LogFiT: Log Anomaly Detection Using Fine-Tuned Language Models

Crispin Almodovar, Fariza Sabrina, *Member, IEEE*, Sarvnaz Karimi, and Salahuddin Azad, *Member, IEEE*

***Abstract*—System logs are a valuable source of information for monitoring and maintaining the security and stability of computer systems. Techniques based on Deep Learning and Natural Language Processing have demonstrated effectiveness in detecting abnormal behaviour from these system logs. However, existing anomaly detection approaches have limitations in terms of flexibility and practicality. Techniques that rely on log templates such as DeepLog and LogBERT fail to capture semantic information and are unable to handle variability in log content. On the other hand, classification-based approaches such as LogSy, LogRobust and HitAnomaly require time-consuming data labelling for supervised training. In this paper, a novel log anomaly detection model named LogFiT is proposed. The LogFiT model doesn't make use of a vocabulary of log templates and it doesn't require any labeled data as the model only requires self-supervised training. The LogFiT model uses a pretrained Bidirectional Encoder Representations from Transformers (BERT)-based language model fine-tuned to recognise the linguistic patterns of the normal log data. The LogFiT model is trained using masked sentence prediction on the normal log data only. Consequently, when presented with the new log data, the model's top-*k* token prediction accuracy serves as a threshold for determining whether the new log data deviates from the normal log data. Experimental results show that LogFiT's F1 score exceeds that of baselines on the HDFS, BGL, and Thunderbird datasets. Critically, when variability is introduced in the log data during evaluation, LogFiT retains its effectiveness compared to that of baselines.**

***Index Terms*—Service monitoring, fault management, log anomaly detection, deep learning, natural language processing, language modeling.**

I. Introduction

Annually, cybercrime results in billions of dollars of losses for businesses [1], [2], [3]. Log anomaly detection is an active area of research owing to its relevance to the problem of ensuring the security and reliability of organisations' digital infrastructure. The large amounts of log data generated by computer systems provide valuable information about the systems' real-time operation. However, human operators are increasingly unable to cope with the volume and velocity of log data generated by the systems being monitored. Consequently, Machine Learning (ML) and Deep Learning (DL)-based solutions have been proposed to automatically detect anomalies from system log data, thereby reducing the burden on human operators [4], [5], [6].

System logs are produced by the logging instructions that software engineers insert in a computer program's source code to communicate the program's run-time state. The system logs thus produced consist of ordered sequences of log sentences that assert the occurrence of certain events in the system [7]. Typically, log sentences are grouped according to some criteria, such as time window (e.g., 60 second intervals) or unique identifier (e.g., HDFS block ID). In this study, such grouping of log sentences is referred to as a "log paragraph". The idea behind log anomaly detection is to learn a model of the normal behaviour of a computer system based on the sequence of log sentences that it generates during normal operations. If there is a significant deviation in the new observed behaviour from the learned normal behaviour, the deviation can be regarded as an anomaly. For instance, a normal sequence of log sentences may consist of one or more "File opened successfully" entries followed by "File write operation completed" and/or "File read operation completed" entries. However, a sequence of log sentences that only contains "File opened successfully" entries without corresponding "read" or "write" entries can be considered an anomaly.

There are several considerations in the study of log anomaly detection. The first consideration is the overall Machine Learning approach to be used for anomaly detection. In literature, approaches based on Deep Learning have been proven to be more effective than traditional Machine Learning based approaches such as Principal Component Analysis (PCA), Support Vector Machines (SVM) and Isolation Forest [6], [8], [9].

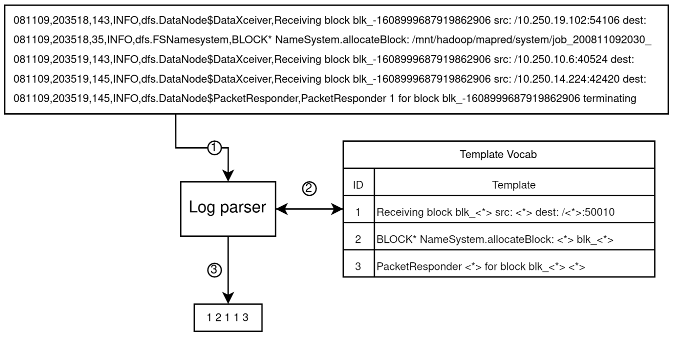
A further consideration is whether to use supervised or unsupervised Machine Learning. Due to the high cost of preparing labelled data, supervised methods such as LogSy [10], LogRobust [11] and HitAnomaly [12] have limited utility in production settings. As a result, unsupervised techniques dominate; these techniques assume a zero-positive training scenario, where normal log data is the only data available for training [4]. The study in [13] identified two categories for unsupervised models for log anomaly detection: forecasting-based model which learns by predicting the next log sentence based on immediately preceding log sentences; and reconstruction-based model which learns by reconstructing sequences of log sentences that have been intentionally corrupted.

Manuscript received 13 March 2023; revised 15 September 2023 and 9 December 2023; accepted 23 December 2023. Date of publication 25 January 2024; date of current version 15 April 2024. This work was supported in part by the Central Queensland University's School of Graduate Research and in part by the Commonwealth Scientific and Industrial Research Organisation (CSIRO)'s Data61 unit. The associate editor coordinating the review of this article and approving it for publication was Y. Diao. *(Corresponding author: Crispin Almodovar.)*

Crispin Almodovar, Fariza Sabrina, and Salahuddin Azad are with the School of Engineering and Technology, Central Queensland University, Rockhampton, QLD 4701, Australia (e-mail: crispin.almodovar@cqumail.com; f.sabrina@cqu.edu.au; s.azad@cqu.edu.au).

Sarvnaz Karimi is with Data61, CSIRO, Sydney, NSW 2015, Australia (e-mail: sarvnaz.karimi@csiro.au).

Digital Object Identifier 10.1109/TNSM.2024.3358730



```
081109,203518,143,INFO,dfs.DataNode$DataXceiver,Receiving block blk_-1608999687919862906 src: /10.250.19.102:54106 dest:
081109,203518,35,INFO,dfs.FSNamesystem,BLOCK* NameSystem.allocateBlock: /mnt/hadoop/mapred/system/job_200811092030_
081109,203519,143,INFO,dfs.DataNode$DataXceiver,Receiving block blk_-1608999687919862906 src: /10.250.10.6:40524 dest:
081109,203519,145,INFO,dfs.DataNode$DataXceiver,Receiving block blk_-1608999687919862906 src: /10.250.14.224:42420 dest:
081109,203519,145,INFO,dfs.DataNode$PacketResponder,PacketResponder 1 for block blk_-1608999687919862906 terminating
```

Template Vocab

ID	Template
1	Receiving block blk_<*> src: <*> dest: /<*>:50010
2	BLOCK* NameSystem.allocateBlock: <*> blk_<*>
3	PacketResponder <*> for block blk_<*> <*>

Fig. 1. Log parsing applied to HDFS log data.

