electronics

MDPI

Article

IPLog: An Efficient Log Parsing Method Based on Few-Shot Learning

Shuxian Liu, Libo Yun *, Shuaiqi Nie, Guiheng Zhang and Wei Li

School of Computer Science and Technology, Xinjiang University, Urumqi 830046, China; liushuxian@xju.edu.cn (S.L.); 107552204054@stu.xju.edu.cn (S.N.); 107552204118@stu.xju.edu.cn (G.Z.); liwei123@xju.edu.cn (W.L.)
* Correspondence: ylb@stu.xju.edu.cn

Abstract: Log messages from enterprise-level software systems contain crucial runtime details. Engineers can convert log messages into structured data through log parsing, laying the foundation for downstream tasks such as log anomaly detection. Existing log parsing schemes usually underperform in production environments for several reasons: first, they often ignore the semantics of log messages; second, they are often not adapted to different systems, and their performance varies greatly; and finally, they are difficult to adapt to the complexity and variety of log formats in the real environment. In response to the limitations of current approaches, we introduce IPLog (Intelligent Parse Log), a parsing method designed to address these issues. IPLog samples a limited set of log samples based on the distribution of templates in the system's historical logs, and allows the model to make full use of the small number of log samples to recognize common patterns of keywords and parameters through few-shot learning, and thus can be easily adapted to different systems. In addition, IPLog can further improve the grouping accuracy of log templates through a novel manual feedback merge query strategy based on the longest common prefix, thus enhancing the model's adaptability to handle complex log formats in production environments. We conducted experiments on four newly released public log datasets, and the experimental results show that IPLog can achieve an average grouping accuracy (GA) of 0.987 and parsing accuracy (PA) of 0.914 on the four public datasets, which are the best among the mainstream parsing schemes. These results demonstrate that IPLog is effective for log parsing tasks.

Keywords: log parsing; prompt tuning; manual feedback; computer science and engineering

Citation: Liu, S.; Yun, L.; Nie, S.; Zhang, G.; Li, W. IPLog: An Efficient Log Parsing Method Based on Few-Shot Learning. *Electronics* **2024**, *13*, 3324. https://doi.org/10.3390/electronics13163324

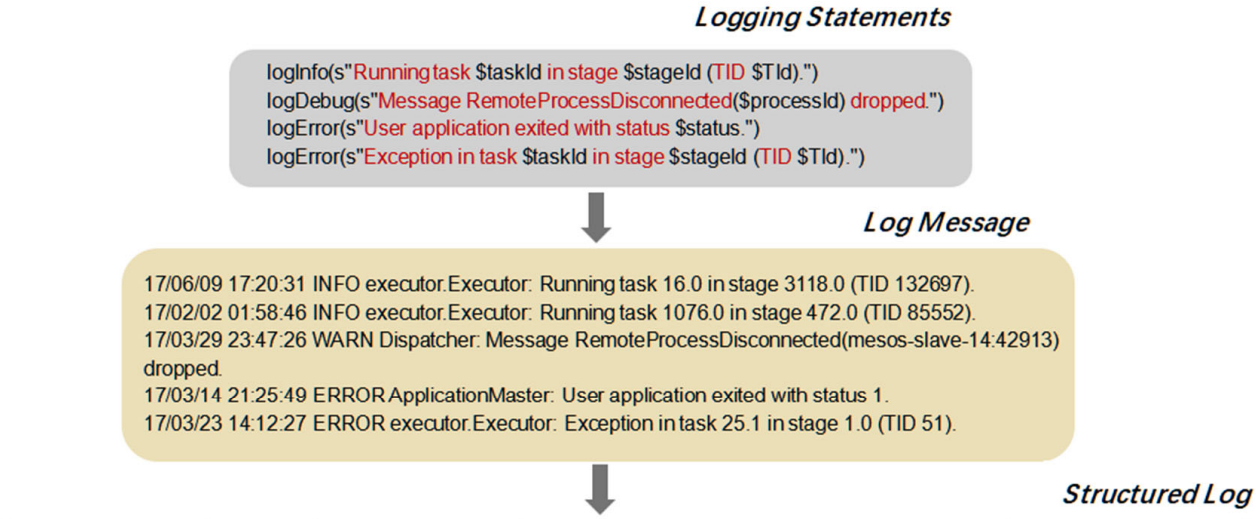
Received: 14 July 2024
Revised: 18 August 2024
Accepted: 20 August 2024
Published: 21 August 2024

1. Introduction

Logs are mainly used to record the state of a system during operation, and engineers can utilize these logs for fault prediction [1,2] or system anomaly detection [3–6]. In recent years, online services and system software have become an indispensable part of our daily life, and they generate a huge number of software logs every hour, so it is impossible to ensure the quality of service by manually analyzing so many logs. In order to improve the diagnostic efficiency in cases of system anomalies and to reduce the labor cost, the task of automated log parsing has received a lot of attention. The core of this task is to transform the unstructured raw log information into a structured data format, which in turn can be used to feed into various deep learning or machine learning models in order to perform various subsequent missions. Such automated processing significantly speeds up the response time of engineers when dealing with anomalies in engineering projects.

Log parsing involves transforming raw log messages into predefined log templates. Log messages originate from log statements within the source code. Typically, as illustrated in Figure 1, a log message consists of a log header, generated by the logging system, and a log message body. The log header contains information such as timestamps, level of detail, and components, while the log message body typically comprises two parts: a template

constant string (keyword) depicting system incidents, and a parameter (variable), which is constantly changing at runtime, reflecting the detailed information of the system at runtime. In one of the log messages in Figure 1, the log header is obtained after processing (i.e., "17/06/09 17:20:31", "INFO", and "executor. Executor"), and the log message body consists of the template "Running task <*> in slabele <*> (TID <*>)" and a list of "16.0", "3118.0", and "132697".



Datetime	Level	Component	Log Template	Parameters
17/06/09 17:20:31	INFO	executor.Executor	Running task <*> in stage <*> (TID <*>)	16.0, 3118.0, 132697
17/02/02 01:58:46		executor.Executor	Running task <*> in stage <*> (TID <*>)	1076.0, 472.0, 85552
17/03/29 23:47:26	WARN	Dispatcher	Message RemoteProcessDisconnected(<*>) dropped	mesos-slave-14:42913
17/03/14 21:25:49	ERROR	ApplicationMaster	User application exited with status <*>	1
17/03/23 14:12:27		executor.Executor	Exception in task <*> in stage <*> (TID <*>)	25.1, 1.0, 51

Figure 1. Example of log parsing.

