Published: 29 May 2024

RESEARCH-ARTICLE

PreLog: A Pre-trained Model for Log Analytics

VAN HOANG LE, The University of Newcastle, Australia, Callaghan, NSW, Australia

HONGYU ZHANG, Chongqing University, Chongqing, China

Proceedings of the ACM on Management of Data, Volume 2, Issue 3 (May 2024)
https://doi.org/10.1145/3654966
EISSN: 2836-6573

PreLog: A Pre-trained Model for Log Analytics

VAN-HOANG LE, The University of Newcastle, Australia and Chongqing University, China
HONGYU ZHANG*, Chongqing University, China

Large-scale software-intensive systems often produce a large volume of logs to record runtime status and events for troubleshooting purposes. The rich information in log data enables a variety of system management and diagnosis tasks. Over the years, many approaches have been proposed for automated log analytics. However, these approaches usually design separate models for each specific task, which cannot be generalized to other tasks. They are also not robust when dealing with logs from heterogeneous sources. In this paper, we propose PreLog, a novel pre-trained model for log analytics. PreLog is pre-trained on a large amount of unlabelled log data to capture the semantic meaning of logs. We design two log-specific pre-training objectives, including entry-level and sequence-level objectives, which enable PreLog to better understand the hidden structure and semantics of logs. To perform downstream log analytics tasks, we leverage a prompt tuning paradigm to convert downstream tasks' objectives into a similar form as the pre-training stage. We have conducted extensive experiments on two main log analytics tasks (i.e., log parsing and log-based anomaly detection). Experimental results show that PreLog achieves better or comparable results in comparison with the state-of-the-art, task-specific approaches. PreLog is cost-effective and can be uniformly applied to many log analytics tasks through the prompt tuning paradigm.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Information systems** → **Data management systems**.

Additional Key Words and Phrases: Log Data, Log Analytics, Pre-training, Log Parsing, Log-based Anomaly Detection

ACM Reference Format:
Van-Hoang Le and Hongyu Zhang. 2024. PreLog: A Pre-trained Model for Log Analytics. *Proc. ACM Manag. Data* 2, 3 (SIGMOD), Article 163 (June 2024), 28 pages. https://doi.org/10.1145/3654966

1 INTRODUCTION

Software logs are semi-structured data printed by logging statements (e.g., `printf()`, `logInfo()`) in source code. Logs allow engineers to better understand system behaviours and diagnose problems. The rich information included in log data enables a variety of system management and diagnosis tasks, such as analyzing usage statistics [10, 56], ensuring application security [71, 76], and diagnosing errors [46, 51]. Therefore, log data plays an important role in the maintenance, operation, and development of large-scale software systems.

With the ever-increasing scale and complexity of software systems such as data center and cloud services, a large amount of logs is routinely generated by most components (e.g., web servers, databases). To avoid the error-prone and time-consuming manual work, many data-driven approaches [25, 36, 38, 97] have been proposed to analyze logs. For example, He et al. [36] and Du

*Hongyu Zhang is the corresponding author.

Authors' addresses: Van-Hoang Le, vanhoang.le@uon.edu.au, The University of Newcastle, Australia and Chongqing University, China; Hongyu Zhang, hyzhang@cqu.edu.cn, Chongqing University, Chongqing, China.

ACM 2836-6573/2024/6-ART163
https://doi.org/10.1145/3654966

et al. [24] proposed to extract event templates and parameters from raw log messages by mining the heuristic characteristics of log data. Du et al. [25] and Zhang et al. [98] proposed DeepLog and LogRobust to detect anomalies from software logs.

Despite the progress, it is still challenging to effectively analyze the log data. We have identified the following three main challenges:

- Existing log analytics methods [51, 67, 98] are specifically designed for a certain type of log analytics tasks. Although effective, they require much effort to train a model from scratch with a significant amount of labelled log data, which is scarce and costly to obtain [25, 52].
- Like other software artifacts, logs are constantly evolving. Developers may frequently modify logging statements in source code, leading to changes to log data over time. Empirical studies find that 20-40% of the logging statements change throughout the whole lifetime [47] and up to 30% logs are modified in the latest version [98].
- Logs are heterogeneous across different software systems and different logging frameworks [62]. Different software systems and frameworks have distinct rules and policies to generate logs. To achieve good performance, current methods require to re-train their models when dealing with logs from new sources [62, 100].

In recent years, pre-trained language models (LMs) such as BERT [21], GPT [79], and BART [57] have significantly improved the performance of a variety of natural language processing (NLP) tasks. These pre-trained models learn effective contextual representations from massive unlabelled text optimized by self-supervised objectives, such as masked language modelling [21, 61]. The success of pre-trained models in NLP also promotes the rapid development of these models in other fields such as data operation [31, 85] and software development [32, 90]. For example, Tu et al. [85] proposed Unicorn, a unified pre-trained model to support common matching tasks in data integration. Gu et al. [31] applied pre-trained models for Text-to-SQL translation to enable data querying for non-technical users. Recently, there have been some studies that apply pre-trained NLP models to log analytics [66, 74]. For example, NeuralLog [51] and LogStamp [83] utilize the pre-trained BERT [21] model in anomaly detection and log parsing, respectively. Some recent studies (e.g., LogBERT [34], Logsy [74]) train language models on log data for anomaly detection. However, they are proposed as task-specific models for anomaly detection, and cannot be generalized for other log analytics tasks (e.g., log parsing) due to the differences between the tasks' training objectives. Moreover, when applying to new datasets, they must retrain the detection models from scratch with a large amount of labelled data, which is often unavailable in practice.

Existing log analytics studies only attempt to solve a single task for log analytics, such as log parsing [15] or anomaly detection [45]. Unifying multiple log analytics tasks is in urgent demand yet not well-explored. Therefore, in this paper, we propose PreLog, a pre-trained model with contrastive learning that unifies different log analytics tasks into a single framework. PreLog adopts the sequence-to-sequence Transformer [87] architecture and is pre-trained on a large amount of unlabelled log data. We design two pre-training objectives, including entry-level and sequence-level objectives, to not only teach the model to learn the robust semantic representations of logs but also force the model to understand the common types of log evolution [47]. Particularly, the entry-level pre-training objective simulates the changes to logging statements by corrupting the original inputs and then forces the model to reconstruct them. The sequence-level pre-training objective simulates the unstable log sequences caused by the changes in log sequences during log evolution. It guides the model to learn effective and robust representation for log sequences with contrastive learning. The pre-trained PreLog model is then leveraged for downstream tasks by a prompt tuning paradigm. By converting the training objectives of downstream tasks into the next token prediction task, which is similar to the objective of the pre-training phase, PreLog can be easily applied to different

downstream tasks without introducing any new model parameters or re-training the model from scratch.

We pre-train the PreLog model on a corpus consisting of 12 log datasets collected from a variety of software systems with more than 215 million tokens in total. We then evaluate the proposed model on two representative log analytics tasks (log template generation and log-based anomaly detection) on public log datasets. The extensive experimental results show that PreLog achieves better or comparable results in comparison with state-of-the-art, task-specific methods. Compared to task-specific log analytics methods, the advantages of PreLog are four-fold: (1) PreLog learns general log patterns from diverse data sources during the pre-training stage, thus enabling it to effectively handle the heterogeneity of log data, (2) the proposed pre-training objectives allow PreLog to handle unstable log events and log sequences caused by evolving logging statements that arise during software maintenance, (3) prompt tuning requires less data compared to training task-specific models from scratch, making PreLog a cost-effective solution when dealing with limited labelled log data, and (4) PreLog can be better generalized to different log analytics tasks as it converts the training objectives of these tasks into a similar form as the pre-training stage.

The main contributions of this paper are as follows:

- We propose PreLog, a pre-trained model for log analytics. We also propose two log-specific pre-training objectives that enable PreLog to effectively learn semantics and representations of logs from heterogeneous log data.
- We leverage a prompt tuning paradigm to uniformly adapt the pre-trained PreLog to downstream log analytics tasks. It reduces the gap between pre-training and downstream tasks, leading to better performance while requiring less domain knowledge than standard fine-tuning.
- We perform extensive experiments and the results show that PreLog is effective in both log template generation and log-based anomaly detection tasks. We release our pre-trained model [7], which can be easily adapted to other downstream log analytics tasks.

The remainder of this paper is organized as follows. We introduce the background and related work of our work in Section 2. Section 3 describes our approach. Section 4 presents our experimental design, followed by experimental results in Section 5. Section 6 discusses why PreLog can work, the challenges of adopting large language models for log analytics, and the threats to validity. We survey related work in Section 7 and conclude this paper in Section 8.

2 PRELIMINARIES

2.1 Log Data

Logs are generated at runtime from logging statements in the source code. They record systems' events and internal states for trouble shooting purposes. In general, logs are semi-structured text printed by logging statements (e.g., `printf()`, `logInfo()`) in the source code. Figure 1 shows an example of log messages printed by two logging statements in the Spark's source code. Each log message contains its message header and content. The message header is determined by the logging framework, such as timestamp, component, PID, and verbosity level [100]. Log content is a composition of constant strings written by developers and parameters recorded by systems during runtime. The constant part reveals the event/template of a log message and remains the same for every event occurrence. The parameter part carries dynamic runtime information (i.e., parameters), which may vary across different event occurrences.

Logging Statements

```
# Logging statements from Spark (spark/storage/BlockManager.scala)
private def reportAllBlocks(): Unit = {
    logInfo(s"Reporting ${blockInfoManager.size} blocks to the master.")
    for ((blockId, info) <- blockInfoManager.entries) {
        // ...
        if (info.tellMaster && !tryToReportBlockStatus(blockId, status))
            logError(s"Failed to report $blockId to master; giving up.")
            // ...
        }
    }
}
```

Log Message

```
17/08/22 15:50:02 INFO BlockManager Reporting 12 blocks to the master.
17/08/22 15:50:46 ERROR BlockManager Failed to report rdd_5_1 to master; giving up.
17/08/22 15:52:36 ERROR BlockManager Failed to report rdd_3_3 to master; giving up.
...
```

Fig. 1. An example of logging statements and the generated logs from Spark

2.2 Log Analytics

Log analytics employs data mining and machine learning techniques to automatically explore and analyze a large volume of log data. Figure 2 shows the common workflow of log analytics to support log management operations. Firstly, log messages from system consoles or files are collected during system runtime. Then, all entities/fields are extracted from raw unstructured logs (i.e., log parsing) for indexing, compression, or querying of logs [2, 16, 60]. After parsing, data-driven approaches are applied to analyze large volumes of log data to glean meaningful patterns and informative trends [37]. The extracted patterns and knowledge could facilitate monitoring, administering, and troubleshooting of software systems, which enable notifying, visualizing, and auditing of log management platforms. As a key component, log analytics lately has become an appealing selling-point [1, 3, 5, 8] of many industrial log management solutions [37, 100] (e.g., Splunk [8]). In this section, we shall introduce preliminaries and related works of log parsing and log mining, which are the core of log analytics.

