LogGPT: Log Anomaly Detection via GPT

AUTHORS:

XIAO HAN, UTAH STATE UNIVERSITY

SHUHAN YUAN, UTAH STATE UNIVERSITY

MOHAMED TRABELSI, NOKIA BELL LABS

INTRODUCTION

The purpose of the paper is to propose a faster and better solution to the currently existing methods for Log anomaly detection. The motivation for the authors is that detecting anomalies through log data is paramount to ensuring proper computer system functionality, system security , debugging system errors and preventing data breaches.

Currently, LSTM neural networks, transformers (like BERT) and some classic ML algorithms like Principal Component Analysis and Support Vector Machines are mostly used to predict long-term sequences in log data. Long-term sequences are the series of events (or just one event) that lead to another event. In the anomaly detection case for log systems, the Machine learning model tries to predict long-term sequences that lead to an error or a security breach.

The authors took the initiative to test the newly found technology of Generative Pre-trained Transformers (GPTs) on Log anomaly detection , since there is a gap between language modelling (which is prediction of the next word based on a given set of words) and anomaly detection, calling it LogGPT. LogGPT's purpose is to predict the next log entry based on the previous sequence (prompt). The authors fine-tuned the model using reinforcement learning which will be seen in detail later on and three datasets were used to train, test and evaluate LogGPT in comparison to other existing Log anomaly detection techniques.

INTRODUCTION (CONTINUED)

Some more details about the introduction of the paper include the following: The classic ML algorithms that were used before neural networks and transformers were One-class SVM, PCA and Isolation forest. These however required a lot of manual feature engineering making it tedious for data scientists to process all the data from logging systems which could have millions of data at a time.

Neural networks (LSTMs and RNNs) were then used (the ones used for log anomaly detection were called DeepLog, LogAnomaly and OC4Seq) with a learning system that would predict if a log sequence was in a list of top K predictions predicted by a log language model. The top K predictions are essentially log entries that are sorted based on a probability score which describes how likely that log entry is to appear in the log system after the examined log sequence. If the log sequence was not present in the top K prediction list it would be an anomaly.

BERT transformers (namely LogBERT) used the same model as above.

LogGPT uses the log language model (DeepLog) described above and is pre-trained with these top K predictions to give the next log entry based on the preceding sequence (prompt). As mentioned before, it is fine tuned with a reinforcement learning algorithm that gives it a positive reward if the observed log entry is inside the top-K predictions list and a negative reward if it is not. The top-K predictions for LogGPT contain log keys (which are created for each log entry) and if the prompt log entry key is contained in the list then the reward is positive otherwise it is negative.

RELATED WORK

Each of the algorithms mentioned before was used to surpass limitations of the previous one. The classic ML algorithms as mentioned had manual feature engineering for a very large dataset and could not handle complex sequences all that well (having difficulty identifying patterns in log entries).