Over the years, several data-driven methods [7,8] have been developed to facilitate automated log parsing. These methods aim to identify the repetitive common elements of log messages as templates and to extract the variable elements as parameters. Despite the progress made, existing log parsing methods are still criticized for their unsatisfactory results in production environments, which may have a significant negative impact on subsequent tasks such as anomaly detection [9,10]. We pinpoint three main factors contributing to inaccurate parsing results. Firstly, existing log parsing techniques focus solely on extracting common elements as templates, overlooking the semantic content of logs. For example, as shown in Figure 1: considering the semantics of log messages, "mesos-slave-14:42913" should clearly be recognized as a parameter. Here, "mesos-slave-14:42913" refers to the identity of a specific Mesos slave node. However, without considering the semantics, existing log parsing methods often fail to recognize the noun part of "mesos-slave-14:42913", and thus incorrectly consider this parameter part of the template. Secondly, the log contents vary significantly across different services and systems. Log events and descriptions generated by diverse systems differ greatly, posing a challenge to the generalization of log parsing methods across various environments. Thirdly, existing log parsing methods are difficult to adapt to the complexity and variety of log formats in real-world environments, where many log statements change throughout the life cycle of a software system. This instability stems from the fact that developers add new log statements or modify existing log content to accommodate new features or bug fixes during ongoing development and maintenance, which leads to a significant increase in the variety and complexity of log templates, making it difficult for existing parsing methods to cope.

To address the above problem, we propose a new log parsing method, IPLog. In this work, we argue that logs are always heavily repetitive during generation regardless of the system, and that this repetitiveness causes the model to ignore truly valuable information. Based on this thinking, we propose a new adaptive system sampling method to select a small number of training data from historical logs, and then learn from this small number of data to grasp the semantic content of log messages, enabling the identification of parameters and keywords within them. Second, we use the template-free prompt tuning method [11] to adjust a pretrained model to forecast a particular virtual label word for parameter locations. The embedding vector for the synthetic label "Variable" is derived from the word "distribution" calculated by the unlabeled log dataset and the language model. After training, the model is capable of directly parsing new log data. To further improve the performance of the parsing method on different systems, we finally propose a postprocessing method for a manual feedback merge query strategy based on the longest common prefix, which allows the model to achieve a sizable increase in performance with very little human involvement. Finally, we demonstrate the superiority of IPLog by evaluating it on four newly released public log datasets.

In summary, our contributions are as follows.

In this article, we design a new adaptive system sampling method that can take a small and representative set of training samples for different systems as a way to help the model better understand the differences between parameters and templates, and thus extract templates more accurately.

We present IPLog, a few-shot log parsing method based on prompt tuning that accurately identifies patterns of log messages. IPLog employs an innovative template-free prompt tuning approach to efficiently grasp semantic content from a limited set of labeled log samples. This technique eliminates the need for manually defining regular expressions for preprocessing, allowing for rapid adaptation to new logs. In addition, we design a postprocessing method for merging queries with manual feedback based on the longest common prefix, which effectively improves the model performance at a fraction of the cost.

We assess IPLog on four public log datasets and demonstrate that it surpasses existing methods. The experimental results verify the effectiveness of our proposed approach.

2. Background and Motivation

2.1. Log Parsing

Log records are critical to system maintainers for diagnosing issues and monitoring performance. Log parsing, a key step in these tasks, aims to transform raw, semi-structured or unstructured log records into a structured format for further analysis and processing. With structured log records, system maintainers can detect and troubleshoot problems more efficiently while optimizing system performance to ensure stable system operation. Log parsing is a process of extracting static log templates and parameters and processing them [12].

2.2. Related Work

Regular expression filtering was the earliest method used for log parsing, but it requires a large amount of manual work, making it difficult to cope with the rapid growth in log volume in software-intensive systems. For efficient log parsing, various data-driven techniques have been developed to transform raw log messages into templates and parameters. The classification of methods and the technical features of each method are shown in Table 1. These methods can be generally classified into four primary categories: similarity-based clustering, frequent pattern mining, heuristic methods, and neural networks.

Table 1. Overview of log parsing methods.

Category	Method	Feature
Frequent pattern mining	SLCT	Frequent word matching
	LFA	Frequency and regular expression
	LogCluster	Hash table clustering
	Logram	N-gram dictionary
Similarity-based clustering	LKE	Edit distance clustering
	LenMa	Incremental clustering
Heuristics	AEL	Clone detection
	Drain	Parsing tree
	Spell	LCS mapping
Neural network	Nulog	Transformer encoder
	Semlog	Semantic contribution scoring
	LogPPT	Pretrained model

Frequent Pattern Mining: Intuitively, the public portion of a log should frequently appear throughout the log dataset. Therefore, frequent pattern mining techniques exploit this feature. Typical approaches include SLCT [13], LFA [14], LogCluster [15] and Logram [8]. These methods first traverse the log data and construct frequent itemsets based on labels, labeled position pairs, or labeled n-tuple models. A given frequent itemset can then group log messages into multiple clusters, and log templates can be extracted from the clusters. SLCT was the first to apply frequent pattern mining to log parsing [12]. For SLCT, it not only applies frequent pattern mining to log parsing but also constructs clustering candidates by scanning the input dataset multiple times, each time by extracting words that occur more frequently than a predefined threshold. LFA and LogCluster build on this by further considering the frequency distribution of tokens and the locations of the tokens. LFA not only considers the frequency distribution and location of tokens but also recognizes the constant and variable portions of log messages to construct event types as regular expressions. LogCluster uses a hash table to locate frequent words and extracts all frequent words from each log message to build or update candidate groups. Logram aims to extract frequent 2 g and 3 g tokens. This process typically involves building a word frequency table that records the number of occurrences of each phrase. Logram then uses these frequent phrases to build a template for the log, inferring the basic structure of the log message by recognizing and combining common phrases.

Similarity-Based Clustering: This class of parsing methods assumes that logs belonging to the same log template can be clustered together by certain features. LKE [16] uses a hierarchical clustering algorithm with custom-weighted edit distances. This weighting reflects the importance of the different log sections, allowing the algorithm to more accurately distinguish between log templates. LenMa [17] focuses on word-length features and converts logs into vectors representing the number of letters in a word. This approach clusters log messages by calculating the similarity between them or their signatures.

Heuristics: Log messages differ from general text data due to their unique characteristics. Consequently, some log parsing techniques leverage these traits to extract common segments as templates. AEL [18] classifies log messages by comparing the frequency of occurrence of constant and variable lexical elements in the logs, effectively identifying and grouping logs with similar patterns. Drain [7] borrows the idea of prefix trees to group logs by parsing the logs online using a tree structure with a fixed depth. Spell [19] uses the longest common subsequence algorithm to analyze logs, identify recurring patterns in logs, and extract generic log templates from them.

