decreasing when the ratio is greater than a threshold. This happens because when we have a large ratio, most of the keys are considered normal, leading to a low recall. On the other hand, if the observed log key is predicted with an extremely low probability, with a high chance, this log key is abnormal. Therefore, we can observe the increase in precision along with the increase in ratios.

For the Thunderbird dataset, the precision increases as the Top-K ratio increases, while the recall remains almost constant, with a slight decrease at higher Top-K ratios. The F-1 score increases steadily, reaching a peak at a specific Top-K ratio. The reason for this behavior is likely that the normal data within the Thunderbird dataset has high variability, which needs a broader range of acceptable continuations in the log sequences to reduce the false positive. As the Top-K ratio increases, LogGPT becomes more selective in flagging anomalies, thereby increasing precision by reducing false positives.

Overall, a low Top-K ratio tends to have high recall but low precision, while a high Top-K ratio leads to high precision but potentially lower recall.

VI. Conclusion

In this work, we have developed LogGPT, a novel approach to log anomaly detection that builds upon GPT models, further enhanced by a reinforcement learning strategy. Through modeling log sequences as natural language, LogGPT adapts GPT for log anomaly detection. Recognizing the gap between language modeling and anomaly detection, LogGPT integrates a fine-tuning process guided by a Top-K reward metric for anomaly detection. Extensive experiments conducted across various datasets have demonstrated the effectiveness of LogGPT, showcasing significant improvements over existing state-of-the-art methods.

Acknowledgment

This work was supported in part by NSF 2103829.

References

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017.
[2] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *IJCNN*, 2021.
[3] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM computing surveys (CSUR)*, vol. 54, no. 2, pp. 1–38, 2021.
[4] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *ICSE*, 2022.
[5] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, 2019.
[6] Z. Wang, Z. Chen, J. Ni, H. Liu, H. Chen, and J. Tang, "Multi-scale one-class recurrent neural networks for discrete event sequence anomaly detection," in *KDD*, 2021.
[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *NeurIPS*, 2017.
[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[9] S. Zhang, Y. Liu, X. Zhang, W. Cheng, H. Chen, and H. Xiong, "Cat: Beyond efficient transformer for content-aware anomaly detection in event sequences," in *KDD*, 2022.
[10] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS*, 2017.
[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.
[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[13] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP*, 2009.
[14] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN*, 2007.
[15] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Largescale system problem detection by mining console logs," *SOSP*, 2009.
[16] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*, 2008.
[17] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.
[18] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ICSE Companion*, 2016.
[19] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *ISSRE*, 2016.
[20] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," *arXiv preprint arXiv:2107.05908*, 2021.