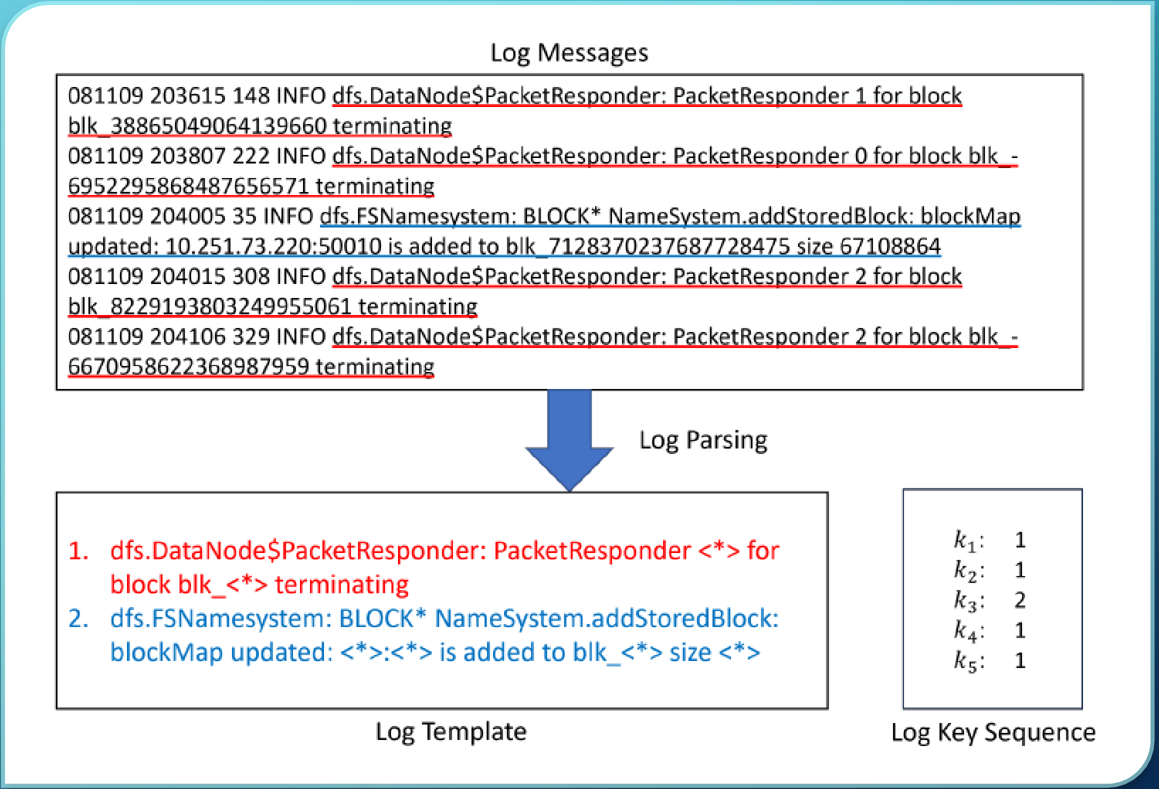
Deep Learning methods (LSTMs) address the feature engineering issue but they still underperform with very long or complex sequences due to their structure.

BERT Transformers (LogBERT) perform well and resolve the two issues above but they have difficulty handling natural language context in log entries and predicting the wrong log entry to appear. This happens because they work with a Masking log language model system, which creates masks (hides or replaces certain parts of the log sequence with mask tokens) and then predicts the sequence using the surrounding context.

LogGPT resolves this issue because it uses the GPT model which pre-trains itself on normal log sequences and therefore knows their patterns in order to predict the next log entries.

PRELIMINARY – LOG SEQUENCE PREPROCESSING

The first step in anomaly detection , as in most Machine Learning problems, is the preprocessing of the data. In Log anomaly detection a Log parser is usually used to extract the message template of the log messages. The authors used one called Drain. Each template is given a unique key called log key as mentioned in previous pages. After getting the log keys , the sequence of raw log messages can be transformed to a sequence of log keys. Log keys represent the vocabulary in a similar NLP problem and the sequence of log keys represents a sentence. Therefore , we can use a language model to interpret the log sequences. After preprocessing , log messages with the same template are represented by a log key (k) which belongs to a set of log keys (K). Then a log sequence is organized which contains all keys in order denoted as S = {k1, ..., kt, ... ,kT} where t is the position of each key and T is the size of the sequence S.



```
081109 203615 148 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block
blk_38865049064139660 terminating
081109 203807 222 INFO dfs.DataNode$PacketResponder: PacketResponder 0 for block blk_-
6952295868487656571 terminating
081109 204005 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap
updated: 10.251.73.220:50010 is added to blk_7128370237687728475 size 67108864
081109 204015 308 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block
blk_8229193803249955061 terminating
081109 204106 329 INFO dfs.DataNode$PacketResponder: PacketResponder 2 for block blk_-
6670958622368987959 terminating
```

1. dfs.DataNode$PacketResponder: PacketResponder <*> for block blk_<*> terminating
2. dfs.FSNamesystem: BLOCK* NameSystem.addStoredBlock: blockMap updated: <*>:<*> is added to blk_<*> size <*>

k_1:	1
k_2:	1
k_3:	2
k_4:	1
k_5:	1

Log key extraction from the HDFS dataset. The red/blue text are the log entry formats for the specific entries.