Neural Network: Nulog [20] uses the transformer encoder to classify mask words one by one for log parsing. After encoder processing, a linear layer converts the output matrix into a vector representation of the log messages, a representation that captures the key features and intrinsic structure of the log data. NuLog identifies the variable portions of the logs by masking operations on each lexical element and generates the appropriate event templates accordingly. Semlog [21] trains BERT-based models to enhance the semantic

distinction between constants and variables. During the online parsing phase, Semlog finally applies a template extraction algorithm to determine the log template for each log based on the computed semantic contribution score. In LogPPT [22], an adaptive random sampling algorithm was designed to select a diverse training set, which is then used to tune the pretrained model. This allows for the prediction of log templates and parameters.

2.3. Limitations

Despite the progress made, existing log parsing methods still suffer from low parsing accuracy in production environments, which may seriously affect subsequent analyses such as anomaly detection based on logs. In general, existing log parsing methods suffer from the following limitations:

Ignoring semantics: Log parsing methods typically focus on extracting structured data from logs, but this approach can ignore the deeper meaning of log content. For example, an error log may contain key performance metrics or configuration information that may be missed in a simple text extraction process. Traditional log parsing methods often misidentify parameters as keywords, especially when semantic information is not taken into account. For example, similarity-based clustering, frequent pattern mining, and heuristic methods often rely on syntactic rules that fail to capture semantic information in logs, leading to poor parsing results. Specifically, SLCT, LFA, LogCluster, and Logram address data only from a statistical perspective, focusing on frequently occurring patterns. LKE and LenMa concentrate on structural similarities, while AEL and Drain employ heuristic rules and prefix tree structures for rapid log template extraction. Spell relies on the longest common subsequence to process logs. These methods generally focus on the format and syntactic features of logs. The phenomenon of misidentifying variables as templates is particularly evident in these methods that do not take advantage of semantic information. Due to the inability to accurately identify templates and parameters in logs, these methods result in suboptimal performance in terms of log parsing accuracy.

Lack of generalizability: Logs generated by different systems, applications, or devices may have a high degree of diversity, including different formats, terminology, and logging conventions. The challenge for a generalized log parsing method is that it needs to be flexible enough and broadly compatible to accommodate these diverse formats. While some parsing methods may perform well with specific types of log files, their performance may degrade significantly across systems or applications. For example, a parsing method designed for a particular application may not be able to accurately process logs from another system because differences in the representation of key data between the two may result in parsing errors and missing information. The limitation of such approaches is that they typically need to be adapted or reconfigured for each new log type, adding complexity and cost to maintenance. For similarity-based clustering, frequent pattern mining, and heuristic methods, specific adjustments are often necessary to effectively handle log data from different systems, limiting their applicability across diverse log formats or systems. For instance, SLCT requires presetting the regular expression "[r'blk_-?\d+', r'(\d+\.){3}\d+(:\d+)?']" to match block IDs and IP addresses before parsing HDFS system logs. However, in the Zookeeper system, where the focus is more on paths and IP addresses, the regular expression must be changed to "[r'(/|)(\d+\.){3}\d+(:\d+)?']" for correct matching. This also means that rule-based methods, to be effective across different systems, must rely on domain knowledge to craft specific regular expressions, undoubtedly limiting their universality across various systems.

Poor adaptability: As technology evolves and systems are upgraded, the emergence of complex log formats is inevitable. Traditional log parsing methods often struggle to adapt promptly to these changes, especially when new log patterns or types are introduced. For example, the introduction of new software modules or updates to existing systems may generate entirely new types of log entries. If the parsing method fails to recognize these new patterns, it may miss critical information or produce incorrect parsing results. In real production environments, the complexity of log templates and parameters also becomes

more complex over time, and traditional parsing methods have difficulty adapting to such changes. Currently, almost all methods struggle to easily adapt to new log formats and content. Specifically, for similarity-based clustering, frequent pattern mining, and heuristic methods, the regular expressions or parameters established for systems no longer suffice with system updates, requiring continual manual revisions based on newly emerged logs, thereby reducing parsing performance. NuLog, SemLog, and LogPPT typically rely on neural network technology to parse log data. Although these methods excel in handling logs with known formats and content compared to traditional log parsing techniques, they still encounter accuracy issues when faced with new log templates, leading to incorrect template grouping and adversely affecting log parsing performance.

2.4. Insights and Opportunities

The three aforementioned issues prompted us to rethink our approach to log parsing. By examining logs from various public datasets and industrial sources, we identified a crucial feature of log data: despite the diverse content across different systems or services, logs typically adhere to specific logging conventions. Inspired by the use of virtual labeling tokens to identify keywords and parameters in LogPPT [22], we accomplish the goal of log parsing by selecting a comprehensive sample from historical logs labeled as training data. We then use a prompt tuning strategy. We reformulate the task of log parsing into a predictive labeling problem. The underlying premise is that the majority of keywords in log statements consist of valid, intelligible words readily accessible in dictionaries [23], making them easier for a language model to predict. In contrast, the parameters are ever-changing, making it difficult for a language model to predict. In this process, we set up a strategy such that the model uses the specific virtual label word "Variable" for parameter identification, while keeping the original text predicted for keywords in the log. Unlike traditional prompt tuning methods, we adopt a template-free prompt tuning strategy [11], which does not need to rely on predefined prompt templates to guide the model. Hence, in the process of log parsing, the model is primed to anticipate the virtual label "Variable" precisely where parameters occur, while preserving their original manifestation at keyword positions.

3. Method

This section details our approach, which aims to overcome the limitations of traditional methods. Our model leverages the extensive knowledge embedded in pretrained language models to detect patterns in parameters and templates within the contextual aspects within log messages. By utilizing a prompt tuning technique, the model can effectively perform log parsing with a limited number of training samples, transferring the language model's knowledge to the task. To maximize the benefits of prompt tuning, selecting an appropriate training set is essential. Therefore, we have developed a new adaptive system sampling method to precisely choose training samples, enhancing both training efficiency and effectiveness. In this section, we arrange it as follows: we firstly state the problem definition in Section 3.1, describe the data sampling methodology in Section 3.2, detail the IPLog core model methodology design in Section 3.3, describe how to implement the IPLog and how to parse online in Section 3.4, and finally introduce a manual feedback merge query method in Section 3.5.