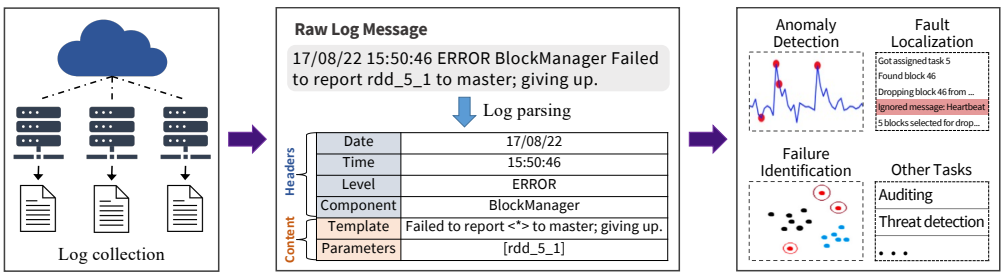

Fig. 2. The workflow of log analytics pipeline

To enable automated log analytics, log parsing is the first and foremost step [100]. It is a process to extract the static log template parts and the corresponding dynamic parameters (or variables) from semi-structured log messages. As the example shown in Figure 4, each log message is parsed into *log header* and *log content*. The log header (e.g., Datetime, Component, and Level) generated by the logging framework are generally easy to extract. The log template "`Failed to report <*> to master; giving up.`" associated with parameters (e.g., "`rdd_5_1`"), in contrast, is often difficult to identify [54, 100]. The ultimate goal of log parsing is to convert each log message into a specific log template and extract the corresponding parameters [62, 100]. Existing log parsing approaches can

be divided into two main categories. (1) *Data-driven log parsing*: many studies have been proposed to leverage data-driven approaches for log parsing, including frequent pattern mining [19, 72, 86], clustering [28, 59, 81], and heuristics [24, 36, 69]. These methods, although effective, ignore the semantic meaning of log data, thus achieving sub-optimal results [44, 54]. (2) *Deep learning-based log parsing*: recent studies [54, 83] consider semantic information from logs to formulate log parsing as a token classification task, which is found to outperform traditional data-driven log parsers.

After parsing, logs are often grouped into sequences (i.e., series of log events that record specific execution flows [51]) and are used in various reliability assurance tasks (i.e., log mining), such as anomaly detection and failure diagnosis [37]. Among them, anomaly detection is the most popular task, which aims at identifying the system's anomalous patterns that do not conform to expected behaviors on log data [37]. Many approaches have been proposed to apply machine learning (ML) [14, 38, 95] and deep learning (DL) [25, 34, 73] methods for automated log-based anomaly detection. DL-based approaches have been found to outperform traditional ML-based approaches, which have the limitations of inflexible features and weak adaptability [51, 98]. For example, LogRobust [98] and SwissLog [58] train an attention-based Bi-LSTM model to detect anomalies from unstable logs with features extracted by word2vec [70] and BERT [21] models. Log data is also used in failure diagnosis [11, 99] to help engineers understand the underlying causes and take mitigation actions. For example, CLog [11] models log sequences with a self-attention encoder model to identify failure types. EvLog [43] localizes individual fault-indicating logs based on the context of their surrounding log messages. Onion [99] pinpoints incident-indicating logs by examining their consistency, impact, and bilateral-difference.

Although effective, the existing log analytics approaches often make assumptions about specific tasks, the characteristics of log data, and the availability of domain expertise [54, 101]. Therefore, they require significant efforts to design task-specific models and to train a model from scratch with a significant amount of labelled log data, which is scarce and costly to obtain [25, 52].

2.3 Pre-trained Models

Large pre-training models trained on massive corpora of unlabelled data have been shown to perform well on a wide range of tasks, including natural language processing [21, 80], computer vision [23, 78], and code intelligence [32, 90]. These models are pre-trained in a self-supervised manner (i.e., pre-training) to capture sufficient domain knowledge. The pre-trained models can then be tuned for downstream tasks.

2.3.1 Fine Tuning. Fine-tuning a pre-trained model for downstream tasks is a prevalent paradigm in NLP field that adopts the pre-trained model in a supervised manner. A standard way to apply fine-tuning is to add task-specific layers on top of the pre-trained model and train it with a supervised objective such as classification. Although straightforward, the inconsistency between pre-training and fine-tuning objectives often restrains the rich knowledge distributed in pre-trained models, thus leading to sub-optimal results [88]. Besides, fine-tuning requires domain and task-specific knowledge to design and optimize new parameters for additional layers.

2.3.2 Prompt Tuning. To reduce the gap between pre-training and downstream tasks, prompt tuning (i.e., prompt-based fine tuning) has been proposed to convert the training objective of downstream tasks into a similar form as the pre-training stage. Instead of designing a new training objective for each downstream task, prompt tuning rewrites the input by adding a natural language instruction to reuse the pre-training objective [29, 88].

Specifically, prompt tuning employs a prompt template $f_{prompt}(.)$ to reconstruct the original input X, producing new input $\overline{X}$. As illustrated in Figure 3, the prompt template (e.g., "`[X] This sequence is [MASK]`") is a textual string that contains two types of reserved slots, i.e., input slot

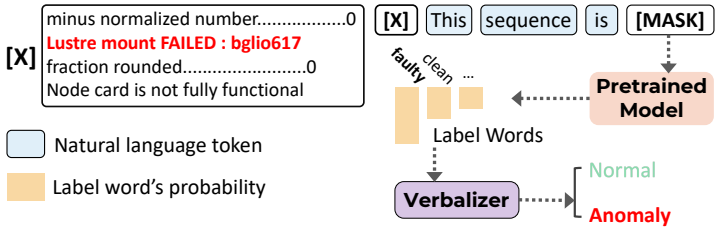

Fig. 3. An illustration of prompt tuning

$[X]$ and answer slot $[MASK]$. The input slot $[X]$ is reserved to be filled with the input text, and the answer slot $[MASK]$ is to be filled by predicted label words such as *faulty*. Prompt tuning outputs the final class by mapping the predicted label words through a verbalizer, i.e., $\mathcal{V}$, an injective function to map each predicted label word to a class in the target class set Y:

$$\mathcal{V} : W \rightarrow Y \tag{1}$$

where W indicates the label word set. For the example in Figure 3, W includes *"[faulty]"* for anomalies and *"[clean]"* for normal. According to the flexibility of the prompt template, prompt tuning techniques can be categorized into two types: *hard prompt* that uses fixed tokens and *soft prompt* that uses learnable tokens to construct the prompt template.

Pre-training has been widely applied in software engineering to model source code [9, 27, 90]. Recently, some studies have been proposed to utilize the benefits of pre-training models in NLP for log analytics. For example, NeuralLog [51] leverages a pre-trained BERT model to extract the semantic meaning of logs for anomaly detection. LogStamp [83] performs log parsing as a token classification problem by fine-tuning the pre-trained BERT model [21]. However, these methods solely apply pre-trained models for feature extraction and are specifically designed for a single log analytics task (such as anomaly detection).

Different from them, we pre-train a model on large-scale unlabelled log data. Furthermore, our approach utilizes prompt tuning for performing unified downstream tasks into the next token prediction task with less task-specific knowledge and domain expertise.

3 THE DESIGN OF PRELOG

In this section, we introduce our proposed model PreLog for log understanding and analysis. The overall framework of PreLog is shown in Figure 4. It is comprised of the following components:

- **Pre-trained model** consists of encoder and decoder layers. Encoder layers capture the semantic information of an input log sequence and map it into a high-dimensional embedding space. Decoder layers reconstruct/generate the target sequence given the input log sequence.
- **Pre-training objectives** are to generate a similar log sequence using *text corruption* or *sequence manipulation* as a form of data augmentation. They are used to guide the model to learn log syntax and semantics through LM (language modelling) loss and align embedding space through contrastive loss.
- **Prompt tuning paradigm** is used to convert the training objective of downstream tasks into a similar form as the pre-training stage to unify different log analytics tasks into a standard language modelling task (i.e., predicting the next tokens).

We first illustrate our model with a concrete example shown in Figure 4 and then introduce each component in detail.

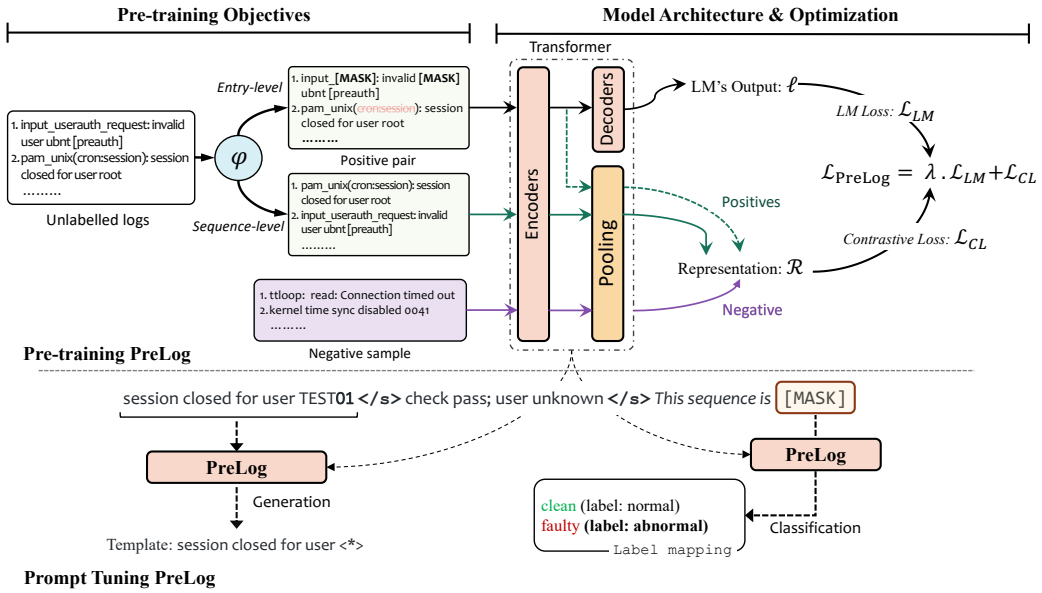

Fig. 4. An overview of our framework

3.1 An Illustrative Example

In this section, we introduce our model with an illustrative example shown in Figure 4. The top side illustrates the pre-training process of PreLog, and the bottom shows how to apply the well-trained PreLog model on downstream tasks. Specifically, first, at each iteration, we perform *text corruption* (i.e., the entry-level objective) and *sequence manipulation* (i.e., the sequence-level objective) to generate the positive samples from the input log sequence (i.e., a set of log messages). With the entry-level objective, we randomly replace some tokens with "[MASK]" or random tokens, delete some tokens, and insert some tokens into the input log sequence. With the sequence-level objective, we randomly shuffle/duplicate/remove some log entries from the input log sequence. Second, we randomly select an auxiliary log sequence from different software systems to serve as the negative sample. Then, we optimize the model using a joint loss function consisting of an LM loss and a contrastive loss. The LM loss is used to learn the syntax and semantics of log data by reconstructing the corrupted log sequence. The contrastive loss is used to align the representations of logs by pulling in similar/positive samples and pushing away different/negative samples. Finally, the well-trained model is used for log analytics. In detail, we use prompt tuning to convert the training objectives of downstream tasks into predicting the next tokens (i.e., generation manner). For example, in Figure 4, PreLog generates the complete log template for the log parsing task and the label word for the anomaly detection task.