PRELIMINARY – LOG LANGUAGE MODEL

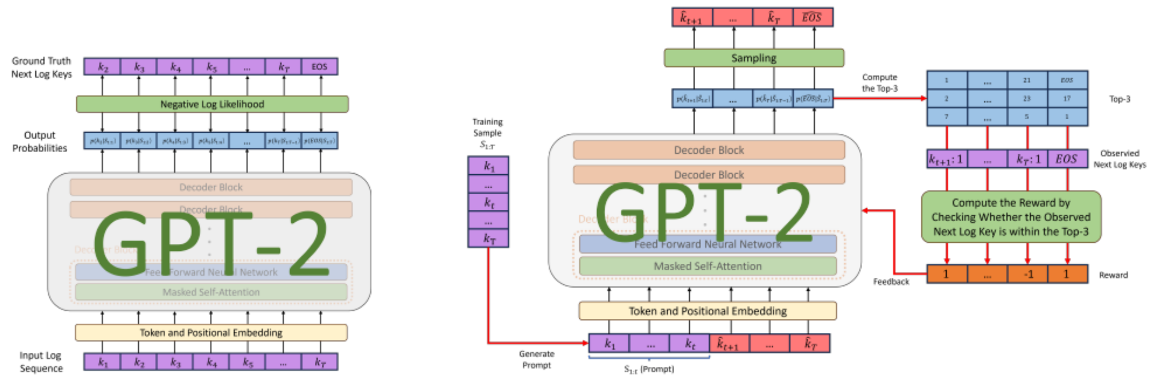
LogGPT uses DeepLog to represent the language model. DeepLog uses LSTMs to make predictions. The model essentially produces probabilistic predictions from log data and then detects anomalies that differ significantly from normal log patterns. The dataset can be defined as D = $\{S^i\}_{\{i=1\}}^{\{N\}}$ which contains normal sequences. The LSTM network in DeepLog is trained to predict the next log key in the sequence based in the preceding sequence with the following conditional probability: $p(k_{t+m+1}|S_{t:t+m})$ where m is the window size. The sliding window technique is used to split the sequences to smaller sets and predict the next log key based on the m previous keys. (e.g. if we have {k1,k2,k3} and m = 2 we'll have {k1,k2} -> predict new key based on k1,k2 and {k2,k3} -> predict new key based on k2,k3.) The LSTM is trained to maximize the likelihood of the next log key with the following loss function:

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

PRELIMINARY – LOG LANGUAGE MODEL (CONTINUED)

Where θ is the LSTM parameters. In this problem, we take the sum of the probabilities of each window of a sequence added by the sum of all other sequence windows' probabilities. Then we divide this numerator with N which is the number of sequences of the dataset to get the average. It's called a Negative-log likelihood function and is used in sequence related tasks. With each training epoch this improves the LSTM's ability to predict the next log key based on the model parameters θ.

Finally, the model predicts the probability scores and sorts them descendingly in the Top-K prediction list. Here , the logic is that instead of predicting a single 'most likely' log key, the prediction gives a set of Top-K predicted log keys. If the observed log key is outside of that list , it's sequence is considered abnormal , if it is inside the list the sequence is considered normal. The K is a hyper-parameter that can be tuned based on the experiment's needs. A small K leads to a strict model which accurately predicts anomalies but might mark some normal sequences as anomalies (Low precision, high recall). A large K leads to a more relaxed model which can learn more normal patterns due to more keys, but it might mark anomalies as normal sequences , leading to false positives (High precision, low recall).



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

Fig. 2: Framework of LogGPT.