3.1. Definition of the Problem

In this research, we metamorphose the log parsing task into a challenge of parameter recognition by employing a novel prompt tuning method that is trained with a limited set of labeled samples. To handle a dataset D, we customize a pretrained model M through prompt tuning to distinguish between templates and variables in log messages. The model processes an initial log message comprising n tokens T = $\{t_1, t_2 \ldots, t_n\}$. At parameter

positions, the model predicts the virtual label "Variable" while for keywords, it predicts the initial word. Formally, model M is trained to generate output O = $\{o_1, o_2 \ldots, o_n\}$, where:

$$o_i = M(t_i) = \begin{cases} \text{"Variable" if } t_i \text{ is one of the variables} \\ t_i \text{ if } t_i \text{ is part of the template} \end{cases} \quad (1)$$

Figure 2 illustrates the model's training process to predict parameters "3357" and "1343" using the label word "Variable". For keywords like "child", the model predicts the original word. "Variable" is a specific placeholder word with no inherent meaning. The model identifies parameters within the log message and recognizes them through their association with the word "Variable". The embedding vector for "Variable" is computed based on the most commonly occurring parameters in the log message, enhancing the expression of their meaning. During the online parsing phase, all instances of "Variable" are considered parameters, while other words form the log template.

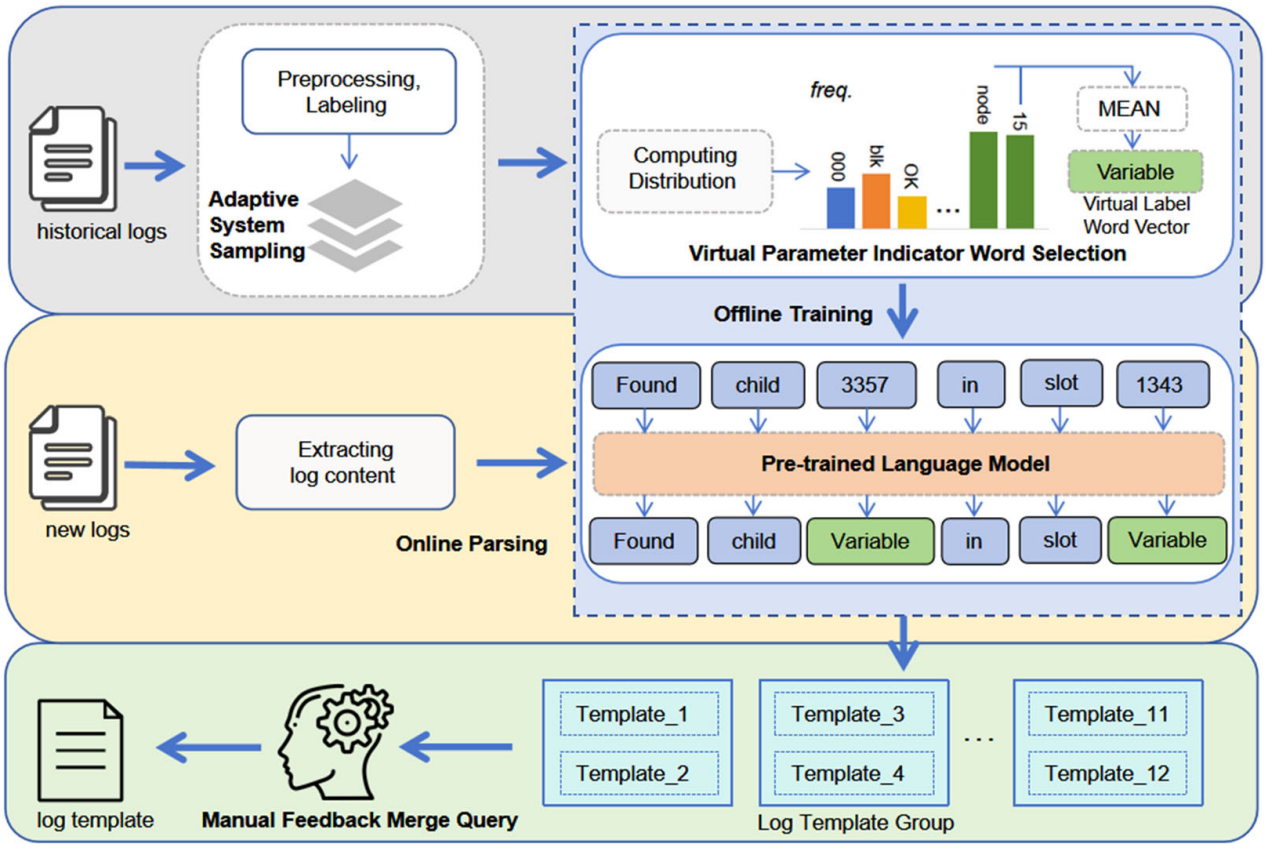


Figure 2. Overview of IPLog (where the top gray box represents the steps in the offline training phase of IPLog, the blue box is the process of prompt tuning, the yellow box represents the steps of online parsing, and the bottom green box is the postprocessing process).

This shift allows the model to leverage existing pretraining knowledge and effectively identify and differentiate key information in logs with a small amount of labeled data. With this approach, we significantly reduce our reliance on large amounts of labeled data while maintaining a high level of accuracy, which is especially critical when dealing with large-scale or dynamically changing log data.

3.2. Adaptive System Sampling

During the training phase, our method necessitates only a sparse set of labeled log data to establish the training dataset. In order to improve the parsing of the model, we design a simple and efficient strategy, the adaptive system sampling algorithm, to pick a

small but comprehensive collection of labeled samples for the current system. In adaptive system sampling, the algorithm first parses historical log data to identify the kinds of log templates that the current system already has. Each template represents the format of a particular class of events or behaviors in the system's logs, and the distribution of these templates may vary significantly across systems. This analysis enables the sampling strategy to understand which templates are common in the current system and which are rarer. During the sampling process, the adaptive system sampling algorithm will adjust the selected samples based on the type and frequency of occurrence of the templates. For high-frequency logs, frequent occurrences may indicate that they are critical to the normal operation of the system, and thus sampling from these templates can help the model to better learn the normal behavioral patterns of the system. For less frequent logs, on the other hand, although they appear infrequently, they may be associated with critical events or abnormal states in the system. Identifying and training such templates can greatly improve the accuracy and robustness of the model in dealing with anomalies, and adaptive system sampling can effectively improve the quality of the training data.

The adaptive system sampling method not only improves the comprehensiveness and diversity of the training data but also significantly reduces the training cost and time by selecting a small number of the most informative samples. When a new log template is added to the historical log, the method can adaptively update the number of samples selected, which allows the model to easily adapt to the new log format.