3.2 Model Architecture

PreLog follows the sequence-to-sequence architecture [57, 87]. It is based on the standard Transformer model [87], which contains encoder, decoder, and attention mechanism. Following [9, 57], PreLog is designed to have six layers of encoder and six layers of decoder. The model dimension is 768, and the number of attention heads is 12. In total, it has approximately 140M parameters.

Following the work of [61, 79], we use a byte-level BPE vocabulary containing 50K subword units to encode the input sequence into a set of tokens. Specifically, given a log sequence, we first

tokenize it into a set of tokens (words and subwords) and separate different log messages by a special token (i.e., "</s>"). Next, we transform this set of tokens into an augmented sequence (to be described in the next subsection). We then pass the augmented sequence to the Transformer-based LM and get the output (LM's output), which captures the semantic meaning of each log message through language modelling. We also obtain the representation of the augmented sequence as the last hidden state of the Transformer's encoders through a pooling layer. The pooling layer gathers the representation vector for a log sequence as the mean of embedding vectors of the tokens. The representation then serves as the input for contrastive learning to guide the model to enhance the robustness of log sequence representation.

3.3 Pre-Training Objectives

We describe the pre-training objectives used in PreLog in this section. Pre-training objectives are crucial for the model to understand the hidden structure and the semantic meaning of logs. As shown in Figure 4, we design two log-specific pre-training objectives, including entry-level and sequence-level pre-training objectives. Formally, given a pre-training corpus $\mathcal{D} = \{X_1, X_2, \ldots, X_n\}$, where X_i is the log sequence generated using fixed or sliding windows. The pre-training objective is a function $\varphi : X_i \rightarrow (\widehat{X}_i, X_i^+)$ to transform the original log sequence X_i into augmented sequences (1) $\widehat{X}_i$ using text corruption (within the entry-level pre-training objective) and (2) X_i^+ using sequence manipulation (within the sequence-level pre-training objective). The PreLog model is then pre-trained to reconstruct X_i from $\widehat{X}_i$ and align the representation of $X_i, \widehat{X}_i,$ and X_i^+, simultaneously.

3.3.1 Entry-level pre-training objective. In practice, developers often insert/remove some words when they update a logging statement in the source code during software maintenance, which in turn leads to new log events (i.e., log entry) with similar meaning [98]. Therefore, we design the entry-level pre-training objective based on *text corruption* [57, 68] to simulate these cases of log instability in real-world software systems. Specifically, we adopt token masking, token deletion, and token insertion techniques [57, 68] as the entry-level pre-training objective to corrupt the input sequence and force the model to learn these changes. Table 1 illustrates examples of three corrupting techniques presented in this section.

Table 1. An illustration of text corruption for the entry-level pre-training objective

	Noisy Input	**Original Input and Target**
Token Masking	session [MASK] [MASK] user cyrus by (uid=0)	session opened for user cyrus by (uid=0)
Token Deletion	session _ _ user cyrus by (uid=0)	
Token Insertion	session opened for user cyrus by anything (uid=0)	

Inspired by recent studies [9, 57], we randomly sample 30% tokens in each sequence to mask. Of the selected tokens, 80% are replaced with a "`[MASK]`" token, 10% are left unchanged, and 10% are replaced by a randomly selected vocabulary. Then, the model is asked to predict the original tokens of these sampled tokens based on their local context, which has proven effective in previous work [21, 32]. Token masking is used to learn an effective representation by discovering the relationship between tokens in a log message through the attention mechanism. As an example, from the original text "`session opened for user cyrus by (uid=0)`" in Table 1, we replace the

whole words "opened" and "for" with the "[MASK]" tokens. Note that we mask the whole word instead of subword as it can improve the performance of pre-training models [18].

We leverage the token deletion technique to instruct the model to learn the changes in log data when developers remove some words from a logging statement. Specifically, random tokens are deleted from the input. The model is then asked to recognize which positions are missing and reconstruct the original input. Compared to masking, the model should determine which tokens are deleted and not replaced with anything else (i.e., "[MASK]" token). Previous studies [13, 57, 91] show that randomly removing some tokens during pre-training language models can improve performance on many tasks such as text classification [91] and text generation [57]. Table 1 shows an example where two words "opened" and "for" are removed from the original sequence.

Recent studies [47, 98] point out that during active development and maintenance of software systems, developers also add new words into logging statements to explain or record more information. To simulate these changes, we randomly insert some tokens into the original log entries. This objective forces the model to handle the changes in log data when developers add new words to existing logging statements. The model must give more attention to important tokens as well as determine the noisy tokens in order to discover the semantic structure of log messages and reconstruct the original log messages. Table 1 shows an example where we randomly insert the word "anything" into the log message.

Formally, we train PreLog to recover the original sequence X_i given the corrupted log sequence $\widehat{X}_i$ by minimizing the loss $\mathcal{L}_{LM}$:

$$\mathcal{L}_{LM} = -\sum_{X_i \in \mathcal{D}} \log P(X_i \mid \widehat{X}_i = \varphi(X_i)) \,, \tag{2}$$

where the likelihood P is estimated following the standard sequence-to-sequence decoding.

3.3.2 Sequence-level pre-training objective. Small changes to execution paths during software maintenance could lead to new but similar log sequences in practice [47, 98]. We propose the sequence-level pre-training objective based on *sequence manipulation* to enhance the robustness of log representation. To this end, we apply *contrastive learning* [30, 35] to maximize the similarity between the representation of semantically related log sequences (i.e., positive pairs) and minimize the similarity between discriminate log sequences (i.e., negative samples). The essence behind this idea is that a log analytics model should be able to (1) represent different log sequences with high discrimination, and (2) identify semantically related log sequences [98].

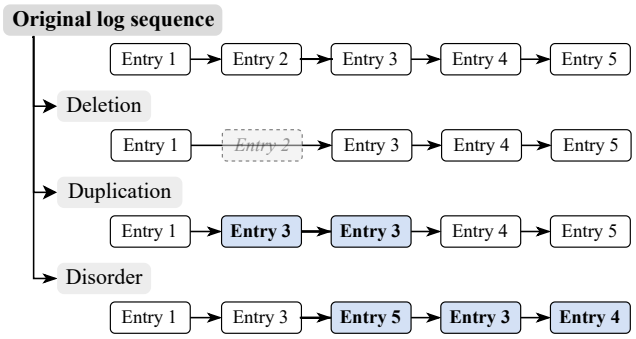

Fig. 5. Illustration of the alteration process for the sequence-level pre-training objective

To generate similar log sequences, we follow [58, 98] to manipulate the sequence by randomly removing a few log entries from the original log sequence. We also duplicate and shuffle the order of a few log entries to simulate the changes in log data in real-world scenarios. We keep the length of this *manipulated* sequence 10% different from the original log sequence so that the major sequence meaning is preserved [68]. Figure 5 illustrates this process.

For example, as shown in Figure 4, we construct the positive pair with a corrupted sequence (entry-level) and a manipulated log sequence (sequence-level). We also select a random auxiliary log sequence from different software systems to serve as the negative sample to further improve the performance. The model is trained to align the positive pair originated from the same source sequence while separating them from the negative, unrelated sample.

Formally, given a mini-batch (a small set of $\mathcal{D}$) with N samples x_i, we extend it to (x_i, x_i^+, x_i^-), where x_i (generated using text corruption) and x_i^+ (generated by sequence manipulation) is the positive pair. x_i^- is the negative sample. PreLog is trained to minimize the contrastive loss $\mathcal{L}_{CL}$:

$$\mathcal{L}_{CL} = -\log \frac{\exp\big(\texttt{sim}(r_i, r_i^+)\big)}{\sum_{j=1}^{N}\Big(\exp\big(\texttt{sim}(r_i, r_j^+)\big) + \exp\big(\texttt{sim}(r_i, r_j^-)\big)\Big)} \tag{3}$$

where r_i, r_i^+, r_i^- are the representations of the sequence x_i, x_i^+, x_i^-, respectively, produced by the PreLog model. The similarity metric "`sim`" is cosine similarity. We choose contrastive learning for the sequence-level objective because sequence manipulation makes it too challenging to reconstruct the original sequence. Recent studies find that simply deleting, swapping, or duplicating text spans (i.e., log entries in our paper) often degrades the performance of models on downstream tasks, especially on generation tasks [26]. Hence, we leverage these sequence manipulation techniques with contrastive learning to align semantically related sequence representations [26, 37, 50, 93].

Overall Training. PreLog uses the following loss function:

$$\mathcal{L}_{\text{PreLog}} = \lambda \times \mathcal{L}_{LM} + \mathcal{L}_{CL} \tag{4}$$

The full encoder-decoder model is pre-trained by minimizing the loss $\mathcal{L}_{\text{PreLog}}$ to align the representation of log sequences through contrastive learning and reconstruct the original log sequences from corrupted sentences. λ is the weighting factor for the LM loss $\mathcal{L}_{LM}$. We follow [30] to empirically set $\lambda = 0.1$. These two pre-training objectives are used simultaneously in the loss function defined in Equation (4). The full encoder-decoder model is utilized in downstream applications.

3.4 Prompt Tuning PreLog

After pre-training PreLog on large-scale unlabelled data, we unify different log analytics tasks into a standard language modelling task (i.e., predicting the next token) via prompt tuning. Specifically, inspired by recent studies [12, 29, 88], we leverage natural-language prompts to guide the model for downstream tasks. In this section, we describe how to apply prompt tuning to two kinds of downstream tasks.

Classification Tasks. To apply PreLog to downstream classification tasks such as anomaly detection [38] and failure identification [11], we leverage hard prompts to tune PreLog. Specifically, given the input sequence X, we add a natural language prompt at the end of the input, and let PreLog predict the masked token "`[MASK]`". After that, we map each predicted label token to a class for each log sequence (see Figure 4). We use the same prompt template $f_{prompt}(x)$ = "`This sequence is [MASK]`" for all tasks and manually define verbalizer $\mathcal{V}$ for different tasks. We choose

hard prompt tuning because it outperforms fine-tuning and soft prompt tuning [54, 88], especially for log analytics [54].

Generation Tasks. PreLog has an encoder-decoder architecture where the decoder is capable of generating the target sequence. Therefore, it can be directly applied to generation tasks such as log template generation. The original sequence is given as input to the encoder during prompt tuning, and the decoder generates the target sequence autoregressively. An example is shown in Figure 4.