Another consideration is the use of a log parser and log templates to generate representations of log sentences. The quality of the representation of log sequences is a critical factor that impacts the effectiveness of log anomaly detection models, especially when the evolution of log content is taken into account [14], [15]. As shown in Figure 1, log parsers [8], [9] extract string templates from the log data to build a vocabulary of all known log templates (steps 1 and 2). Subsequently, all input log sentences are mapped to an entry in the log template vocabulary (step 3). Thus a sequence of log sentences is represented by a sequence of log template IDs. This approach has been shown to negatively affect model effectiveness because of its semantically deficient representation of log sentences. Furthermore, the unseen input log sentences are expected to match a log template, which makes this approach incapable of handling variability in log sentences over time [10], [16], [17].

An important consideration is the choice of model architecture. DL-based approaches have taken inspiration from the field of Natural Language Processing (NLP) to process log data that are in the form of natural language sentences. Literature reveals that state-of-the-art research in this area employs the Long Short-Term Memory (LSTM) architecture, as exemplified by the DeepLog model [8], and Transformers, represented by the LogBERT model [9]. The problems identified with LSTMs are (1) Inability to cater to longer sequences owing to their sequential nature; and, (2) Deficient capacity to learn complex and ambiguous contextual relationships in the input data [9]. Being an LSTM-based architecture, DeepLog inherits the limitations of LSTM. On the other hand, Bidirectional Encoder Representations from Transformers (BERT) [18] has several advantages over LSTM. It inherits Transformer's [19] self-attention mechanism which helps it avoid local bias, and its use of bidirectional context allows it to capture complex and ambiguous contextual relationships. Although LogBERT inherits the advantages of BERT, its dependency on log parsing and log templates makes it less adaptable to changes in log structures. Moreover, LogBERT trains a BERT model customised to a specific dataset from scratch, which limits its ability to generalise to other datasets. Furthermore, LogBERT does not benefit from prior semantic knowledge already learned by a pretrained BERT model.

To address the limitations of existing approaches, this work introduces the following contributions:

- We propose LogFiT, a log anomaly detection model implemented as a Python package. LogFiT leverages a fine-tuned, pretrained BERT-based model to semantically analyse logs without requiring an intermediate parsing step. During inference, top-k prediction accuracy is used for anomaly identification, with heuristics to set the threshold automatically.
- The model employs a novel masked sentence prediction training objective to enhance the contextual understanding of log sentences and their constituent tokens, thus improving anomaly detection performance.
- LogFiT works directly on raw logs, eliminating the need for a separate log parsing step to extract log templates. LogFiT uses the pretrained model's extensive vocabulary of sub-word tokens to adapt to diverse log content.
- LogFiT offers a provision of using one of two base models - RoBERTa and Longformer. The former allows faster training, while the latter is capable of handling longer log sequences.
- Heuristics are used for base model selection and hyper-parameter tuning, enhancing ease of use.
- LogFiT can also serve as a semantic embedding tool, generating log representations that can be used for other tasks outside of anomaly detection.
- LogFiT is designed for easy integration into existing NLP tool sets and the larger system observability ecosystem.
- We compare LogFiT against established methods, DeepLog and LogBERT, to demonstrate its effectiveness.

The present paper differs from the authors' earlier conference paper [20], outlined as follows. Firstly, the robustness of LogFiT against gradual changes to the lexical content of log data has been demonstrated. Secondly, the centroid minimisation training objective that was used in the earlier method has been removed, and the model's effectiveness is not degraded by its removal. Furthermore, the present paper leverages k-fold cross-validation to improve the reliability of the experiments, and includes experiments that explore the effect of varying the top-k and top-k accuracy threshold values, and different time windows.

The rest of the paper is organised as follows. Section II discusses the related works on Deep Learning and natural language-based approaches for anomaly detection using log data and the pros and cons of those approaches. Section III illustrates the proposed LogFiT model in detail. Section IV describes the datasets, experimental setup and implementation details. Section V analyses the experimental results. Finally, Section V concludes the paper with future directions.

II. Related Work

Considerable research has been done on Deep Learning based supervised and unsupervised anomaly detection. Many of the approaches [8], [9], [11], [12] require log parsing, while recent studies [11], [16] suggested that log parsing can reduce accuracy. Graph-based anomaly detection [21], [22], [23], [24], [25], [26] is an emerging topic, drawing interest from researchers.

A. Supervised vs. Unsupervised Methods

1) Unsupervised Methods: DeepLog [8] pioneered the application of Deep Learning and Natural Language Processing to the log anomaly detection problem domain. By utilising a forecasting-based approach based on the patterns of past log sentences, it improved upon previous methods, such as Principal Component Analysis (PCA), Support Vector Machines (SVM) and Isolation Forest. However, its use of LSTM and log template indexes negatively affects its capability to handle complex semantic relationships between log sentences. LogAnomaly [27] built upon the principles set by DeepLog by introducing the encoding of semantic information. LogAnomaly used the template2vec method to encode the semantic content of log data, enhancing the model's ability to detect anomalies. However, LogAnomaly's use of LSTM hampers its ability to process longer sequences and complex log relationships. LogBERT [9] adopted the BERT architecture to learn more nuanced patterns in the log data, which allowed it to perform better than previous methods. However, LogBERT's use of log parsing and log templates makes it less adaptable to changes in the lexical structure of log data.

2) Supervised Methods: LogRobust [11] and HitAnomaly [12] integrate semantic vectorisation, bidirectional LSTM with attention, and transformer architecture, offering more accurate anomaly classification. However, the supervised nature limits their generalisability and scalability, demanding a significant effort in labelling data. The requirement for labelled data makes them less flexible in scenarios where obtaining labelled anomalies is challenging. Additionally, domain-specific embedding might limit their applicability across various domains. Furthermore, their reliance on specific log parsers like the Drain parser may also limit flexibility. LogSy [10] introduced flexible preprocessing through its use of a Transformer architecture. LogSy also introduced a spherical loss function, which was later adopted by LogBERT. However, LogSy's classification-based approach can create challenges in scenarios where labelled data is not readily available. The method's requirement for abnormal log lines imported from an auxiliary data source creates an external dependency, which hinders its usability in settings where auxiliary data sources are not available.