3.3. Log Parsing Based on Prompt Tuning

In this study, we apply prompt tuning techniques that set new standards in a variety of natural language processing (NLP) tasks, aiming to achieve the goal of entity-oriented language modeling. The main rationale is that keywords in logs are usually valid and easy-to-understand words that can be found in the lexicon, making it easier for language models to make predictions; on the contrary, parameters in logs change frequently and are more difficult for language models to predict. Based on this observation, we transform the log parsing task into a label word prediction problem. Specifically, we let the model predict parameters as virtual label words "Variable" and keywords as original words.

3.3.1. Pretrained Language Model

Over the last few years, pretrained language models have demonstrated significant effectiveness in numerous NLP tasks [24–26]. Typically, these models undergo initial pretraining on extensive corpora of unlabeled text, followed by fine-tuning for specific downstream missions. Recent research has shown [10] that pretrained models can effectively parse the semantics of log messages and provide support for various log analysis tasks. In this paper, we adopt the popular RoBERTa [24] model for our study. RoBERTa utilizes solely the encoder component and adopts the identical transformer architecture as BERT. Both RoBERTa and BERT utilise the byte-pair encoding (BPE) strategy for vocabulary processing, which avoids the introduction of unknown words by subdividing uncommon words into sub-words. The main reason for choosing RoBERTa over BERT is that RoBERTa improves the performance of the model by scaling up the training data and optimizing the training strategy. RoBERTa uses larger training datasets, longer training time and larger batch size, and removes the next-sentence prediction (NSP) task. This quality renders RoBERTa exceptionally apt for log parsing, since developer-defined parameters frequently consist of word combinations that extend beyond standard English vocabulary, such as the parameter "mesos-slave-14:42913" in Figure 1. Example of log parsing studies have shown [27] that RoBERTa performs well in log analysis.

3.3.2. Virtual Parameter Indicator Word Selection

In this study, we use the template-free prompt tuning technique [11] to predict a virtual label word "Variable" for each parameter position in the input sequence T = $\{t_1, t_2 \ldots, t_n\}$.

Since all parameters are converted to the same word, it becomes particularly important to identify some indicator words that can effectively represent these parameters.

Using the training dataset $D_{train} = \{(T_i, O_i)\}_{i=1}^{K}$, we apply a pretrained language model M to analyze the probability distribution of the predicted likelihood tokens at each position i. We compute the output of the model M for each sample pair (T, O) to obtain the probability distribution p of the individual tokens x in the prediction log message T. To proceed, at each location i identified as a parameter, we select the top k predicted tokens to initialize the set of parameter indications, as P_{ini}. This step aims to broaden the parameter indication set by selecting tokens that closely align in meaning with the original parameter tokens.

Based on the initial set of parameter indications P_{ini}, we then look for the most frequent words in the untagged data. We calculate the frequency $\varnothing(t = x|D)$ of occurrence of each token x in the set and select the most common words by sorting them according to frequency:

$$P = \underset{x}{\text{argmax}} \varnothing(t = x|D), \forall x \in P_{ini} \quad (2)$$

Upon acquiring the set P, we calculate the mean vector of all the tokens in the set P and add this vector as an embedding vector into the language model M to set the embedding vector for the virtual label word "Variable". This approach not only enhances the ability of the model to identify the parameters but also optimizes the overall model performance. In addition, this approach enables the model to better capture the parameter patterns in the log data, which improves the accuracy and efficiency of parsing.

3.3.3. Offline Training

During the training phase, we fully utilize historical logs for training. We use log messages $T = \{t_1, t_2 \ldots, t_n\}$ as input and construct the target sequence $O = \{o_1, o_2 \ldots, o_n\}$ by substituting the parameter at position i with the virtual label word "Variable" while retaining the original word at the keyword position. Identification is then performed according to Equation (1). As shown in Figure 3, we take the log message "Adding an already existing block blk_-2074647" as the input, and at the same time, construct the target sequence "Adding an already existing block <*>", where we denote the position i where the virtual label word is located by <*>.

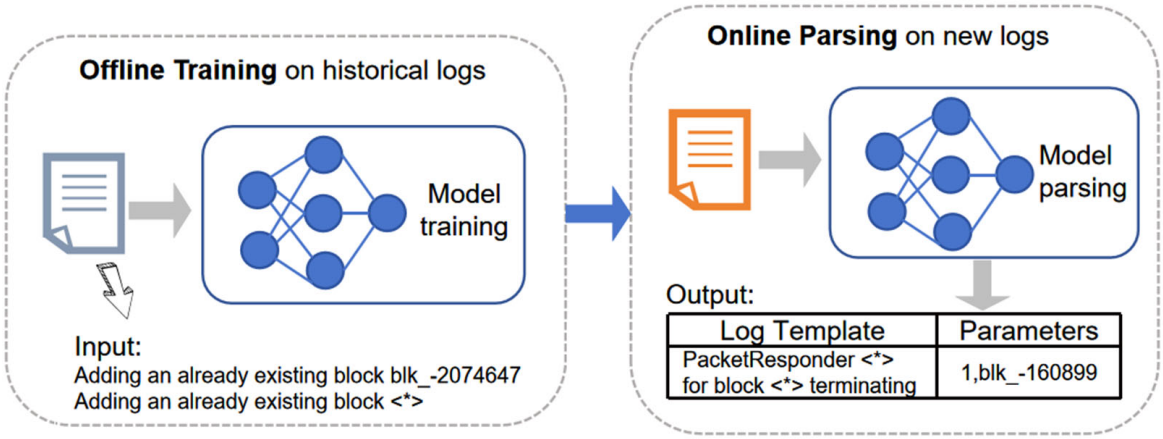


Figure 3. Offline training phase (**left**) and online parsing phase (**right**) of IPLog, where the offline-trained model can be directly used for online parsing.

It is important to emphasize that the pretrained model is completely reused during the adaptation process. This entity-focused objective, similar to the mask token prediction in language models, minimizes the discrepancy between fine-tuning and pretraining. Thus, the model is able to retain the knowledge gained from the pretrained language model. In addition, with this approach, we not only maintain the model's consistency across tasks but

also ensure its efficiency and accuracy when processing log data. In this way, our model is able to flexibly use the pretrained knowledge in different application scenarios to further enhance its parsing and prediction capabilities.