For both tasks, we directly use PreLog's seq2seq model without introducing any new parameters during the tuning phase. The objective of downstream tasks is converted into the next token prediction problem. The advantages of prompt tuning allow us to treat different log analytics tasks in a unified way. Moreover, prompt tuning also reduces the gap between pre-training and downstream tasks, which leads to better performance than standard fine-tuning. Compared to existing methods, PreLog does not require re-training a new model from scratch for downstream tasks, thus making it easier to apply on new datasets.

4 EXPERIMENTAL DESIGN

4.1 Pre-training Dataset

We pre-train PreLog on a large log corpus containing 12 real-world log datasets from multiple sources (26.2M log messages and 8.3GB in total). Specifically, we leverage four datasets from LogPai benchmark [39], including Proxifier, Thunderbird, Linux, and OpenStack datasets. Apart from that, to ensure diversity, we collect eight datasets from different benchmarks such as Liberty [75] and Cray [20]. These datasets are collected from a variety of large-scale real-world systems, including standalone software, supercomputers, operating systems, and distributed systems. This corpus allows the model to discover the meaning of words in different contexts, thus enabling the model to comprehensively represent the semantic meaning of log data. After pre-processing, we obtain a pre-training dataset of more than 215 million tokens.

4.2 Downstream Tasks and Evaluation Metrics

4.2.1 Log Parsing as a Generation Task. To demonstrate the generation ability of PreLog, we formulate the log parsing task as a log template generation task. The task aims to generate a log event template from a raw log message. We conduct experiments on 16 datasets originally from the LogPai benchmark [100], which consists of log data from 16 different systems spanning from distributed systems, supercomputers, operating systems, mobile systems, server applications, to standalone software. Each of them contains 2,000 manual-labelled log messages, which do not appear in the pre-training dataset. Recent studies [49, 62] point out that there are some errors in the original datasets, therefore, we use the corrected version of these 16 datasets from [54] in our evaluation.

To evaluate the effectiveness and robustness of our proposed method, we apply four evaluation metrics, including:

Group Accuracy (GA): Group Accuracy [100] is the most commonly used metric for log parsing. The GA metric is defined as the ratio of "correctly parsed" log messages over the total number of log messages, where a log message is considered "correctly parsed" if and only if it is grouped with other log messages consistent with the ground truth.

Parsing Accuracy (PA): Parsing Accuracy (or Message Level Accuracy [62]) is defined as the ratio of "correctly parsed" log messages over the total number of log messages, where a log message is considered to be "correctly parsed" if and only if every token of the log message is correctly identified as template or parameter.

Edit Distance (ED): Edit Distance assesses the performance of template extraction in terms of string comparison [73]. It calculates the minimum number of actions needed to convert one template into another. We apply normalized Edit Distance [65], which computes the mean Edit Distance of all compared template pairs in the dataset (parsed templates vs ground truth templates).

Accuracy with Unseen Logs (uPA): We follow [54] to evaluate the Parsing Accuracy on unseen log data. Those log events appearing only once in a dataset are considered as unseen [54]. To compute the uPA metric, we extract those log messages whose corresponding log templates only appear once and then compute the Parsing Accuracy on these log messages.

GA and ED are used to measure the effectiveness of the proposed method in terms of understanding the semantic and lexical meaning. PA and uPA, on the other hand, measure the robustness when dealing with heterogeneous and unseen log data.

4.2.2 Anomaly Detection as a Classification Task. Log-based anomaly detection is one of the most important topics in log analytics. In this task, the model aims to detect the existence of system anomaly given a set of log messages. We formulate this task as a binary classification problem (anomalous or normal) to evaluate the classification ability of PreLog. We conduct experiments on three public datasets, namely HDFS, BGL, and Spirit, which are not used in the pre-training dataset. These datasets have been used by many studies to demonstrate the effectiveness of those proposed methods [51, 52, 73].

We follow recent studies [38, 52, 98] to generate log sequences from the HDFS dataset using session windows [38], and to generate log sequences from BGL and Spirit datasets using a fixed window of 20 log lines [52]. Then, we randomly sample a small proportion (i.e., 2k log sequences, around 0.3-0.8% of the dataset) of labelled logs from a dataset as training data and use the rest for testing. We repeat this process five times and report the average results to avoid bias from randomness.

To measure the performance of our framework, we leverage the Precision, Recall, and F-measure metrics:

- *Precision:* the percentage of correctly detected abnormal log sequences amongst all detected abnormal log sequences by the model.
  $Precision = \frac{TP}{TP+FP}$.
- *Recall:* the percentage of log sequences that are correctly identified as anomalies over all real anomalies.
  $Recall = \frac{TP}{TP+FN}$.
- *F-Measure:* the harmonic mean of *Precision* and *Recall*.
  $F - measure = \frac{2*Precision*Recall}{Precision+Recall}$.

TP (True Positive) is the number of abnormal log sequences that are correctly detected by the model. FP (False Positive) is the number of normal log sequences that are wrongly identified as anomalies. FN (False Negative) is the number of abnormal log sequences that are not detected by the model.

4.3 Research Questions

In this study, we aim to investigate the following three research questions (RQs) through experimental evaluation:

RQ1: How effective and robust is PreLog on the log template generation task?
RQ2: How effective and robust is PreLog on the anomaly detection task?
RQ3: How do different pre-training objectives contribute to PreLog?
RQ4: How does PreLog perform for other log analytics tasks?

Table 2. Comparison with the state-of-the-art log parsers with 32 labelled samples

	Spell				Drain				Logram				SPINE				LogPPT				PreLog			
	GA	PA	ED	uPA	GA	PA	ED	uPA	GA	PA	ED	uPA	GA	PA	ED	uPA	GA	PA	ED	uPA	GA	PA	ED	uPA
HDFS	**1.000**	0.487	0.983	**1.000**	0.998	0.999	0.985	**1.000**	0.930	0.961	0.978	0.000	0.809	0.622	0.726	0.000	**1.000**	**0.942**	0.993	0.500	**1.000**	**0.943**	**0.994**	**1.000**
Hadoop	0.778	0.203	0.505	0.290	0.948	0.378	0.793	0.406	0.694	0.195	0.509	0.072	0.931	0.294	0.838	0.319	**0.994**	**0.912**	**0.983**	0.797	0.968	0.789	0.944	**0.812**
Spark	0.905	0.337	0.843	0.368	0.920	0.376	0.945	0.421	0.470	0.274	0.854	0.158	0.887	0.337	0.889	0.368	**1.000**	**0.999**	**0.999**	**0.842**	**1.000**	0.967	0.939	0.789
Zookeeper	0.964	0.750	0.959	0.316	0.967	0.795	0.974	0.421	0.956	0.805	0.952	0.105	0.989	0.753	0.949	0.421	**1.000**	0.990	0.990	0.947	**1.000**	**0.993**	**0.998**	**0.984**
BGL	0.787	0.275	0.760	0.227	**0.963**	0.463	0.833	0.409	0.702	0.260	0.683	0.045	0.923	0.376	0.816	0.318	0.610	0.801	0.837	0.591	0.950	**0.905**	**0.983**	**0.682**
HPC	0.654	0.575	0.637	0.600	0.932	0.683	0.580	0.800	0.933	0.649	0.565	0.400	0.900	0.650	0.943	0.400	0.990	0.926	**0.947**	**0.900**	**0.994**	**0.942**	0.941	**0.900**
Thunderbird	0.844	0.036	0.658	0.316	**0.955**	0.177	0.720	0.558	0.554	0.097	0.595	0.042	0.603	0.051	0.488	0.463	0.680	**0.933**	**0.818**	**0.747**	0.675	0.885	0.812	0.737
Windows	0.992	0.004	0.954	0.150	0.997	0.465	0.937	0.550	0.694	0.141	0.881	0.050	0.684	0.154	0.755	0.350	**0.996**	**0.999**	**1.000**	**0.950**	0.717	0.894	0.994	0.850
Linux	0.605	0.114	0.501	0.232	0.876	0.210	0.592	0.737	0.186	0.125	0.441	0.032	0.545	0.113	0.624	0.474	0.934	0.880	**0.985**	0.842	**0.994**	**0.924**	0.979	**0.842**
Android	0.921	0.245	0.903	0.294	0.885	0.750	0.962	0.784	0.795	0.436	0.555	0.314	0.755	0.139	0.754	0.431	**0.890**	**0.802**	**0.984**	**0.765**	0.840	0.614	0.781	0.747
HealthApp	0.639	0.154	0.666	0.387	0.901	0.375	0.564	0.903	0.833	0.679	0.728	0.419	0.847	0.445	0.896	0.484	**1.000**	0.789	**0.791**	**0.935**	**1.000**	0.676	0.786	0.839
Apache	**1.000**	0.978	0.995	—	**1.000**	0.978	0.995	—	**1.000**	0.972	0.994	—	**1.000**	0.276	0.804	—	**1.000**	**0.994**	**0.999**	—	**1.000**	**0.994**	**0.999**	—
Proxifier	0.527	0.478	0.735	—	0.527	0.527	0.787	—	0.504	0.000	0.380	—	0.036	0.272	0.734	—	**1.000**	**0.999**	**1.000**	—	**1.000**	**0.999**	**1.000**	—
OpenSSH	0.556	0.378	0.946	—	0.789	0.593	0.909	—	0.802	0.928	0.934	—	0.676	0.253	0.919	—	0.628	**0.975**	0.997	—	**1.000**	0.747	**0.986**	—
OpenStack	0.764	0.000	0.667	—	0.733	0.105	0.698	—	0.823	0.071	0.565	—	0.330	0.011	0.102	—	**1.000**	0.918	**0.998**	—	0.969	**0.940**	0.986	—
Mac	0.778	0.055	0.718	0.141	0.803	0.453	0.813	0.511	0.700	0.307	0.725	0.074	0.710	0.209	0.716	0.215	**0.906**	**0.663**	**0.912**	0.548	0.831	0.587	0.746	**0.615**
Average	0.794	0.317	0.777	0.360	0.887	0.520	0.818	0.625	0.723	0.431	0.709	0.143	0.726	0.309	0.747	0.354	0.914	**0.905**	**0.952**	0.806	**0.933**	0.862	0.929	**0.816**

Note: "—" denotes there are no unseen logs on these datasets

4.4 Implementation and Environment

We pre-train the PreLog model starting from a pre-trained checkpoint [57] on 4 Nvidia Tesla V100 GPUs for 100,000 steps. The effective batch size is maintained at 256 sequences × 1024 tokens ≈ 260,000 tokens/batch. We use Adam optimizer with a linear learning rate decay schedule for optimization. The total training time was approximately 7 days. All experiments are done with Fairseq [77], HuggingFace [92], and OpenPrompt [22] libraries. For prompt tuning, we train with a 5e-5 maximum learning rate. We use a maximum of 2,000 training updates for all tasks. For the template generation task, we use beam-search with a beam size of 8.