B. Parsing-Based vs. Parsing-Free Methods

1) Parsing-Based Log Anomaly Detection: As mentioned earlier, DeepLog [8] and LogBERT's [9] dependency on standardized templates limits their adaptability to new or rare log sentence structures. Also, the use of a log parsing tool might affect its ability to capture the nuanced differences in log messages. Similarly, LogRobust [11] and HitAnomaly's [12] dependency on log parsing tools limits flexibility in capturing nuanced differences.

2) Parsing-Free Log Anomaly Detection: As mentioned earlier, while LogSy's [10] use of flexible pre-processing and Transformer architecture makes it robust against the evolution of log data, its classification-based approach poses a challenge where labelled data is not readily available. LAnoBERT [28] utilises BERT's natural language understanding to detect anomalies without relying on conventional parsing. This approach allows it to handle variability in formats. However, similar to the LogBERT method, this approach trains a BERT model from scratch. This limits its reuse for other datasets and schemas and foregoes the benefits of a pretrained model's prior semantic knowledge.

C. Graph-Based Approaches

Log analysis for the intrusion detection use case primarily use graph-based models, requiring substantial feature engineering and domain knowledge to identify entities and their interactions [21], [22], [23], [24], [25], [26]. The input to graph-based models is formatted as information triples, such as (user1, authenticate, computer2). In contrast, LogFiT operates on well-formed sentences, leveraging its transformer architecture to understand both structural and semantic properties of log data [29], [30]. Recent studies indicate that Transformer models can learn representations comparable to graph-based approaches [31], [32]. Therefore in future, we aim to adapt LogFiT for intrusion detection scenarios, for example by converting intrusion detection log data into processable sentence structures.

D. Log Anomaly Detection Workflow

The log anomaly detection workflow involves four steps as identified in prior studies [5], [33], [34], [35]: pre-processing for data quality, vectorisation for Machine Learning, model development and evaluation, and final operationalisation in production. Current research does not recommend log parsing for vectorisation due to accuracy issues [11], [16], and recommends NLP-based architectures like LSTM and Transformers for optimal performance [8], [9], [10], [16].

E. Pretrained Language Models

In recent research, there has been a growing interest in the use of pretrained *language models* (LMs) such as BERT [18] to improve anomaly detection in system logs. Studies in [15], [16] demonstrated that pretrained LMs can offer significant advantages over word embeddings, that were used in the LogRobust model [11]. According to these studies, pretrained LMs capture contextual information at the level of the whole log sentence, whereas word embeddings only provide representations for individual words in a single sentence. Furthermore, pretrained LMs are capable of handling out-of-vocabulary words, unlike static word embeddings. Existing research suggests that pretrained LMs such as BERT can learn both syntactic and semantic information, which can improve the effectiveness of Natural Language Processing tasks [29], [30], [36], [37]. The BERT model is a Transformer encoder model that has the capabilities of an auto-encoder. The BERT model's encoder capability allows it to generate semantic vectors that are sensitive to the full context of the input log sequence, and to reconstruct log sequences that have been corrupted [18]. Therefore, in this study, a pretrained LM is incorporated into the LogFiT model to leverage its ability

TABLE I
COMPARATIVE ANALYSIS OF LOG ANOMALY DETECTION METHODS

Method	**Key Technical Aspects**	**Notable Limitations**
DeepLog [8]	LSTM, log templates, forecasting-based, unsupervised	Struggles with long sequences and variability, limited semantic ability
LogAnomaly [27]	LSTM, template2vec semantic encoding, forecasting-based, unsupervised	Struggles with long sequences
LogBERT [9]	BERT, log templates, reconstruction-based, spherical loss, unsupervised	Struggles with variability, limited semantic ability, reduced adaptability, miss benefits of transfer learning
LogRobust [11]	Transformer & LSTM, semantic vectorisation, classification-based	Requires labeled data, less generalisable
LogSy [10]	Transformer, semantic vectorisation, spherical loss, classification-based	Dependent on external source of abnormal logs
LAnoBERT [16]	BERT, semantic vectorisation, unsupervised	Trains from scratch, miss benefits of transfer learning
Graph-based Models [21]–[26]	Entity interaction analysis, graph-structures	Requires considerable domain knowledge
LogFiT (ours)	Transformer (any encoder model), LM fine-tuning, semantic understanding, reconstruction-based, unsupervised	Sub-optimal detection throughput

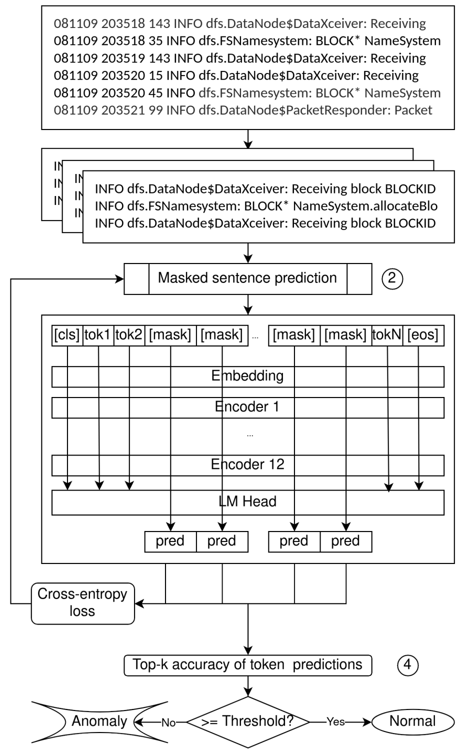
to understand the sequential properties and linguistic structure of system logs.

In Table I, various log anomaly detection methods are summarised and compared, highlighting their key technical aspects and limitations. Table I presents the landscape of existing approaches to highlight the research gap addressed by the proposed LogFiT method. LogFiT distinguishes itself from other methods by leveraging transfer learning and LM fine-tuning through masked sentence prediction, enabling it to adapt more robustly to evolving log data, unlike other methods that require log templates, LSTM, labelled data, from-scratch training, or extensive domain knowledge. Furthermore, it is important to note that LogFiT is not limited to using RoBERTa or Longformer. It is straightforward to swap out LogFiT's underlying model and replace it with other Transformer encoder models available on the Hugging Face model hub.