3.4. Online Parsing

In the online parsing phase, the trained model efficiently processes incoming log data in real time. Initially, the model takes the incoming log messages and splits them into separate tokens by performing a parsing process. Subsequently, each token is analyzed and the model predicts whether it is marked as parameters or just plain keywords. This approach enables the model to instantly identify and differentiate between various elements of the log in practice. It not only identifies which parts are dynamically changing parameters but also pinpoints fixed template parts, allowing for effective structuring and classification of log messages. Instead of choosing to represent consecutive parameters as just <*>, as most parsing methods do, we replaced each parameter with <*> after parsing it out, which facilitates more differentiation when analyzing log templates at a later stage. As shown in Figure 3, in the face of the new log message "PacketResponder 1 for block blk_-160899 terminating", IPLog can accurately identify the parameter "1" and the parameter "blk_-160899".

This real-time parsing capability greatly improves the speed and accuracy of log data processing, enabling system administrators or automated monitoring tools to quickly respond to information displayed in the logs, such as detecting potential system problems or abnormal activity.

3.5. Manual Feedback Merge Query

Inspired by Hue [28], we designed and implemented an interactive function for manual feedback to enhance the model's adaptability to changing and complex log templates. This feature is implemented as follows. After the model automatically extracts log data and forms preliminary log templates to form template groups, the operator has the option to further improve the grouping accuracy by merging the templates. These generated templates are first analyzed, and then similar templates are identified by matching the longest common prefix between different templates. The maintenance personnel can view the similarity information of these templates on the user interface and artificially choose whether to merge these templates according to the actual situation and maintenance requirements.

As shown in Figure 4, for the templates with the same public prefix "Error reading message prefix on socket to <*>", the maintenance personnel can determine whether the current merge query is correct or not through the system, and if the current merge does not meet the requirements, they can choose to skip it. For the "ambient=<*>" merge request in Figure 4, it is approved, and the two templates are merged into a unified format. Human feedback allows the maintenance staff to adjust the clustering results based on actual experience and log specifics, correcting potential misclustering cases and ensuring that the clustering results match the actual application scenarios better. During the merging process, the maintenance staff can effectively improve the model's performance on complex log formats at minimal cost.

User choices and feedback are recorded by the system and used to adjust the model's parsing strategy over time, allowing the model to better adapt to changes in log format. This continuous feedback loop ensures system optimization, enhancing parsing accuracy and template clustering accuracy.

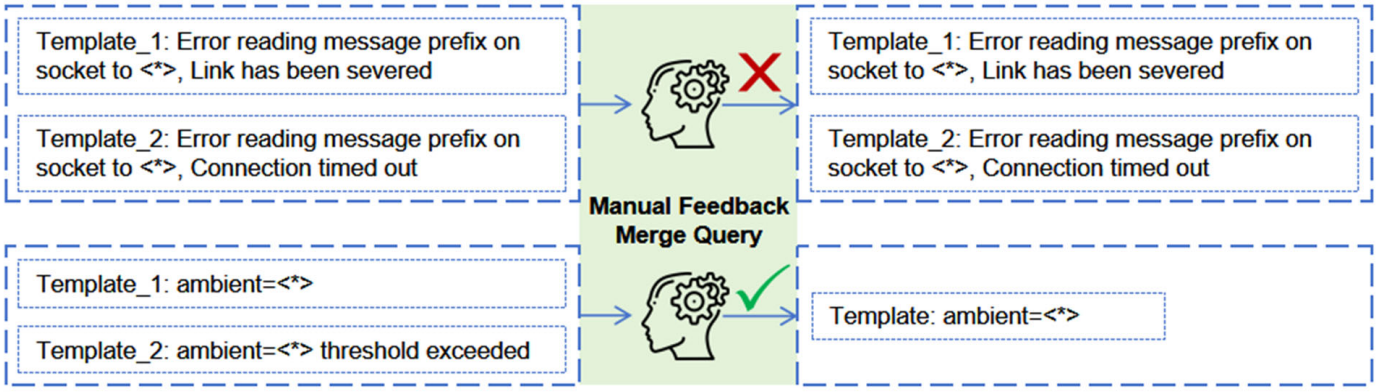


Figure 4. Manual feedback merge query.

4. Experimental Design

In order to evaluate the effectiveness and efficiency of IPLog, we conducted a number of experiments. In this section, we first describe the information of the dataset in Section 4.1, then we will present our experimental environment in Section 4.2, the evaluation metrics used in our experiments will be presented in Section 4.3, and the baselines we compared will be presented in Section 4.4.

4.1. Datasets

Our experiment primarily utilizes the newly released loghub2.0 [29] by LogPai, which encompasses log data from supercomputers, distributed systems, and server applications. Compared to the data from loghub-2k [30], which was commonly used in previous research, loghub2.0 more accurately reflects log data observed in real-world software systems, and the challenges of using the loghub2.0 dataset become more difficult, both in terms of the complexity of the templates and the size of the data. Each system in loghub-2k contains only 2000 logs, and the variety of log templates is far less compared to loghub2.0. Detailed system information for our current study is shown in Table 2, where Hadoop is a big data processing framework that allows distributed processing of large datasets across computer clusters using a simple programming model. Hadoop has been extensively studied in the literature due to its growing importance in industry. HDFS is a distributed file system designed to run on commercial hardware. It is one of the core components of Hadoop and exists as the lowest-tier distributed storage service. ZooKeeper is an open source service managed by the Apache Software Foundation. It is primarily used to orchestrate services in distributed applications. Apache is open-source web server software developed by the Apache Software Foundation, and is one of the most widely used web servers on the internet due to its cross-platform nature, stability, and scalability.

Table 2. Details of the datasets (including 3 distributed systems and 1 service application).

		loghub2.0		loghub-2k	
System	**Dataset**	**Templates**	**Logs**	**Templates**	**Logs**
Distributed systems	Hadoop	236	179,993	114	2000
	HDFS	46	11,167,740	14	2000
	Zookeeper	89	74,273	50	2000
Server application	Apache	29	51,977	6	2000

4.2. Implementation Details

All our experiments were conducted on an Ubuntu 20.04 server equipped with an Nvidia RTX 4090 (24 GB) GPU and an Intel(R) Xeon(R) Platinum 8352 V CPU. Our CUDA version is 11.3 and the code is based on PyTorch 1.10.0 and Python 3.8 implementation. We

utilized the AdamW [31] optimizer and set the initial learning rate to 0.00005, the batch size to 8, and performed a total of 200 steps of training. Also, we employed a linear learning rate decay strategy and implemented a 10% warm-up step at the beginning of training. In the online parsing phase, the batch size is increased to 32, and in the adaptive system sampling phase, we set the number of samples for each template to 1. For the virtual label word "Variable", we compute the embeddings based on the 8 most frequent label words in the experiment.