5 EXPERIMENTAL RESULTS

5.1 RQ1: Performance on Log Parsing

In this section, we evaluate the accuracy of log parsing performed by PreLog. We compare PreLog with Spell [24], Drain [36], Logram [19], and SPINE [89], i.e., the top-performing data-driven log parsers. We also compare with LogPPT [54], i.e., the current state-of-the-art deep learning based log parser. For each dataset, we randomly select 32 samples for prompt tuning (1.6% of the dataset) using an adaptive random sampling algorithm [54]. For a fair comparison, we follow LogPPT to extend unsupervised baselines, i.e. Spell, Drain, Logram, and SPINE, to include those 32 labelled samples in their parsing process. We also use those 32 labelled samples to train LogPPT. To avoid bias caused by randomness, we repeat the evaluation process five times and report the average results. Table 2 shows the comparison with the existing methods.

From the results, we can see that our model outperforms existing methods or achieves comparable results on almost all datasets in all evaluation metrics. Specifically, in terms of Group Accuracy, PreLog exceeds the baselines by 2.08% (LogPPT) to 29.05% (Logram) on average. PreLog achieves the best GA on 9 out of 16 datasets. It also achieves a GA of over 0.9 on 12 datasets and 1.0 accuracy on seven datasets. In terms of PA and ED, PreLog is 1.66-2.79x and 1.14-1.31x better than the data-driven baselines, respectively. PreLog can obtain results comparable to LogPPT, the current state-of-the-art, by achieving 95% of LogPPT's accuracy. Note that we simply tune PreLog by asking the model to generate the template given a raw log message. The obtained results are comparable to those of LogPPT, which is specifically designed for the log parsing task. In terms of uPA, PreLog achieves the best average uPA of 0.816, 1.01-5.71x better than the baselines. Among 12 datasets

that contain unseen logs, PreLog achieves the best results on 7 of them. The high results in terms of GA and ED confirm that PreLog is able to recognize similar log messages and generate the corresponding log templates effectively. Moreover, the high PA and uPA demonstrate the robustness of PreLog when dealing with different types and unseen log messages.

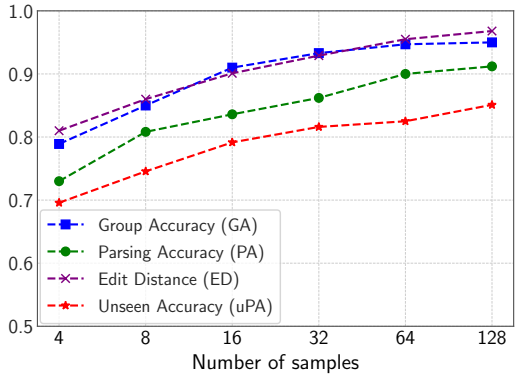

Fig. 6. Results with different numbers of labelled samples

PreLog requires a small amount of labelled data to tune the model. To evaluate the sensitivity of PreLog to the amount of labelled data, we conduct an experiment using different numbers of labelled log samples. Figure 6 shows the results, which indicate that the model's performance witnesses a small drop when less data is used for tuning. For example, the average GA and PA values obtained by PreLog drop from 0.933 and 0.862 (with 32 labelled samples) to 0.910 and 0.836 (with 16 labelled samples), respectively. The performance of PreLog can be improved if more labelled samples are provided, and it is noticeable that PreLog achieves good results when we tune it with 16 or more labelled samples.

In summary, the experimental results indicate that PreLog is effective and robust in generating log templates across heterogeneous log sources.

5.2 RQ2: Performance on Anomaly Detection

In this section, we evaluate the ability of PreLog on the anomaly detection task. We employ the prompt template $f_{prompt}(.)$ defined in Section 3.4, and a verbalizer defined as follows:

$$\mathcal{V} = \begin{cases} [\text{faulty}] & \rightarrow \text{ anomaly} \\ [\text{clean}] & \rightarrow \text{ normal} \end{cases} \tag{5}$$

As our method performs anomaly detection in a supervised manner, we compare PreLog with CNN [63], LogRobust [98], and NeuralLog [51], which are the state-of-the-arts on anomaly detection [52]. We tune the hyper-parameters of these baselines for each dataset to achieve the best results. Table 3 shows the results.

The results show that PreLog achieves the best results on HDFS, BGL, and Spirit datasets. Compared to CNN and LogRobust, NeuralLog and PreLog achieve better results since they leverage the semantic meaning of the raw log messages, thus they can avoid the errors of log parsing [51]. CNN and LogRobust rely on a log parser for pre-processing, so their performance is affected by log parsing errors [51, 52]. Another reason is that, by pre-training on a large amount of log data, PreLog can capture the semantic meaning of log sequences more effectively, thus leading to better results.

Table 3. Comparison with the state-of-the-art anomaly detection methods

Dataset	Metric	CNN	LogRobust	NeuralLog	PreLog
HDFS	Precision	0.824	0.831	0.861	0.897
	Recall	0.997	0.974	1.000	1.000
	F-measure	0.902	0.897	0.925	**0.946**
BGL	Precision	0.876	0.878	0.955	0.967
	Recall	0.996	0.988	0.988	0.982
	F-measure	0.932	0.930	0.971	**0.974**
Spirit	Precision	1.000	0.997	0.997	1.000
	Recall	0.996	0.996	0.999	0.996
	F-measure	0.997	0.996	**0.998**	**0.998**

We next evaluate the robustness of PreLog when dealing with unstable logs. As described in Section 1, like other software artifacts, logs are always evolving. Therefore, we follow recent work [58, 98] to create two common scenarios of unstable logs, including:

- Unstable log events: We randomly insert/remove a few words into/from original log events in advance. The motivation behind this setting is that developers often insert or remove some words when they update a logging statement in the source code leading to the appearance of new log events [98].
- Unstable log sequences: Log sequences are likely to be changed during the process of log evolution or collection [98]. Therefore, we follow [41, 98] to randomly remove, duplicate, or shuffle some unimportant log lines from the original log sequences to simulate the unstable log sequences.

For both scenarios, we first randomly collect 50,000 normal and 1,500 abnormal log sequences from the testing set of the BGL dataset. The percentage of anomalies is 3%, which is close to that of the original BGL dataset. Then, we inject the unstable log data into it with injection ratios from 5% to 20%. A detailed explanation of this process is presented in Section 3.3. Tables 4 and 5 show the results.

Table 4. Experimental results on synthetic BGL dataset of unstable log events

Injection Ratio	Metric	CNN	LogRobust	NeuralLog	PreLog
5%	Precison	0.614	0.683	0.755	0.900
	Recall	0.996	0.985	0.917	0.988
	F-measure	0.760	0.807	0.829	**0.942**
10%	Precison	0.594	0.501	0.656	0.897
	Recall	0.993	0.953	0.983	0.985
	F-measure	0.743	0.657	0.787	**0.939**
15%	Precison	0.657	0.507	0.643	0.891
	Recall	0.995	0.970	0.988	0.986
	F-measure	0.791	0.666	0.779	**0.936**
20%	Precison	0.567	0.581	0.632	0.889
	Recall	0.995	0.920	0.985	0.987
	F-measure	0.722	0.597	0.770	**0.936**

Table 5. Experimental results on synthetic BGL dataset of unstable log sequences

Injection Ratio	Metric	CNN	LogRobust	NeuralLog	PreLog
5%	Precison	0.690	0.693	0.864	0.903
	Recall	0.996	0.985	0.991	0.988
	F-measure	0.816	0.814	0.923	**0.943**
10%	Precison	0.676	0.689	0.861	0.915
	Recall	0.996	0.985	0.991	0.988
	F-measure	0.805	0.811	0.921	**0.950**
15%	Precison	0.667	0.693	0.893	0.912
	Recall	0.996	0.985	0.991	0.988
	F-measure	0.799	0.814	0.909	**0.936**
20%	Precison	0.657	0.688	0.763	0.905
	Recall	0.995	0.985	0.991	0.988
	F-measure	0.791	0.810	0.862	**0.945**

From the results, we can see that PreLog performs better and more stable than other approaches. With the increasing injection ratio of unstable log events, the performance of related approaches has declined within different degrees. However, PreLog still maintains a consistently high accuracy (F-measure ranging from 0.936 to 0.942 with unstable log events and from 0.936 to 0.950 with unstable log sequences) under high injection ratios. For example, PreLog can achieve an F-measure of 0.936 when performing with 20% unstable log events. We also observe that NeuralLog can retain a relatively good performance because it can capture the semantic information embedded in log messages via the pre-trained BERT model. However, PreLog achieves much better results than NeuralLog. The reason is that PreLog is pre-trained with two log-specific objectives, which can enhance the robustness and effectiveness of the representation of log sequences.

Overall, the experimental results show that PreLog is effective and robust for log-based anomaly detection on both stable and unstable log data.

5.3 RQ3: Ablation Study

In this section, we evaluate the contributions of the major components in our proposed model. Specifically, we evaluate the effectiveness of each pre-training objective when the model is trained without it. Table 6 shows the average results with log parsing.

Table 6. Ablation study results of log parsing

	GA	PA	ED	uPA
Full PreLog	0.933	0.862	0.929	0.816
w/o$_{\text{Entry-level Obj.}}$	0.906	0.828	0.861	0.749
w/o$_{\text{Sequence-level Obj.}}$	0.891	0.836	0.901	0.804

We can see that PreLog exhibits a significant drop in performance when one of the pre-training objectives is excluded. For example, the GA and ED values drop by 4.5% and 3.01% when PreLog is trained without the sequence-level objective. Overall, both pre-training objectives are important for log parsing. The entry-level objective is essential for capturing the syntax information of logs, while the sequence-level objective is vital for grouping logs with the same templates.

Table 7. Ablation study results (F-measure) of anomaly detection on stable logs

	HDFS	BGL	Spirit
Full PreLog	0.946	0.974	0.998
w/o$_{\text{Entry-level Obj.}}$	0.913	0.954	0.993
w/o$_{\text{Sequence-level Obj.}}$	0.912	0.958	0.996

Table 8. Ablation study results (F-measure) of anomaly detection on unstable BGL logs

	Injection Ratio	5%	10%	15%	20%
Unstable Event	Full PreLog	0.942	0.939	0.936	0.936
	w/o$_{\text{Entry-level Obj.}}$	0.832	0.839	0.821	0.824
	w/o$_{\text{Sequence-level Obj.}}$	0.875	0.882	0.868	0.860
Unstable Sequence	Full PreLog	0.943	0.950	0.936	0.945
	w/o$_{\text{Entry-level Obj.}}$	0.876	0.878	0.878	0.878
	w/o$_{\text{Sequence-level Obj.}}$	0.865	0.869	0.867	0.863

We next evaluate the contributions of the two proposed pre-training objectives in the anomaly detection task. Tables 7 and 8 show the results with stable and unstable logs, respectively. We only report F-measure values here due to space constraints. We can see that, PreLog still achieves acceptable performance without one of the pre-training objectives on stable log data. However, on unstable log events and sequences, PreLog performs worse in both scenarios when one of the pre-training objectives is excluded. Specifically, PreLog achieves the lowest results on unstable events without the entry-level objective because this objective is used to simulate the change in log events. Meanwhile, without the sequence-level pre-training objective, the results on unstable sequences drop significantly because the model cannot learn the robust representation without contrastive learning.