III. LogFiT

The proposed LogFiT approach takes advantage of advancements in Deep Learning and NLP for system log anomaly detection. It employs pretrained foundation models [38], specifically fine-tuning RoBERTa [39] or Longformer [40] to learn the linguistic and sequential properties of normal log data. Longformer is selected for its support of sequences exceeding 512 tokens, overcoming limitations in BERT [18] and RoBERTa. LogFiT incorporates a heuristic to choose between RoBERTa and Longformer based on log sequence lengths: RoBERTa for sequences up to 512 tokens, and Longformer for longer log sequences.

LogFiT utilises a self-supervised training strategy, focusing exclusively on normal log data to learn its linguistic and sequential patterns. The model aims to predict masked tokens in log sentences, employing cross-entropy loss to optimise its predictions. This loss function is logarithmic, penalising incorrect predictions more severely than correct ones. LogFiT's semantic vectors, stored in the [CLS] token, are suitable for downstream tasks like clustering and visualisation. As shown in Figure 2, the LogFiT architecture and workflow involve: (1) preprocessing raw log lines; (2) and (3) fine-tuning a pretrained RoBERTa/Longformer model on normal logs using masked sentence prediction and cross-entropy loss; and, (4) detecting anomalies in new log data. Additionally, Figure 3 shows LogFiT can be integrated into a system observability platform (such as the Elasticsearch stack) and configured



```
081109 203518 143 INFO dfs.DataNode$DataXceiver: Receiving
081109 203518 35 INFO dfs.FSNamesystem: BLOCK* NameSystem
081109 203519 143 INFO dfs.DataNode$DataXceiver: Receiving
081109 203520 15 INFO dfs.DataNode$DataXceiver: Receiving
081109 203520 45 INFO dfs.FSNamesystem: BLOCK* NameSystem
081109 203521 99 INFO dfs.DataNode$PacketResponder: Packet
```

```
INFO dfs.DataNode$DataXceiver: Receiving block BLOCKID
INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlo
INFO dfs.DataNode$DataXceiver: Receiving block BLOCKID
```

Fig. 2. Logical architecture of the LogFiT log anomaly detection approach.

for multiple log streams. That is, (1) LogFiT preprocesses centralised log data for training and inference; (2) The model trains on normal historical logs; (3) Optimal threshold is determined; (4) During inference, anomalies are detected using the model and threshold; and, (5) Alerts appear on the platform's UI for operator action.

In its early version [20], LogFiT incorporated centroid loss, inspired by the LogBERT [9] method. Subsequent research indicated that centroid loss is unnecessary, thus the present version of LogFiT does not incorporate it.

A. Framework

Input Representation: The LogFiT model utilises normal log data for training, which is converted to semantic vectors before being passed to the model. In contrast to both the DeepLog and LogBERT methods, the LogFiT model does not require the input log data to first be converted to log templates. Rather, log data is directly tokenised using the pretrained RoBERTa

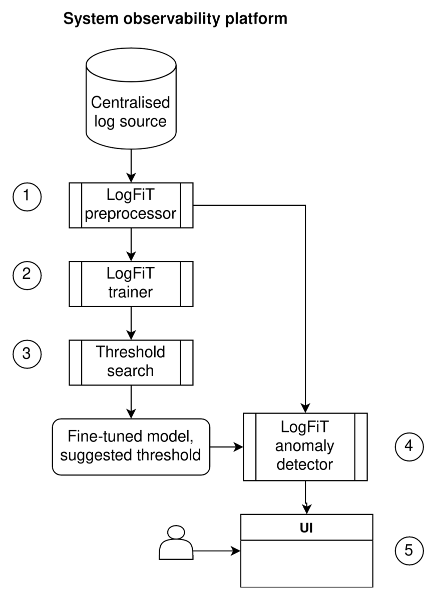


Fig. 3. LogFiT integrated into existing system observability platform.

or Longformer model's default tokeniser component. LogFiT takes inspiration from recent models [15], [16] that forego the log template extraction step and directly convert the log data into semantic vectors using a pretrained LM. These models however treat the pretrained language model as a static semantic embedding generator, whereas LogFiT fine tunes it to the log anomaly task.

Transformer Architecture: LogFiT inherits from the innovations introduced by the BERT-based language models. BERT's ability to accurately reconstruct corrupted input logs can be used as a threshold-based anomaly detection method. Thus by inheriting from BERT, LogFiT includes both the vectoriser and log anomaly detection component in a single package, allowing for end-to-end training that does not require intermediate log template extraction or vectorisation steps. As mentioned earlier, LogFiT makes use of the RoBERTa model to process log datasets containing up to 512 tokens per sample, and the Longformer model for datasets where the token count per sample exceeds 512. The Longformer model overcomes BERT's limitation on the number of tokens, which is due to the quadratic computational complexity of BERT's self-attention mechanism, by introducing a sliding window strategy that effectively reduces attention computation to a linear time [40]. The LogFiT model consists of 12 stacked Transformer encoders, 12 attention heads per layer, 768-dimension vectors, and a maximum possible sequence length of 4096 tokens – as illustrated in Figure 2.

Heuristic: The LogFiT tool also includes a heuristic to automatically select between RoBERta and Longformer based on the 0.8-quantile length (in words) of the training log samples, found during preprocessing. LogFiT selects RoBERTa for datasets with log samples containing no more than 512 tokens. For log datasets with samples that exceed 512 tokens, LogFiT switches to Longformer to take advantage of its ability to handle longer log samples via its use of local and global attention.

Fine-tuning: To adapt the RoBERTa or Longformer model to the log anomaly detection task, LogFiT utilises fine-tuning methods based on super-convergence techniques [41] implemented in the FastAI/ULMFiT framework [42], [43]. Research has demonstrated that fine-tuning yields significant performance improvements (an average gain of 2%) [44] and that gradual unfreezing [42], [44] counters the negative effects of weight dissipation during fine-tuning of pretrained language models [45].

B. Training Objective

As mentioned previously, the LogFiT model is trained in a self-supervised manner using *masked sentence* prediction. This training objective is a modified version of the self-supervised *masked language modelling* (MLM) training objective used to pretrain BERT-based language models [18]. In LogFiT a variable ratio (default 0.5) of the sentences that constitute a log paragraph is randomly chosen for masking, unlike BERT which randomly chooses a set ratio (0.15) of all the tokens that make up a log paragraph is randomly chosen for masking. Subsequently, LogFiT masks a variable ratio (default 0.8) of the tokens that make up each log sentence. Afterwards, the model is tasked with predicting what the masked tokens were. The intuition behind this training objective is to force the model to learn not just the contextual relationships between the tokens that make up a log sentence, but also the contextual relationships between the log sentences to accurately predict the masked tokens. This enables the model to develop an understanding of the language rules used by the normal system logs. As a result, it can differentiate normal log data from anomaly log data.