4.3. Evaluation Metrics

Group Accuracy (GA): The GA metric assesses the capability to accurately group log messages that belong to the same template. The GA metric quantifies the effectiveness of log message grouping by calculating the proportion of correctly categorized messages among the total. It hinges on the coherence between a message's template and a cluster of messages sharing identical underlying facts. Only messages with templates aligning precisely with such clusters are deemed correctly grouped. This metric's emphasis lies in ensuring robust categorization that reflects the nuanced relationships within log data. If we represent CG as the number of correctly grouped log messages and T as the total number of log messages, the exact formula is as follows:

$$\mathrm{GA} = \frac{\mathrm{CG}}{\mathrm{T}} \tag{3}$$

Parsing Accuracy (PA): The PA metric evaluates the accuracy of log message parsing by measuring the percentage of correctly parsed messages among the total. It defines a message as "correctly parsed" if every token is accurately identified as either a template or parameter. This metric is more stringent than group accuracy, because a single misclassified token can lead to the entire message being parsed incorrectly, underscoring its rigorous assessment of parsing precision. The PA used evaluates the ability to correctly extract the template portion and parameter portion of each log message, which is crucial for various log analysis tasks. If we represent CP as the number of log messages parsed correctly and T as the total number of log messages, the exact formula is given below:

$$\mathrm{PA} = \frac{\mathrm{CP}}{\mathrm{T}} \tag{4}$$

4.4. Baselines

We compare IPLog with six other methods, comprising the most common methods currently available in the log parsing domain—LFA [15], LenMa [17], Spell [19], Drain [7], SLCT [13]—and the most recent one proposed in 2023, LogPPT [22]. We choose these six methods for evaluation because not only do they have publicly available source code, but these methods are representative of various schemes such as similarity-based clustering (LenMa), frequent pattern mining (LFA and SLCT), heuristics methods (Drain and Spell), and neural network-based methods (LogPPT). In addition, we refer to the evaluation results and recommendations for each parsing method published by LogPai [30,32–34] to make our experiments richer.

5. Results

5.1. Comparison Experiments

We make a comparison between IPLog and the six methods introduced in baselines. From the results in Table 3, we can see that our method outperforms almost all baseline methods in the evaluation metrics GA and PA. Specifically, in terms of group accuracy (GA), IPLog can achieve more than 0.95 accuracy on all datasets. In addition, IPLog achieves nearly 1.0 group accuracy on all three datasets except Hadoop, which outperforms existing log parsing methods. IPLog's performance is even more outstanding in terms of parsing accuracy (PA), which reaches more than 0.99 on both HDFS and Apache datasets,

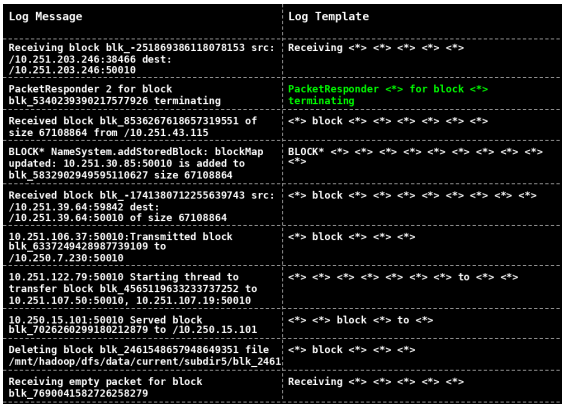
significantly outperforming the baseline method. For the Hadoop and Zookeeper datasets, IPLog still achieves the best performance, although it does not achieve accuracy above 0.9. IPLog's remarkable parsing accuracy underscores its ability to effectively discern the templates and corresponding parameters within log messages. The experimental results conclusively validate IPLog's capability to accurately identify log templates and parameters, thereby streamlining the categorization of logs into cohesive templates.

Table 3. Comparison with other log parsing methods in GA and PA.

	LFA		SLCT		Spell		Drain		LenMa		LogPPT		IPLog	
	GA	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**
HDFS	0.748	0.153	0.414	0.146	0.961	0.290	**0.999**	0.621	**0.999**	0.137	0.694	0.897	**0.999**	**0.998**
Hadoop	0.827	0.432	0.234	0.066	0.449	0.116	0.921	0.541	0.796	0.052	0.533	0.725	**0.953**	**0.816**
Zookeeper	0.839	0.346	0.749	0.684	0.987	0.789	0.994	0.843	0.857	0.683	0.973	0.843	**0.999**	**0.852**
Apache	0.805	0.637	0.420	0.175	**1.000**	0.250	**1.000**	0.727	0.993	0.031	0.786	0.952	0.998	**0.992**

Bold values indicate the optimal results.

In order to further study the parsing effect of IPLog on specific logs, we randomly selected 10 log messages in the HDFS dataset and studied and compared the parsing results of the seven methods for these 10 log messages. As shown in Figure 5, the green log templates represent the correctly parsed templates, e.g., for the LFA method, only the second log message "PacketResponder 2 for block blk_5340239390217577926 terminating" is correctly parsed as the log template "PacketResponder <*> for block <*> terminating", and the other nine log messages are parsed incorrectly. By comparison, we find that IPLog performs the best, successfully parsing all 10 log messages into correct log templates. For example, for complex logs like "Receiving block blk_-251869386118078153 src:/10.251.203.246:38466 dest:/10.251.203.246:50010", IPLog can accurately parse it into the log template "Receiving block <*> src: <*> dest: <*>". Except for LogPPT, which is also based on a neural network and can also parse it correctly, the other methods all parse it incorrectly. Overall, Drain, the best parser among the non-neural network methods, only parsed five of the ten log messages correctly, which also shows that it is difficult to adapt traditional parsing methods to complex system logs. In the neural network-based comparison, compared with LogPPT, IPLog parses the parameters more accurately. In the log message "Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010,10.251.107.19:50010", LogPPT incorrectly replaces the last two parameters "10.251.107.50:50010,10.251.107.19:50010" with a single <*>, which is obviously not conducive to log template differentiation.