LogGPT – Intro to Model Architecture

LogGPT is the solution proposed by the authors. It uses the same method as DeepLog , predicting the next log key based on the Top-K predicted values. However, LogGPT does not require to split the sequences to small windows and thus can learn normal log patterns and sequences easily. The training process consists of two stages: pre-training and fine-tuning.

Pre-training as shown in figure 2a, involves the model being trained on a corpus D of normal log sequences therefore learning normal patterns and structure of normal system messages.

Fine-tuning, as shown in figure 2b, uses a reinforcement learning system called Top-K metric. There is also a set of prompts P which contains sequences denoted as $S^i_{1:t}$

which are a subset of $S^i_{1:T}$

The prompts then generate the sequence above (which is the following sequence) step-by-step.

LogGPT – GENERATIVE LOG LANGUAGE MODEL

LogGPT uses the same loss function as DeepLog , but without the window part. So it is denoted as follows:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T-1} \log p(k_{t+1}^i | S_{1:t}^i),$$

In order to derive the probability p mentioned in the formula , LogGPT uses GPT structure methods:

$$\mathbf{h}_t^i = \text{Transformer\_Decoder}(S_{1:t}^i)$$
$$p(k_{t+1}^i | S_{1:t}^i) = \text{Softmax}(\mathbf{h}_t^i \mathbf{W}),$$

Where h is defined as the hidden representation given by the transformer decoder. W is the matrix that maps the hidden representation to probability distributions of all log keys in the K set which contains them. After pre-training GPT is able to generate a sequence $\hat{S}_{t+1:T}^i = \{\hat{k}_{t+1}^i, \ldots, \hat{k}_T^i\}$ based on the given part of a log sequence $S_{1:t}^i$.

LogGPT – REINFORCEMENT LEARNING

The reinforcement learning process has 4 stages. State, Action, Policy, Reward.

The state is initially defined as the given part of a log sequence. The state changes each step, essentially becoming the concatenation between the given part of the sequence and the generated sequence.

The action is defined as sampling a log key from the K log keys with the highest probability distributions denoted as $a^i_{j+1} \sim \text{Top-K}(p(\hat{k}^i_{j+1}|\tilde{S}^i_{1:j}))$ where S is the state.

The policy until the j-th position of a given sequence outputs a probability score over the action space denoted as $\pi_\theta(a^i_{j+1}|\tilde{\mathrm{h}}^i_j)$, , where θ parameters of LogGPT.

The reward provides feedback to the policy and uses the author-created Top-K metric. The Top-K metric essentially checks if the observed log key is in the Top-K predictions list. If so, then the model receives a reward of 1 otherwise it receives a reward of -1. It is denoted by the following double-branched equation:

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

This greatly improves LogGPT learning process.

LogGPT – POLICY UPDATE / ANOMALY DETECTION

Policy update happens using the following function to maximize the expected reward:

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

Where πθ is the new policy , πθold is the old policy and rj+1 is the reward for an action. This helps further improve LogGPT by maintaining stability and preventing harmful updates.

For anomaly detection, once fine-tuning is done, LogGPT predicts the log key of a given sequence based on the preceding subsequence. For each predicted log key, the model creates a Top-K predicted log keys set. The actual next log key is compared to the keys in the set , and if it's not in there , the whole log sequence is considered anomalous.