To satisfy the masked token prediction training objective, the model minimises the cross-entropy loss between its masked token predictions and the actual tokens. The computation of the cross-entropy loss for a mini-batch of log data is shown in Equation (1). The LogFiT model minimises the training loss using the Adam optimiser, initialised using default values from the FastAI [43] Deep Learning library: *momentum* $= 0.9$, *sqr_momentum* $= 0.99$, *epsilon* $= 1e-5$, and *weight decay* $= 0.01$.

$$Loss = -\frac{1}{b}\sum_{j=1}^{b}\sum_{i=1}^{m} y^{j}_{mask_i} \log\left(p^{j}_{mask_i}\right), \qquad (1)$$

where b is size of the mini-batch, m is the number of masked tokens, y and p are the true and predicted values, respectively.

C. Anomaly Detection

The LogFiT model, which is exclusively trained on normal data, can then be used to detect abnormal log data. During the inference stage, log paragraphs are processed in the same way as during training. To determine whether a log paragraph is anomalous, LogFiT uses a technique adopted from LogBERT. The trained model's top-k accuracy in correctly predicting the masked tokens is used as an anomaly score. If the model's top-k predictions for a masked token contain the correct token, the model's prediction is considered correct. A log paragraph is considered normal if the model's accuracy in correctly predicting the masked tokens is above some threshold. If the model's accuracy falls below the threshold, the log paragraph is deemed an anomaly. LogFiT includes a heuristic to determine the optimal threshold during hyperparameter tuning, as described in the *Experimental Setup* section.

IV. Experiments

In this section, the datasets, experimental setup, and implementation details are described. Subsequently, the results of running the experiments are evaluated.

A. Experimental Setup

Datasets: The LogFiT model is trained and evaluated using three public datasets: HDFS [46], BGL [47] and Thunderbird [47], as used by baseline models [8], [9]. While these datasets are partially labelled, LogFiT uses the labels solely for model validation. In real-world applications, logs are often unlabeled. **HDFS** logs were generated by the Hadoop Distributed File System and contain both normal and anomalous events, manually tagged by experts. Anomalies in this dataset pertain to abnormal file system operations. The dataset comprises 11,175,629 log sentences, with 284,818 identified as anomalies. **BGL** logs come from the Blue Gene/L supercomputer at Lawrence Livermore National Laboratory. It has 4,747,963 log sentences, with 348,460 categorised as anomalies. **Thunderbird** logs were produced by the Thunderbird supercomputer system at Sandia National Laboratories. The full dataset contains 211,212,192 log sentences; this study considers the first 20,000,000, of which 758,562 are anomalies.

Log paragraphs: The HDFS log sentences are chunked into log paragraphs using the HDFS block ID, which represents a session in HDFS. The BGL and Thunderbird datasets do not have a natural grouping field, so the log sentences are grouped using time windows of 10, 30 and 60 seconds. A shorter time window facilitates timely feedback for system operators. Table II shows some statistics about the HDFS, BGL, and Thunderbird datasets.

K-fold Cross-validation: In the experiments, *k*-fold cross-validation was used, specifically using a five-fold approach. For each dataset, a total of 25,000 normal and 2,000 anomaly log paragraphs were allocated for this process. The normal logs were used exclusively for training, while the anomaly logs were reserved for hyperparameter tuning and model evaluation. During each five-fold iteration, 5,000 logs sampled from the 20,000 training split were utilised for training. For hyperparameter tuning, 1,000 normal logs (from the training split) were used, supplemented with 1,000 anomaly logs. The final model evaluation exclusively used the 5,000 logs from the test split, supplemented with 1,000 anomaly logs.

Log Content Variability: To test the models' ability to handle variation in the syntactic structure of the log data, the evaluation set is dynamically modified during model evaluation on the BGL dataset, so that the top 10% most commonly occurring verbs are replaced with their WordNet [48] lemmas.

Baselines: The performance of the LogFiT model is compared against two key baselines. DeepLog [8] and LogBERT [9]. **DeepLog** utilised an LSTM-based architecture and a forecasting-based approach for anomaly detection by predicting the next log template based on its preceding ones. The model deems a log sequence normal if the correct template falls within its top-***k*** predictions. The results are produced using the logdeep library,[1] and it should be noted that the original DeepLog performance metrics are not reproducible with this implementation. **LogBERT**, on the other hand, employs a BERT-based architecture and a reconstruction-based approach. It learns normal log patterns through masked log key prediction and centroid distance minimisation. Anomalies are identified by predicting masked log keys and calculating an anomaly score based on top-***k*** accuracy and centroid distance. If either metric exceeds a certain threshold, the sequence is classified as anomalous. Results are produced from publicly available LogBERT source code, and it is noted that the original evaluations are also not reproducible.

TABLE II
Per-Paragraph Word and Sentence Statistics for the Datasets

Dataset	Avg Word Count	Avg Sentence Count	#Unique Words
HDFS	176.04	18.63	146
BGL	128.66	15.73	6,046
Thunderbird	1445.70	126.63	15,557

Implementation Details: LogFiT was implemented using Python and leveraged several well-known libraries to accelerate the development and evaluation of the model, such as Pytorch, FastAI and Hugging Face. The details of the implementation can be found in the pre-print version of this paper [49]. The source code implementing the LogFiT model, datasets and model checkpoints will be made available online.

Evaluation Metrics: To evaluate the effectiveness of the models, the experiments use the following metrics:

- *Precision* (*P*) measures the proportion of correctly identified anomaly samples (*TP*), out of all the anomalies detected by the model, and is calculated as $P = TP / (TP + FP)$.
- *Recall* (*R*) measures the proportion of correctly identified anomaly samples (*TP*) out of all real anomalies and is calculated as $R = TP / (TP + FN)$.
- *F1 Score* (*F*1) is the harmonic mean of the Precision and Recall and is calculated as $F1 = 2 * (P*R)/(P + R)$.
- *Specificity* (*S*) measures the proportion of correctly identified normal samples (*TN*) out of all real normal samples and is calculated as $S = TN/(TN + FP)$.

In real-world deployment scenarios, having a predictive model with high Specificity is more advantageous since it minimises the chances of producing false positives or false alarms. Furthermore, in [4] it was noted that a high Specificity can help mitigate the impact of having an imbalanced class distribution on the model's overall performance.