Overall, the experimental results demonstrate the usefulness of the proposed pre-training objectives. Each of them plays an important role in different scenarios of downstream tasks.

5.4 RQ4: The Generality of PreLog for Other Log Analytics Tasks

In this section, we evaluate the generality of PreLog for other log analytics tasks. We conduct experiments on two failure diagnosis related tasks, including fault localization [42, 43] and failure identification [11].

5.4.1 Fault Localization. Locating the log messages that are more likely to indicate faults can greatly assist operators in resolving system issues [42]. Since PreLog is based on the attention mechanism, log messages that have a greater impact on the model's prediction results will be given more weight. Thus, we leverage the attention scores assigned for each log message in a log sequence and rank them to localize fault-indicating logs. We evaluate PreLog on BGL and Spirit datasets using Hit Rate of top-k (i.e., $HR@k$) [55], which directly measures how likely the fault-indicating logs will be found within k checks. We choose BGL and Sprit datasets because they provide labels for each log message. The results (Table 9) show that PreLog can locate fault-indicating log messages with high accuracy.

5.4.2 Failure Identification. Identifying the type of anomalies can help system administrators to narrow down and further investigate the root cause of failures. In this section, we ask PreLog to

Table 9. Results of PreLog on fault localization

	BGL			Spirit		
	HR@1	*HR*@3	*HR*@5	*HR*@1	*HR*@3	*HR*@5
Accuracy	0.879	0.966	0.983	0.906	0.982	0.991

categorize the failure types of OpenStack system. Specifically, we adopt the dataset from [48] that contains 651 Virtual Machine (VM) failures of three types, including (1) VM is destroyed, (2) VM's virtual disk is removed, and (3) VM's performance is disturbed. We take 10% of the dataset for prompt tuning and the rest for testing. We employ the prompt template $f_{prompt}(.)$ = "`[X] This sequence is [MASK]`" as defined in Section 3.4. We then ask PreLog to predict the label words of *destroyed*, *removed*, and *disturbed* for the three failure types. We follow the experimental setting in Section 4.4 to apply PreLog on this task. The results (Table 10) show that PreLog can achieve an F-measure of over 0.95 for all failure types on the OpenStack dataset.

Table 10. Results of PreLog on failure identification

Failure types	Precision	Recall	F-measure
VM is destroyed	0.993	0.910	0.950
VM's virtual disk is removed	0.937	0.996	0.966
VM's performance is disturbed	1.0	1.0	1.0

Overall, the experimental results confirm that PreLog is able to (1) localize the fault-indicating logs in log sequences and (2) identify the types of failures. The results confirm the generality of PreLog for multiple log analytics tasks.

6 DISCUSSION

6.1 Why does PreLog work?

There are several reasons affiliated with different parts in PreLog's design make it perform better than the related methods.

6.1.1 Pre-training. PreLog is pre-trained on multiple sources of log data. It allows PreLog to learn the common hidden structure across heterogeneous log sources, making PreLog better handle inconsistent log formats. To handle incomplete log data, we pre-trained PreLog with raw log messages to consider both log templates and parameters. Pre-training enables PreLog to easily adapt to downstream tasks compared to task-specific models. To verify this point, we leverage the model architecture of PreLog to train a model from scratch (w/o pre-training) for anomaly detection and compare the results to those with PreLog. We train this task-specific model longer than PreLog (10,000 steps) because it does not contain any log-specific prior knowledge. The results in Table 11 confirm that the model fails to achieve satisfactory performance without pre-training.

The ablation study in Section 5.3 shows that with two proposed pre-training objectives, PreLog can effectively handle the instability of logs. Specifically, the entry-level pre-training objective (with token masking, deletion, and insertion) allows the model to learn the semantic meaning of tokens in a local context. It also guides the model to handle unstable log events. The sequence-level pre-training objective with contrastive learning significantly enhances the effectiveness and robustness

Table 11. Results (F-measure) of PreLog without pre-training on anomaly detection

	HDFS	BGL	Spirit
PreLog	0.946	0.974	0.998
w/o pre-training	0.853	0.886	0.990

of the sequence representation produced by the model. It instructs the model to represent the related log sequences with high similarity and the non-related log sequences with high discrimination.

6.1.2 Prompt tuning. The hard prompt tuning paradigm allows PreLog to directly transfer the knowledge in the pre-trained model to downstream tasks uniformly without introducing new hyper-parameters or re-configuring the model. To evaluate the contribution of hard prompt tuning in PreLog, we compare PreLog to the variants equipped with different tuning methods on the anomaly detection task, including:

(1) **Soft prompt:** Instead of using a hard prompt template (i.e., $f_{prompt}(x)$ = "`[X] This sequence is [MASK]`"), we use a soft prompt of $f_{prompt}(x)$ = "`[X] [SOFT] [SOFT] [SOFT] [MASK]`" with three learnable tokens (i.e., `[SOFT]`) and ask PreLog to learn the embedding of these tokens during prompt tuning.
(2) **Fine tune (freeze LM):** We add a classification layer on top of PreLog and fine tune it without tuning parameters of the pre-trained PreLog.
(3) **Fine tune (full params):** We add a classification layer on top of PreLog and fine tune it along with all parameters of the pre-trained PreLog.

The results in Table 12 confirm that PreLog with hard prompt tuning significantly outperforms other techniques with the same amount of labelled data.

Table 12. Results (F-measure) of different tuning techniques on anomaly detection

	HDFS	BGL	Spirit
$\text{PreLog}_{\text{w/ Hard prompt}}$	0.946	0.974	0.998
Soft prompt	0.737	0.952	0.797
Fine tune (freeze LM)	0.589	0.885	0.521
Fine tune (full params)	0.746	0.960	0.784

6.1.3 Model interpretation. As PreLog is trained with the attention mechanism, the model assigns different attention weights for each log message in the log sequence. Log messages that have a greater impact on the model's prediction will be given more weights by the attention mechanism. Thus, by ranking these weights, we can locate the most impactful log messages to interpret the model's decision-making process. We apply this approach to perform the fault localization task with PreLog (Section 5.4.1) and achieve accurate results. To demonstrate the decision-making process of PreLog, we show an example of fault localization with PreLog in Table 13. It can be seen that PreLog assigns a high attention value to "`data storage interrupt`", which is the ground truth for the fault-indicating log message.

6.2 Log Analytics with Large Language Models

General large language models (LLM), such as ChatGPT [6] and Llama [84], are being widely used. The capabilities of LLMs on various domains inspire researchers to apply them to AIOps, especially to log analytics. Recent studies [53, 94] have investigated the performance of GPT-3 and GPT-3.5

Table 13. Example of fault localization via attention weight

Level	Log Entry	Attn. Weight
INFO	ciod: Received signal 15, code=0, errno=0, address=0x000001f7	0.0195
INFO	ciod: Received signal 15, code=0, errno=0, address=0x000001f2	0.0197
FATAL	**data storage interrupt (∗)**	**3.8868**
FATAL	instruction address: 0x00001ec0	0.1014
FATAL	data address: 0x0001776f	0.0510
FATAL	Exception Syndrome Register: 0x00800000	1.9528
FATAL	machine check: i-fetch.....0	0.6036
FATAL	program interrupt: illegal instruction.....0	1.3935
FATAL	program interrupt: privileged instruction...0	0.7114
FATAL	program interrupt: trap instruction.....0	0.4984

(*): the ground-truth fault-indicating log message.

on log parsing and found that it is promising to apply LLMs in the domain of log analytics. These studies adopt in-context learning to extract log templates, given a few prompt demonstrations. However, we have identified the following main challenges for adopting LLMs in log analytics:

(1) Log-specific data. System logs contain a variety of dynamic log-specific information, which is generated during runtime (e.g., domain URLs, API endpoint addresses, etc.). These information, although occurs frequently, varies a lot in different software systems, hindering the ability of LLMs in understanding log data [53].

(2) Latency and cost of LLMs. In production, log data is generated in a massive amount (i.e., tens of billions of lines per day [59, 89]). Efficiency is crucial for log analytics models because they must handle large-scale log data. It has been found that the API latency can vary depending on the user's location, and the average latency of the OpenAI API service for a single request can range from a few hundred milliseconds to several seconds [96]. Furthermore, using LLMs such as ChatGPT for large-scale log data poses a financial burden on API costs. Therefore, in scenarios of AIOps where high latency/cost is not acceptable, large LLMs may not be appropriate.

We compare PreLog with a large language model on the log-based anomaly detection task to show its advantage. Specifically, we fine tune a FLAN-T5 model [17], an enhanced version of T5 [80], for log-based anomaly detection. We choose the FLAN-T5 XL version with 3 billion parameters (approximately 20 times larger than PreLog) and fine-tune it using Low-Rank Adaptation (LoRA) [40]. We fine tune FLAN-T5 using 4×A100 GPUs. We show the results in terms of effectiveness (F-measure) and efficiency (T_{infer}: inference time per log sequence) in Table 14.

Table 14. Comparison with an LLM on anomaly detection

	Metric	HDFS	BGL	Spirit
PreLog (140M)	F-measure	0.946	0.974	0.998
	T_{infer} w/ V100	12 (ms)	12 (ms)	11 (ms)
	T_{infer} w/ A100	7 (ms)	7 (ms)	7 (ms)
FLAN-T5 XL (3B)	F-measure	0.870	0.977	0.996
	T_{infer} w/ V100	—	392 (ms)	386 (ms)
	T_{infer} w/ A100	245 (ms)	265 (ms)	247 (ms)

"—" denotes timeout (48 hours).

From the results, we can see that PreLog achieves better or comparable results in terms of effectiveness compared to the fine-tuning of FLAN-T5, while performing approximately 32-37× faster and requiring less computational resources. PreLog can be easily applied to downstream tasks with one V100 GPU with low latency (12 milliseconds per log sequence), thus it is accessible to various organizations. Overall, compared to LLMs, PreLog is more suitable for log analytics. It can effectively learn log representation and handle the instability of log data. As a white-box model, PreLog can be easily updated for log analytics pipelines. Furthermore, PreLog contains only 140M parameters, which is much smaller than LLMs. Therefore, PreLog has better efficiency compared to LLMs and is cost-effective.

6.3 Threats to Validity

We have identified the following threats to validity:

Pre-training data. Due to the restriction of computational resources, we only use 12 log datasets to pre-train our model. Compared to other large-scale pre-train models for NLP or source code, the amount of data is less. To reduce this threat, we collect the datasets from a variety of systems, such as supercomputers or distributed systems. In the future, we will further pre-train our model with a larger and more diverse corpus to comprehensively understand the semantic information and hidden structure of log data.