LogGPT – EXPERIMENTS - DATASETS

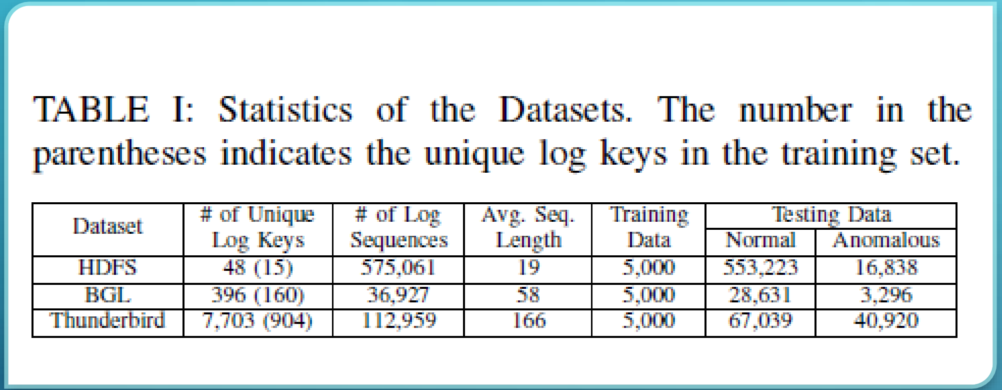


TABLE I: Statistics of the Datasets. The number in the parentheses indicates the unique log keys in the training set.

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

The authors used 3 datasets to evaluate their LogGPT model. First one was the HDFS dataset (Hadoop Distributed File System) which contains 575061 log sequences with 16838 of them being labeled anomalous.

Then they got the BGL dataset, (BlueGene/L Supercomputer System) which contans 36927 log sequences with 3296 of them being labeled anomalous.

Finally, they got the Thunderbird dataset, (another supercomputer) which contains 112959 log sequences with 40920 of them being labeled as anomalous.

LogGPT – EXPERIMENTS - EVALUATION

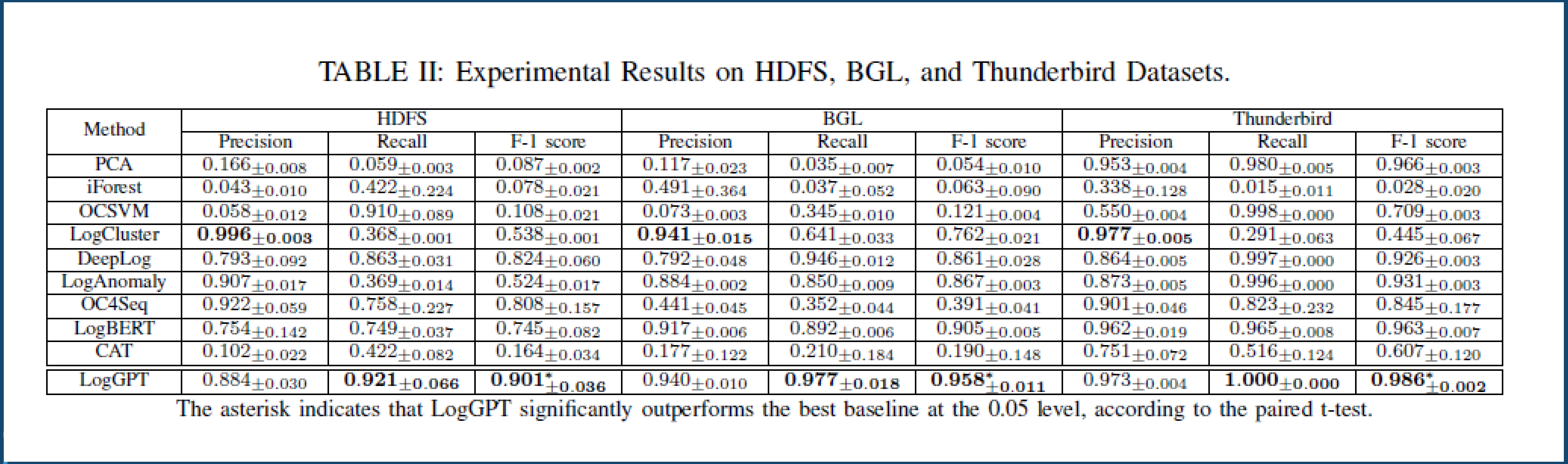


TABLE II: Experimental Results on HDFS, BGL, and Thunderbird Datasets.