Hyperparameter tuning: During the hyperparameter tuning step, LogFiT iterates through top-*k* values in the range: 5, 9, and 12. Subsequently, for the top-*k* accuracy threshold, LogFiT iterates through values based on the top-1 token prediction accuracy of the model during training. For example, if the model's top-1 token prediction accuracy during training was 0.9, the range of values for the top-*k* accuracy threshold search is derived by computing 3 evenly spaced numbers in the

[1]Included in the LogBERT source code distribution.

range $[(0.9-0.1), 0.9]$. This computation is facilitated by the `linspace` function from the NumPy library, which yields the values 0.8, 0.85, and 0.9.

Observations: In our experiments, we observed that the implementations of LogBERT and DeepLog included the test set during the log parsing step (which builds the vocabulary of log templates), thus artificially avoiding out-of-vocabulary issues. The implementations also filtered out log instances with fewer than 10 log template IDs, which avoided a failure condition when the input sequence length is 1. We modified the implementations to align with LogFiT's setup.

We note that *k*-fold cross-validation may lead the models to *peek* into the future, by processing log samples that only occur at later times leading to the model's effectiveness improvement. This has been shown in the past in time-based datasets [50].

V. Experimental Results

Log Anomaly Detection Effectiveness: Table III, Table IV and Table V show the results of running anomaly detection inference using LogFiT, as compared to the results from running DeepLog and LogBERT using the available source code implementation. The results show that LogFiT's F1 scores exceed that of LogBERT and DeepLog on all three datasets, while LogFiT's specificity exceeds that of the baseline models on the HDFS and BGL datasets and is very close to LogBERT's on the Thunderbird dataset. The DeepLog and LogBERT models were trained and evaluated using the source code implementation mentioned earlier.

Analysis: The LogFiT results indicate that fine-tuning pretrained RoBERTa or Longformer models with a novel masked sentence prediction training objective is effective for adapting these language models to the task of detecting anomalies in system logs. This training approach enhances LogFiT's contextual understanding of log data. LogFiT's extensive sub-word token vocabulary and its lack of need for log parsing allow it to easily adapt to diverse log content. Additionally, we corrected implementation issues in DeepLog and LogBERT, which may have influenced these methods' performance. It is emphasized that while LogFiT improves upon baseline methods on standard metrics, the LogFiT method's true effectiveness is evident in scenarios where the textual content of log data changes due to log schema evolution. LogFiT's superior adaptability and robustness in handling dynamic changes in log data - achieved through LM fine-tuning - underscores its significant contribution to the domain of log anomaly detection.

Anomaly Detection Throughput: Table VI presents the throughput rates (in samples per second) for DeepLog, LogBERT and LogFiT on the HDFS, BGL and Thunderbird datasets. DeepLog excels in throughput on the HDFS dataset, owing to this dataset's shorter sequence lengths. However, its LSTM architecture limits its efficiency on the BGL and Thunderbird datasets, where sequence lengths are longer. LogBERT achieves superior throughput on these latter datasets, benefiting from its parallel token processing due to its use of a transformer architecture. LogFiT lags in throughput primarily due to its larger vocabulary size of 50K, which dictates the size of its embedding layer. Unlike DeepLog and LogBERT, whose vocabularies are based on the number of unique log templates, LogFiT's vocabulary is more extensive - which allows it to handle variability in log content. Additionally, LogFiT accommodates up to 4096 tokens, as opposed to the 512-token limit in DeepLog and LogBERT. Finally, LogFiT's lower throughput is also influenced by its generation of detailed metrics and artifacts at inference time. In contrast, DeepLog and LogBERT's outputs are simple statements printed out to the terminal. It is important to clarify that the primary objective of our research was not centred on throughput optimization. Throughput enhancements for LogFiT is a subject we explore in the future.

Centroid Distance Minimisation: The LogFiT model was extended to include a centroid distance minimisation objective (as used in LogBERT) alongside the standard masked token prediction. The loss function, as shown in Equation (2), was thus a combination of the cross-entropy loss from Equation (1) and a new term representing the centroid distance weighted by a hyperparameter *cw* (set to 0.25). The centroid distance is essentially the proximity of each log paragraph's

TABLE III
Anomaly Detection Precision (P), Recall (R), F1 Score (F) and Specificity (S) of Different Methods on the HDFS Dataset. LogFiT Values Are Averaged From 5-Fold Cross-Validation

Method	P	R	F1	S
DeepLog	**100.0**	60.90	75.70	100.0
LogBERT	24.02	82.80	37.24	47.62
LogFiT (ours)	99.78	**90.70**	**95.02**	**99.96**

TABLE IV
Anomaly Detection Precision (P), Recall (R), F1 Score (F) and Specificity (S) of Different Methods on the BGL Dataset. LogFiT Values Are Averaged From 5-Fold Cross-Validation and Time Windows of 10s, 30s, and 60s

Method	P	R	F1	S
DeepLog	90.2	70.68	79.25	98.32
LogBERT	88.92	**88.35**	88.63	97.59
LogFiT (ours)	**94.39**	85.80	**89.89**	**98.98**

TABLE V
Anomaly Detection Precision (P), Recall (R), F1 Score (F), and Specificity (S) of Different Methods on the Thunderbird Dataset. LogFiT Values Are Averaged From 5-Fold Cross-Validation and Time Windows of 10s, 30s, and 60s

Method	P	R	F1	S
DeepLog	65.05	**99.40**	78.64	89.30
LogBERT	**91.75**	95.70	93.69	**98.28**
LogFiT (ours)	89.72	98.60	**93.94**	97.74

TABLE VI
Anomaly Detection Throughput (in Samples Per Second) of Different Methods on the HDFS, BGL and Thunderbird Datasets. BGL and Thunderbird Values Are Averaged Across Time Windows of 10s, 30s, and 60s. Bold Values Represent the Highest in a Column

Method	HDFS	BGL	Thunderbird
DeepLog	**1100.91**	25.43	26.56
LogBERT	497.92	**183.08**	**127.84**
LogFiT (ours)	13.28	9.68	4.09

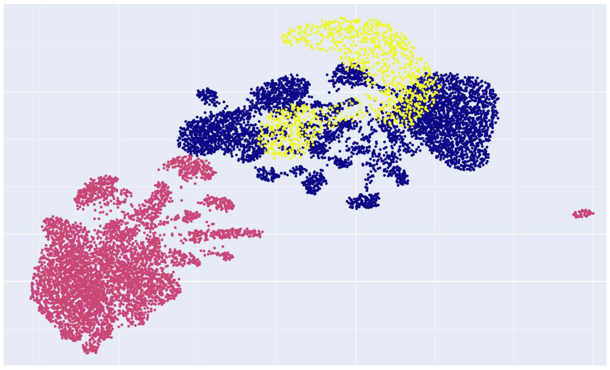


Fig. 4. UMAP plot of Thunderbird semantic vectors, where the blue, pink, and yellow colours of the points represent training samples, normal predictions, and anomaly predictions respectively.