Downstream tasks. There are many downstream tasks for log analytics. However, only a few log datasets with ground truth are publicly available. In our experiments, we comprehensively evaluate our proposed model on log parsing and anomaly detection, two widely-studied log analytics tasks, to demonstrate the generation and classification abilities of PreLog. To reduce this threat, we also evaluate PreLog on the failure identification task and the preliminary results demonstrate that PreLog can achieve high performance for identifying the failure types of OpenStack. We believe that PreLog is capable of supporting more downstream tasks, as they are essentially the classification or generation problems. In our future work, we will comprehensively evaluate PreLog on more downstream tasks.

Selection of baselines. In our evaluation, we have compared our proposed approach with the representative task-specific methods for each downstream task. These methods are widely used for log analytics [38, 52, 100] and achieve state-of-the-art (SOTA) performance [49, 52]. As PreLog applies to downstream tasks in a supervised manner, we follow LogPPT [54] to extend the unsupervised log parsers (i.e., Drain, Spell, Logram, and SPINE) to include the labelled samples in their parsing process. We also compare PreLog with SOTA supervised log-based anomaly detection models (e.g., LogRobust, NeuralLog, etc.). Recently, there are some approaches that apply deep learning to log analytics, such as UniParser [62] for log parsing and LogKG [82] for anomaly detection. However, their source code is not publicly released and it is difficult to re-implement them without introducing bias, thus we do not compare with them in this paper. We will evaluate them experimentally in our future work.

7 RELATED WORK

7.1 Log Analytics

Logs are crucial runtime information recorded by developers, which are widely employed in a variety of reliability assurance tasks. Log analytics platforms enable businesses and organizations to *gather*, *examine*, and *store* logs from various programs and applications in order to diagnose issues, optimize performance, and improve compliance [1, 8].

7.1.1 Log parsing. Log parsing is a process to parse free-text raw log messages into specific events associated with parameters, and is the first and foremost step in log analytics pipelines [100]. Existing log parsing approaches can be divided into two main categories. (1) *Data-driven log parsing*:

many studies have been proposed to leverage data-driven approaches for log parsing, including frequent pattern mining [19, 72, 86], clustering [28, 59, 81], and heuristics [24, 36, 69]. For example, Drain [36] utilizes a tree structure to parse logs into multiple templates. Logram [19] extracts frequent patterns from log data using *n*-gram information. These methods, although effective, ignore the semantic meaning of log data, thus achieving sub-optimal results [44, 54]. (2) *Deep learning-based log parsing*: recent studies [54, 83] formulate log parsing as a token classification task. For example, UniParser [62] trains a token classification model with labelled data across multiple log sources. LogPPT [54] proposes a new paradigm of template-free prompt-tuning [64] that leverages RoBERTa [61] for log parsing.

7.1.2 Log-based anomaly detection. Log-based anomaly detection aims to identify the system's anomalous patterns that do not conform to expected behaviors on log data [37]. Many approaches have been proposed to apply machine learning (ML) [14, 38, 95] and deep learning (DL) [25, 34, 58] methods for automated log-based anomaly detection. DL-based approaches have been found to outperform traditional ML-based approaches, which have the limitations of inflexible features and weak adaptability [51, 98]. DL-based approaches are mainly semi-supervised [4, 25, 34] and supervised [52, 98]. Semi-supervised models such as DeepLog [25] and LogBERT [34] train DL models (e.g., LSTM or Transformers) to predict the next possible events in a log sequence. PLELog [97] trains a weakly supervised GRU model from normal and unlabelled data to classify normal and abnormal log sequences. Supervised approaches, which train classification models using both normal and abnormal logs, significantly outperform the above semi-supervised models [52]. For example, LogRobust [98] and SwissLog [58] train an attention-based Bi-LSTM model to detect anomalies from unstable logs with features extracted by word2vec [70] and BERT [21]. NeuralLog [51] trains a Transformer model to detect system anomalies from raw logs without log parsing. Although effective, existing approaches are specifically designed for the anomaly detection task and cannot be easily generalized for other log analytics tasks.

7.1.3 Log-based failure diagnosis. Log data is also used to analyze system failures, helping engineers and administrators understand the underlying causes of failures and resolve the issues. For example, CLog [11] models log sequences with a self-attention encoder model to identify failure types. Onion [99] pinpoints incident-indicating logs by examining their consistency, impact, and bilateral-differences. EvLog [43] localizes individual fault-indicating logs based on the context of their surrounding log messages.

Unlike all the above approaches, PreLog is pre-trained with log data and can be generalized to multiple log analytics tasks, such as log parsing, anomaly detection, and failure diagnosis. PreLog can be uniformly applied to a variety of downstream tasks without introducing any new model parameters or training from scratch with a large amount of labelled data.

7.2 Pre-trained Models for Log Analytics

Pre-training has been applied to a wide range of tasks, including natural language processing [21, 80], computer vision [23, 78], and code intelligence [32, 90]. Establishing a pre-trained model for unified log analytics is desired but challenging due to log-specific characteristics such as the instability of log data. Recent studies often leverage NLP-based pre-trained models to extract semantic features from log data for downstream tasks [51, 54, 83]. There are a few studies [42, 101] that pre-train models with log data for log analytics. For example, TRANSLOG [33] transfers a pre-trained model on the source log dataset to detect anomalies on target datasets. HilBERT [42] pre-trains a BERT-based model [21] on LogPAI dataset [39] for anomaly detection. These two studies only apply pre-training to a single log-based anomaly detection task. UniLog [101] proposes to pre-train a

BERT model with log data and apply it to log analytics by training additional classification layers for downstream tasks. The above-mentioned pre-trained models solely adopt an NLP-specific objective, masked language modeling, without considering log-specific characteristics. Different from them, we pre-train PreLog with two novel log-specific objectives to learn effective and robust representation. Our experimental results indicate that the proposed log-specific objectives are essential to not only effectively analyze logs but also handle the instability of logs. Besides, by leveraging the prompt tuning paradigm, we can uniformly apply PreLog to downstream tasks without introducing any new model parameters. Noticeably, the existing studies neither make their model nor their source code publicly available, thus we cannot experimentally compare with them without introducing re-implementation bias. We will try to re-implement and evaluate them in our future work.

8 CONCLUSION

In this paper, we propose PreLog, a novel pre-training sequence-to-sequence model for log analytics. We design two pre-training objectives including entry-level and sequence-level pre-training objectives. The entry-level pre-training objective allows PreLog to effectively capture semantic meaning of tokens in log messages by simulating multiple scenarios of unstable log events. The sequence-level pre-training objective powered by contrastive learning enables PreLog to enhance the effectiveness and robustness of log representations. The pre-trained PreLog model can be uniformly applied to downstream tasks such as log template generation and anomaly detection through prompt tuning. Our extensive experimental results show that PreLog can achieve better or comparable results in comparison with state-of-the-art, task-specific methods. We also discuss the generality of PreLog for other log analytics tasks and compare it with LLM (Large Language Model) based approach.

Data Availability: Our source code and experimental data are available at https://github.com/LogIntelligence/PreLog.

9 ACKNOWLEDGEMENTS

This work is supported by Australian Research Council (ARC) Discovery Projects (DP200102940, DP220103044). We also thank anonymous reviewers for their insightful and constructive comments, which significantly improve this paper.

REFERENCES

[1] 2023. Application and Infrastructure Monitoring – Amazon CloudWatch. Retrieved Oct 10, 2023 from https://aws.amazon.com/cloudwatch/
[2] 2023. Centralized Logging on AWS. Retrieved Oct 10, 2023 from https://aws.amazon.com/solutions/implementations/centralized-logging/
[3] 2023. Elastic Stack: Elasticsearch, Kibana, Beats & Logstash. Retrieved Oct 10, 2023 from https://www.elastic.co/elastic-stack/
[4] 2023. Implementation of PLELog. Retrieved August 27, 2023 from https://github.com/YangLin-George/PLELog
[5] 2023. Logz.io: Cloud Observability and Security, Powered by Open Source. Retrieved Oct 10, 2023 from https://logz.io
[6] 2023. OpenAI ChatGPT. Retrieved Oct 6, 2023 from https://openai.com/blog/chatgpt
[7] 2023. PreLog repository. Retrieved Oct 6, 2023 from https://github.com/LogIntelligence/PreLog
[8] 2023. Splunk: The Key to Enterprise Resilience. Retrieved Oct 10, 2023 from http://www.splunk.com
[9] Wasi Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2021. Unified Pre-training for Program Understanding and Generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2655–2668.
[10] Paul Barham, Austin Donnelly, Rebecca Isaacs, and Richard Mortier. 2004. Using Magpie for request extraction and workload modelling.. In *Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Vol. 4. 18–18.

[11] Jasmin Bogatinovski, Sasho Nedelkoski, Li Wu, Jorge Cardoso, and Odej Kao. 2022. Failure identification from unstable log data using deep learning. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 346–355.

[12] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[13] Jiaao Chen, Derek Tam, Colin Raffel, Mohit Bansal, and Diyi Yang. 2023. An empirical survey of data augmentation for limited data learning in nlp. *Transactions of the Association for Computational Linguistics* 11 (2023), 191–211.

[14] Mike Chen, Alice X Zheng, Jim Lloyd, Michael I Jordan, and Eric Brewer. 2004. Failure diagnosis using decision trees. In *International Conference on Autonomic Computing, 2004. Proceedings.* IEEE, 36–43.

[15] Xiaolei Chen, Peng Wang, Jia Chen, and Wei Wang. 2023. AS-Parser: Log Parsing Based on Adaptive Segmentation. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 4 (2023), 1–26.

[16] Qian Cheng, Amrita Saha, Wenzhuo Yang, Chenghao Liu, Doyen Sahoo, and Steven Hoi. 2023. LogAI: A Library for Log Analytics and Intelligence. *arXiv preprint arXiv:2301.13415* (2023).

[17] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2022. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416* (2022).

[18] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, and Ziqing Yang. 2021. Pre-training with whole word masking for chinese bert. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 29 (2021), 3504–3514.

[19] Hetong Dai, Heng Li, Che Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Transactions on Software Engineering* (2020).

[20] Anwesha Das, Frank Mueller, Paul Hargrove, Eric Roman, and Scott Baden. 2018. Doomsday: Predicting which node will fail when on supercomputers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 108–121.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 4171–4186.

[22] Ning Ding, Shengding Hu, Weilin Zhao, Yulin Chen, Zhiyuan Liu, Haitao Zheng, and Maosong Sun. 2022. OpenPrompt: An Open-source Framework for Prompt-learning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 105–113.

[23] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*.

[24] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.

[25] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[26] Steven Y Feng, Varun Gangal, Dongyeop Kang, Teruko Mitamura, and Eduard Hovy. 2020. GenAug: Data Augmentation for Finetuning Text Generators. In *Proceedings of Deep Learning Inside Out (DeeLIO): The First Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*. 29–42.

[27] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. Association for Computational Linguistics, Online, 1536–1547.

[28] Qiang Fu, Jian-Guang Lou, Yi Wang, and Jiang Li. 2009. Execution anomaly detection in distributed systems through unstructured log analysis. In *2009 ninth IEEE international conference on data mining*. IEEE, 149–158.