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	$\mathbf{0.996}_{\pm \mathbf{0.003}}$	$0.368_{\pm 0.001}$	$0.538_{\pm 0.001}$	$\mathbf{0.941}_{\pm \mathbf{0.015}}$	$0.641_{\pm 0.033}$	$0.762_{\pm 0.021}$	$\mathbf{0.977}_{\pm \mathbf{0.005}}$	$0.291_{\pm 0.063}$	$0.445_{\pm 0.067}$
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	$0.884_{\pm 0.030}$	$\mathbf{0.921}_{\pm \mathbf{0.066}}$	$\mathbf{0.901}^{*}_{\pm \mathbf{0.036}}$	$0.940_{\pm 0.010}$	$\mathbf{0.977}_{\pm \mathbf{0.018}}$	$\mathbf{0.958}^{*}_{\pm \mathbf{0.011}}$	$0.973_{\pm 0.004}$	$\mathbf{1.000}_{\pm \mathbf{0.000}}$	$\mathbf{0.986}^{*}_{\pm \mathbf{0.002}}$

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

LogGPT – EXPERIMENTS – EVALUATION (CONTINUED)

Final results on Table II show that all classic ML algorithms along with Deep Learning models and BERT get outclassed by LogGPT , apart from one named Log Cluster, which is a clustering algorithm that creates groups of log messages and is specifically created for log anomaly detection as well. If one log message belongs to a small cluster or does not belong to any cluster it is considered an anomaly.

Furthermore, Table III shows that LogGPT with reinforcement learning performs better than LogGPT without reinforcement learning , showing the importance of the Top-K metric and the reward system to the model.

Lastly, the authors conclude that LogGPT bridges the gap between language modeling and anomaly detection since it models log sequences as natural language and has the Top-K reward metric which fine-tunes the model to improve its performance. LogGPT ,currently, outperforms all existing state-of-the-art methods.

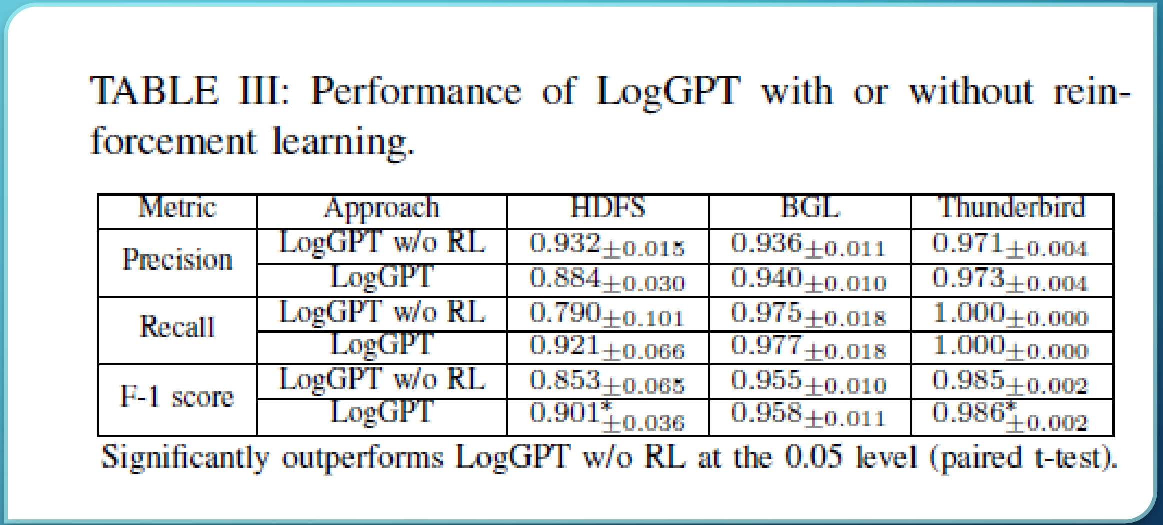


TABLE III: Performance of LogGPT with or without reinforcement learning.

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).

Presentation made by Dimitrios Tsiompikas for the Big Data Management course
Fall Semester 2023-2024,
Thank you for reading!