TABLE VII
ANOMALY DETECTION PRECISION (P), RECALL (R), F1 SCORE (F) AND SPECIFICITY (S) OF DIFFERENT METHODS ON THE BGL DATASET. THE EVALUATION DATA WAS MODIFIED TO REPLACE THE TOP 10 VERBS WITH THEIR WORDNET LEMMAS

Method	P	R	F1	S
DeepLog	40.10	79.79	53.38	68.06
LogBERT	28.82	94.92	44.22	37.18
LogFiT (ours)	95.249	84.20	**89.38**	**99.16**

[CLS] vector to a mean vector calculated from normal logs. A specific q-quantile (with q between 0.65 to 0.9) centroid distance was also calculated to serve as a threshold during inference. However, experimental results indicated that the incorporation of centroid distance did not enhance the effectiveness of the model in distinguishing normal from anomalous log entries. Despite these modifications, the extended model's performance remained comparable to the original LogFiT model. Furthermore, the semantic vectors of the training set did not form distinct clusters, after dimensionality reduction via the UMAP algorithm [51], as shown in Figure 4. Consequently, we concluded that centroid distance minimisation does not offer an advantage and can be omitted from the model.

$$Loss = Loss_{cross-entropy} + cw * \frac{1}{b}\sum_{j=1}^{b}(CV_j - centroid)^2. \quad (2)$$

Variability in input data: In practical applications, it is expected that the content of the log sentences changes over time. This can be because the programmers may change some words in the log sentences, or introduce misspellings. The LogFiT model contains built-in support for log sentence variability due to its large vocabulary of sub-word tokens (around 50K). In contrast, the DeepLog and LogBERT models would fail if they encounter variations in log sentences that cannot be mapped to their list of known log templates. To test the LogFiT model's ability to handle log sentence variability, the evaluation set is dynamically modified during inference so that the top 10 occurring action words (that can be mapped to synonyms in WordNet) are replaced with their WordNet lemmas. Table VII shows the results of feeding the modified BGL evaluation set to the trained LogFiT, DeepLog and LogBERT models. The results indicate that the LogFiT model is robust to changes in the log sentences, as the reduction in F1 is around 2% (from 91.22 to 89.38). However, the drop in F1 performance for LogBERT is large, from 88.63 to 44.22. Similarly for DeepLog, F1 dropped from 79.25 to 53.38.

VI. CONCLUSION

The paper has introduced LogFiT, a novel log anomaly detection model that leverages the general linguistic knowledge of a pretrained BERT-based LM by adapting it to learn the linguistic patterns of normal system logs. LogFiT is trained using a novel self-supervised masked sentence prediction objective, using only normal log data. This approach enables LogFiT to recognise the linguistic structure of normal system logs only, thus anomalies can be flagged when the model fails to predict the correct log sentences for new log data. Critically, LogFiT can handle variability in the content of system logs because of its use of a BERT-based LM. The performance of LogFiT on the HDFS, BGL, and Thunderbird datasets has been evaluated and it has been that LogFiT's F1 score outperformed that of the baseline models. Moreover, LogFiT's specificity exceeded that of the baselines on the HDFS and BGL datasets and was comparable to LogBERT on the Thunderbird dataset. In addition, LogFiT demonstrated superior effectiveness over LogBERT in experiments that tested for variations in the content of input log paragraphs, which is attributed to its ability to handle out-of-vocabulary tokens. LogFiT integrates with the popular Hugging Face ecosystem, making it easy to adapt in future work. Overall, LogFiT presents a flexible approach to detecting abnormal behaviour in computer systems through language model adaptation and fine-tuning.

A. Future Work

While the LogFiT model is intended to be used as a threshold-based anomaly detector trained in a self-supervised manner, it can easily be converted to a classifier. If at some point after the model is deployed, operators can collect and label anomaly log samples, the model can be converted to a classifier by replacing its language modelling head with a classification head. Additionally, the LogFiT LM can be pretrained on diverse log datasets, allowing it to be used as the foundation for downstream NLP and log anomaly detection tasks. Furthermore, LogFiT's suitability for the intrusion detection use case can be considered in future studies. Lastly, ongoing research to address LogFiT's sub-optimal throughput performance focuses on efficient training and deployment strategies. These include the use of LoRA adapters, quantisation, and optimised model serving environments. These initiatives are aimed at improving LogFiT's operational effectiveness in real-world scenarios, balancing its throughput with its anomaly detection capabilities.

REFERENCES

[1] (RiskIQ Inc, San Francisco, CA, USA). *The Evil Internet Minute 2019*. (2019). [Online]. Available: https://www.riskiq.com/resources/infographic/evil-internet-minute-2019

[2] (Australia Dept. Home Affairs, Belconnen, Australia). *Australia's Cyber Security Strategy 2020*. (2020). [Online]. Available: https://www.homeaffairs.gov.au/cyber-security-subsite/files/cyber-security-strategy-2020.pdf

[3] (Int. Bus. Mach. Technol. Corp., Armonk, NY, USA). *Cost of a Data Breach Report 2022*. (2022). [Online]. Available: https://www.ibm.com/security/data-breach

[4] V. H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proc. 44th Int. Conf. Softw. Eng. Assoc. Comput. Mach.*, 2022, pp. 1356–1367. [Online]. Available: http://arxiv.org/abs/2202.04301

[5] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Comput. Surveys*, vol. 54, no. 6, pp. 1–37, 2021.

[6] Z. Chen, J. Liu, W. Gu, Y. Su, and M. R. Lyu, "Experience report: Deep learning-based system log analysis for anomaly detection," 2022, *arXiv:2107.05908*.

[7] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proc. 33rd ACM/IEEE Int. Conf. Autom. Softw. Eng.*, 2018, pp. 178–189.

[8] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2017, pp. 1285–1298.

[9] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw.*, 2021, pp. 1–8. [Online]. Available: https://arxiv.org/abs/2103.04475v1

[10] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *Proc.-IEEE Int. Conf. Data Min., (ICDM)*, 2020, pp. 1196–1201.