[29] Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. Making Pre-trained Language Models Better Few-shot Learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 3816–3830.

[30] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 6894–6910.

[31] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–28.

[32] Daya Guo, Shuo Ren, Shuai Lu, Zhangyin Feng, Duyu Tang, Shujie LIU, Long Zhou, Nan Duan, Alexey Svyatkovskiy, Shengyu Fu, Michele Tufano, Shao Kun Deng, Colin Clement, Dawn Drain, Neel Sundaresan, Jian Yin, Daxin Jiang, and Ming Zhou. 2021. GraphCodeBERT: Pre-training Code Representations with Data Flow. In *International Conference on Learning Representations*.

[33] Hongcheng Guo, Xingyu Lin, Jian Yang, Yi Zhuang, Jiaqi Bai, Tieqiao Zheng, Bo Zhang, and Zhoujun Li. 2021. Translog: A unified transformer-based framework for log anomaly detection. *arXiv preprint arXiv:2201.00016* (2021).

[34] Haixuan Guo, Shuhan Yuan, and Xintao Wu. 2021. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*. IEEE, 1–8.

[35] Raia Hadsell, Sumit Chopra, and Yann LeCun. 2006. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 2. IEEE, 1735–1742.

[36] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R Lyu. 2017. Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40.

[37] Shilin He, Pinjia He, Zhuangbin Chen, Tianyi Yang, Yuxin Su, and Michael R Lyu. 2021. A survey on automated log analysis for reliability engineering. *ACM computing surveys (CSUR)* 54, 6 (2021), 1–37.

[38] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2016. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 207–218.

[39] Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. 2020. Loghub: a large collection of system log datasets towards automated log analytics. *arXiv preprint arXiv:2008.06448* (2020).

[40] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

[41] Shaohan Huang, Yi Liu, Carol Fung, Rong He, Yining Zhao, Hailong Yang, and Zhongzhi Luan. 2020. Hitanomaly: Hierarchical transformers for anomaly detection in system log. *IEEE Transactions on Network and Service Management* 17, 4 (2020), 2064–2076.

[42] Shaohan Huang, Yi Liu, Carol Fung, He Wang, Hailong Yang, and Zhongzhi Luan. 2023. Improving log-based anomaly detection by pre-training hierarchical transformers. *IEEE Trans. Comput.* (2023).

[43] Yintong Huo, Cheryl Lee, Yuxin Su, Shiwen Shan, Jinyang Liu, and Michael R. Lyu. 2023. EvLog: Identifying Anomalous Logs over Software Evolution. In *34th IEEE International Symposium on Software Reliability Engineering, ISSRE*. IEEE, 391–402.

[44] Yintong Huo, Yuxin Su, Cheryl Lee, and Michael R Lyu. 2023. SemParser: A Semantic Parser for Log Analytics. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 881–893.

[45] Peng Jia, Shaofeng Cai, Beng Chin Ooi, Pinghui Wang, and Yiyuan Xiong. 2023. Robust and Transferable Log-based Anomaly Detection. *Proceedings of the ACM on Management of Data (SIGMOD)* 1, 1 (2023), 1–26.

[46] Tong Jia, Lin Yang, Pengfei Chen, Ying Li, Fanjing Meng, and Jingmin Xu. 2017. Logsed: Anomaly diagnosis through mining time-weighted control flow graph in logs. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*. IEEE, 447–455.

[47] Suhas Kabinna, Cor-Paul Bezemer, Weiyi Shang, Mark D Syer, and Ahmed E Hassan. 2018. Examining the stability of logging statements. *Empirical Software Engineering* 23, 1 (2018), 290–333.

[48] Parisa Sadat Kalaki, Alireza Shameli-Sendi, and Behzad Khalaji Emamzadeh Abbasi. 2022. Anomaly detection on OpenStack logs based on an improved robust principal component analysis model and its projection onto column space. *Software: Practice and Experience* (2022).

[49] Zanis Ali Khan, Donghwan Shin, Domenico Bianculli, and Lionel Briand. 2022. Guidelines for assessing the accuracy of log message template identification techniques. In *Proceedings of the 44th International Conference on Software Engineering*. 1095–1106.

[50] Jaehyung Kim, Dongyeop Kang, Sungsoo Ahn, and Jinwoo Shin. 2021. What makes better augmentation strategies? augment difficult but not too different. In *International Conference on Learning Representations*.

[51] Van-Hoang Le and Hongyu Zhang. 2021. Log-based Anomaly Detection Without Log Parsing. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 492–504.

[52] Van-Hoang Le and Hongyu Zhang. 2022. Log-based anomaly detection with deep learning: How far are we?. In *Proceedings of the 44th International Conference on Software Engineering*. 1356–1367.

[53] Van-Hoang Le and Hongyu Zhang. 2023. Log Parsing: How Far Can ChatGPT Go?. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1699–1704.

[54] Van-Hoang Le and Hongyu Zhang. 2023. Log parsing with prompt-based few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2438–2449.

[55] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Eadro: An end-to-end troubleshooting framework for microservices on multi-source data. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1750–1762.

[56] George Lee, Jimmy Lin, Chuang Liu, Andrew Lorek, and Dmitriy Ryaboy. 2012. The Unified Logging Infrastructure for Data Analytics at Twitter. *Proceedings of the VLDB Endowment* 5, 12 (2012).

[57] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7871–7880.

[58] Xiaoyun Li, Pengfei Chen, Linxiao Jing, Zilong He, and Guangba Yu. 2020. Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 92–103.

[59] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log Clustering Based Problem Identification for Online Service Systems. In *Proceedings of the 38th International Conference on Software Engineering Companion* (Austin, Texas) *(ICSE '16)*. 102–111.

[60] Jinyang Liu, Jieming Zhu, Shilin He, Pinjia He, Zibin Zheng, and Michael R Lyu. 2019. Logzip: Extracting hidden structures via iterative clustering for log compression. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 863–873.

[61] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[62] Yudong Liu, Xu Zhang, Shilin He, Hongyu Zhang, Liqun Li, Yu Kang, Yong Xu, Minghua Ma, Qingwei Lin, Yingnong Dang, et al. 2022. Uniparser: A unified log parser for heterogeneous log data. In *Proceedings of the ACM Web Conference 2022*. 1893–1901.

[63] Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. 2018. Detecting anomaly in big data system logs using convolutional neural network. In *2018 DASC/PiCom/DataCom/CyberSciTech*. IEEE, 151–158.

[64] Ruotian Ma, Xin Zhou, Tao Gui, Yiding Tan, Linyang Li, Qi Zhang, and Xuan-Jing Huang. 2022. Template-free Prompt Tuning for Few-shot NER. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 5721–5732.

[65] Andres Marzal and Enrique Vidal. 1993. Computation of normalized edit distance and applications. *IEEE transactions on pattern analysis and machine intelligence* 15, 9 (1993), 926–932.

[66] Antonio Mastropaolo, Luca Pascarella, and Gabriele Bavota. 2022. Using deep learning to generate complete log statements. In *Proceedings of the 44th International Conference on Software Engineering*. 2279–2290.

[67] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. 2019. Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 4739–4745.

[68] Yu Meng, Chenyan Xiong, Payal Bajaj, Paul Bennett, Jiawei Han, Xia Song, et al. 2021. Coco-lm: Correcting and contrasting text sequences for language model pretraining. *Advances in Neural Information Processing Systems* 34 (2021), 23102–23114.

[69] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. 2018. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 167–16710.

[70] Tomáš Mikolov, Édouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

[71] Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. 2019. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 1795–1812.

[72] Meiyappan Nagappan and Mladen A Vouk. 2010. Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 114–117.

[73] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2020. Self-attentive classification-based anomaly detection in unstructured logs. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1196–1201.

[74] Sasho Nedelkoski, Jasmin Bogatinovski, Alexander Acker, Jorge Cardoso, and Odej Kao. 2021. Self-supervised log parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV*. Springer, 122–138.

[75] Adam Oliner and Jon Stearley. 2007. What supercomputers say: A study of five system logs. In *37th annual IEEE/IFIP international conference on dependable systems and networks (DSN'07)*. IEEE, 575–584.

[76] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H Chin, and Sumayah Alrwais. 2015. Detection of early-stage enterprise infection by mining large-scale log data. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems*

and Networks. IEEE, 45–56.

[77] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Association for Computational Linguistics, Minneapolis, Minnesota, 48–53.

[78] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International conference on machine learning*. PMLR, 8748–8763.

[79] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).

[80] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[81] Keiichi Shima. 2016. Length matters: Clustering system log messages using length of words. *arXiv preprint arXiv:1611.03213* (2016).

[82] Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. 2023. LogKG: Log Failure Diagnosis through Knowledge Graph. *IEEE Transactions on Services Computing* (2023).

[83] Shimin Tao, Weibin Meng, Yimeng Cheng, Yichen Zhu, Ying Liu, Chunning Du, Tao Han, Yongpeng Zhao, Xiangguang Wang, and Hao Yang. 2022. Logstamp: Automatic online log parsing based on sequence labelling. *ACM SIGMETRICS Performance Evaluation Review* 49, 4 (2022), 93–98.

[84] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[85] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A unified multi-tasking model for supporting matching tasks in data integration. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[86] Risto Vaarandi. 2003. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*. Ieee, 119–126.

[87] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[88] Chaozheng Wang, Yuanhang Yang, Cuiyun Gao, Yun Peng, Hongyu Zhang, and Michael R Lyu. 2022. No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence. In *Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering*. 382–394.

[89] Xuheng Wang, Xu Zhang, Liqun Li, Shilin He, Hongyu Zhang, Yudong Liu, Lingling Zheng, Yu Kang, Qingwei Lin, Yingnong Dang, et al. 2022. SPINE: a scalable log parser with feedback guidance. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1198–1208.

[90] Yue Wang, Weishi Wang, Shafiq Joty, and Steven C.H. Hoi. 2021. CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8696–8708.

[91] Jason Wei and Kai Zou. 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 6382–6388.

[92] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45.

[93] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. 2020. Clear: Contrastive learning for sentence representation. *arXiv preprint arXiv:2012.15466* (2020).

[94] Junjielong Xu, Ruichun Yang, Yintong Huo, Chengyu Zhang, and Pinjia He. 2024. DivLog: Log Parsing with Prompt Enhanced In-Context Learning. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 983–983.

[95] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 117–132.

[96] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Shaochen Zhong, Bing Yin, and Xia Hu. 2024. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. *ACM Trans. Knowl. Discov. Data* (feb 2024). https://doi.org/10.1145/3649506 Just Accepted.

[97] Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. 2021. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1448–1460.

[98] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[99] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1253–1263.

[100] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. 2019. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130.

[101] Yichen Zhu, Weibin Meng, Ying Liu, Shenglin Zhang, Tao Han, Shimin Tao, and Dan Pei. 2021. Unilog: Deploy one model and specialize it for all log analysis tasks. *arXiv preprint arXiv:2112.03159* (2021).

Received October 2023; revised January 2024; accepted February 2024