[11] X. Zhang et al., "Robust log-based anomaly detection on unstable log data," in *Proc. 27th ACM Joint Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 807–817. [Online]. Available: https://doi.org/10.1145/3338906.3338931

[12] S. Huang et al., "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2064–2076, Dec. 2020.

[13] L.-P. Yuan, P. Liu, and S. Zhu, "Recompose event sequences vs. predict next events: A novel anomaly detection approach for discrete event logs," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2021, pp. 336–348.

[14] D. Hendrycks, X. Liu, E. Wallace, A. Dziedzic, R. Krishnan, and D. Song, "Pretrained Transformers Improve Out-of-Distribution Robustness," in *Proc. 58th Annu. Meet. Assoc. Comput. Linguist.*, 2020, pp. 2744–2751. [Online]. Available: https://arxiv.org/abs/2004.06100v2

[15] H. Ott, J. Bogatinovski, A. Acker, S. Nedelkoski, and O. Kao, "Robust and transferable anomaly detection in log data using pre-trained language models," in *Proc. IEEE/ACM Int. Workshop Cloud Intell. (CloudIntell.)*, 2021, pp. 19–24.

[16] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2021, pp. 492–504.

[17] T. Wittkopp et al., "A2Log: Attentive Augmented Log Anomaly Detection," in *Proc. Hawaii Int. Conf. Syst. Sci. (HICSS)*, 2021, pp. 1–10. [Online]. Available: http://arxiv.org/abs/2109.09537

[18] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Am. Chapter Assoc. Comput. Linguist. Human Lang. Technol. (NAACL) HLT*, 2019, pp. 4171–4186. [Online]. Available: https://arxiv.org/abs/1810.04805v2

[19] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Adv. Neural Inf. Process. Syst.*, 2017, pp. 5999–6009.

[20] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "Can language models help in system security? investigating log anomaly detection using BERT," in *Proc. 20th Annu. Workshop Aust. Lang. Technol. Assoc.*, 2022, pp. 139–147. [Online]. Available: https://aclanthology.org/2022.alta-1.19

[21] A. Alsaheel et al., "ATLAS: A sequence-based learning approach for attack investigation," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, 2021, pp. 3005–3022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/alsaheel

[22] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. N. Venkatakrishnan, "Holmes: Real-time apt detection through correlation of suspicious information flows," in *Proc. IEEE Symp. Secur. Privacy (SP)*, 2019, pp. 1137–1152.

[23] I. J. King and H. H. Huang, "Euler: Detecting network lateral movement via scalable temporal link prediction," *ACM Trans. Privacy Secur.*, vol. 26, no. 3, pp. 1–36, 2023.

[24] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1777–1794.

[25] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "UIScope: Accurate, instrumentation-free, and visible attack investigation for gui applications," in *Proc. NDSS*, 2020, pp. 1–18.

[26] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," 2023, *arXiv:2301.13686*.

[27] W. Meng et al., "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2019, pp. 4739–4745.

[28] Y. Lee, J. Kim, and P. Kang, "LAnoBERT: System log anomaly detection based on BERT masked language model," 2023, *arXiv:2111.09564*.

[29] G. Jawahar, B. Sagot, and D. Seddah, "What does BERT learn about the structure of language?" in *Proc. 57th Annu. Meet. Assoc. Comput. Linguist. Conf.*, 2020, pp. 3651–3657. [Online]. Available: https://aclanthology.org/P19-1356

[30] D. Yenicelik, F. Schmidt, and Y. Kilcher, "How does BERT capture semantics? A closer look at polysemous words," in *Proc. 3rd BlackboxNLP Workshop Anal. Int. Neural Netw. NLP*, 2020, pp. 156–162. [Online]. Available: https://aclanthology.org/2020.blackboxnlp-1.15

[31] C. Ying et al., "Do transformers really perform bad for graph representation?" 2021, *arXiv:2106.05234*.

[32] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," 2021, *arXiv:2012.09699*.

[33] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 1–38, Mar. 2021. [Online]. Available: https://doi.org/10.1145/3439950

[34] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," 2019, *arXiv:1901.03407*.

[35] N. Zhao et al., "An empirical investigation of practical log anomaly detection for online service systems," in *Proc. 29th ACM Joint Meet. Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1404–1415. [Online]. Available: https://doi.org/10.1145/3468264.3473933

[36] Y. Lin, Y. C. Tan, and R. Frank, "Open Sesame: Getting inside BERT's Linguistic Knowledge," in *Proc. ACL Workshop BlackboxNLP Anal. Int. Neural Netw. NLP*, 2019, pp. 241–253. [Online]. Available: https://github.com/

[37] Y. Goldberg, "Assessing BERT's syntactic abilities," 2019, *arXiv:1901.05287*.

[38] R. Bommasani et al., "On the opportunities and risks of foundation models," 2022, *arXiv:2108.07258*.

[39] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.

[40] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," 2020, *arXiv:2004.05150*.

[41] L. N. Smith and N. Topin, "Super-convergence: very fast training of neural networks using large learning rates," in *Proc. SPIE Commer. Sens.*, 2019, p. 36.

[42] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *Proc. 56th Annu. Meet. Assoc. Comput. Linguist.*, 2018, pp. 328–339. [Online]. Available: https://aclanthology.org/P18-1031

[43] J. Howard and S. Gugger, "Fastai: A layered API for deep learning," *Information*, vol. 11, no. 2, p. 108, 2020.

[44] S. Gururangan et al., "Don't stop pretraining: Adapt language models to domains and tasks," in *Proc. 58th Annu. Meet. Assoc. Comput. Linguist.*, 2020, pp. 8342–8360.

[45] A. Kumar, A. Raghunathan, R. Jones, T. Ma, and P. Liang, "Fine-tuning can distort pretrained Features and underperform out-of-distribution," 2022, *arXiv:2202.10054*.

[46] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 37–44.

[47] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2007, pp. 575–584.

[48] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[49] C. Almodovar, F. Sabrina, S. Karimi, and S. Azad, "LogFiT: Log anomaly detection using fine-tuned language models," Techrxiv. 2023.

[50] S. Karimi, J. Yin, and J. Baum, "Evaluation methods for statistically dependent text," *Comput. Linguist.*, vol. 41, no. 3, pp. 539–548, 2015.

[51] L. McInnes, J. Healy, and J. Melville, "UMAP: Uniform manifold approximation and projection for dimension reduction," 2020, *arXiv:1802.03426*.