Log Message	Log Template
Receiving block blk_-251869386118078153 src: /10.251.203.246:38466 dest: /10.251.203.246:50010	Receiving <*> <*> <*> <*> <*>
PacketResponder 2 for block blk_5340239390217577926 terminating	PacketResponder <*> for block <*> terminating
Received block blk_8536267618657319551 of size 67108864 from /10.251.43.115	<*> block <*> <*> <*> <*> <*> <*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.30.85:50010 is added to blk_5832902949595110627 size 67108864	BLOCK* <*> <*> <*> <*> <*> <*> <*> <*> <*> <*>
Received block blk_-1741380712255639743 src: /10.251.39.64:59842 dest: /10.251.39.64:50010 of size 67108864	<*> block <*> <*> <*> <*> <*> <*> <*> <*>
10.251.106.37:50010:Transmitted block blk_6337249428987739109 to /10.250.7.230:50010	<*> block <*> <*> <*>
10.251.122.79:50010 Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010, 10.251.107.19:50010	<*> <*> <*> <*> <*> <*> <*> to <*> <*>
10.250.15.101:50010 Served block blk_7026260299180212879 to /10.250.15.101	<*> <*> block <*> to <*>
Deleting block blk_2461548657948649351 file /mnt/hadoop/dfs/data/current/subdir5/blk_2461	<*> block <*> <*> <*>
Receiving empty packet for block blk_7690041582726258279	Receiving <*> <*> <*> <*> <*>

LFA

Log Message	Log Template
Receiving block blk_-251869386118078153 src: /10.251.203.246:38466 dest: /10.251.203.246:50010	Receiving block <*> src: /<*> dest: /<*>
PacketResponder 2 for block blk_5340239390217577926 terminating	PacketResponder 2 for block <*> terminating
Received block blk_8536267618657319551 of size 67108864 from /10.251.43.115	Received block <*> of size <*> from /<*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.30.85:50010 is added to blk_5832902949595110627 size 67108864	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
Received block blk_-1741380712255639743 src: /10.251.39.64:59842 dest: /10.251.39.64:50010 of size 67108864	Received block <*> src: /<*> dest: /<*> of size 67108864
10.251.106.37:50010:Transmitted block blk_6337249428987739109 to /10.250.7.230:50010	<*>:Transmitted block <*> to /<*>
10.251.122.79:50010 Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010, 10.251.107.19:50010	<*> Starting thread to transfer block <*> to <*>, <*>
10.250.15.101:50010 Served block blk_7026260299180212879 to /10.250.15.101	<*> Served block <*> to /<*>
Deleting block blk_2461548657948649351 file /mnt/hadoop/dfs/data/current/subdir5/blk_2461	Deleting block <*> file /mnt/hadoop/dfs/data/current/subdir5/<*>
Receiving empty packet for block blk_7690041582726258279	Receiving empty packet for block <*>

SLCT

Figure 5. *Cont.*

Spell

Log Message	Log Template
Receiving block blk_-251869386118078153 src: /10.251.203.246:38466 dest: /10.251.203.246:50010	Receiving block <*> src /<*> dest /<*>
PacketResponder 2 for block blk_5340239390217577926 terminating	PacketResponder <*> for block <*> terminating
Received block blk_8536267618657319551 of size 67108864 from /10.251.43.115	Received block <*> of size <*> from /<*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.30.85:50010 is added to blk_5832902949595110627 size 67108864	BLOCK* NameSystem.addStoredBlock blockMap updated <*> is added to <*> size <*>
Received block blk_-1741380712255639743 src: /10.251.39.64:59842 dest: /10.251.39.64:50010 of size 67108864	Received block <*> src /<*> dest /<*> of size <*>
10.251.106.37:50010:Transmitted block blk_63372494289877390 9 to /10.250.7.230:50010	<*> <*> block <*> to /<*>
10.251.122.79:50010 Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010, 10.251.107.19:50010	<*> Starting thread to transfer block <*> to <*> <*>
10.250.15.101:50010 Served block blk_7026260299180212879 to /10.250.15.101	<*> <*> block <*> to /<*>
Deleting block blk_2461548657948649351 file /mnt/hadoop/dfs/data/current/subdir5/blk_2461	Deleting block <*> file <*>
Receiving empty packet for block blk_7690041582726258279	Receiving empty packet for block <*>

Drain

Log Message	Log Template
Receiving block blk_-251869386118078153 src: /10.251.203.246:38466 dest: /10.251.203.246:50010	Receiving block <*> src: /<*> dest: /<*>
PacketResponder 2 for block blk_5340239390217577926 terminating	PacketResponder <*> for block <*> terminating
Received block blk_8536267618657319551 of size 67108864 from /10.251.43.115	Received block <*> of size <*> from /<*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.30.85:50010 is added to blk_5832902949595110627 size 67108864	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
Received block blk_-1741380712255639743 src: /10.251.39.64:59842 dest: /10.251.39.64:50010 of size 67108864	Received block <*> src: /<*> dest: /<*> of size <*>
10.251.106.37:50010:Transmitted block blk_63372494289877390 9 to /10.250.7.230:50010	<*>:Transmitted block <*> to /<*>
10.251.122.79:50010 Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010, 10.251.107.19:50010	<*> Starting thread to transfer block <*> to <*>, <*>
10.250.15.101:50010 Served block blk_7026260299180212879 to /10.250.15.101	<*> Served block <*> to /<*>
Deleting block blk_2461548657948649351 file /mnt/hadoop/dfs/data/current/subdir5/blk_2461	Deleting block <*> file <*>
Receiving empty packet for block blk_7690041582726258279	Receiving empty packet for block <*>

LenMa

Log Message	Log Template
Receiving block blk_-251869386118078153 src: /10.251.203.246:38466 dest: /10.251.203.246:50010	Receiving block <*> src: /<*> dest: /<*>
PacketResponder 2 for block blk_5340239390217577926 terminating	PacketResponder for block <*> terminating
Received block blk_8536267618657319551 of size 67108864 from /10.251.43.115	Received block <*> of size from /<*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.30.85:50010 is added to blk_5832902949595110627 size 67108864	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size
Received block blk_-1741380712255639743 src: /10.251.39.64:59842 dest: /10.251.39.64:50010 of size 67108864	Received block <*> src: /<*> dest: /<*> of size
10.251.106.37:50010:Transmitted block blk_63372494289877390 9 to /10.250.7.230:50010	<*>:Transmitted block <*> to /<*>
10.251.122.79:50010 Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010, 10.251.107.19:50010	<*> Starting thread to transfer block <*> to <*>, <*>
10.250.15.101:50010 Served block blk_7026260299180212879 to /10.250.15.101	<*> Served block <*> to /<*>
Deleting block blk_2461548657948649351 file /mnt/hadoop/dfs/data/current/subdir5/blk_2461	Deleting block <*> file
Receiving empty packet for block blk_7690041582726258279	Receiving empty packet for block <*>

LogPPT

Log Message	Log Template
Receiving block blk_-251869386118078153 src: /10.251.203.246:38466 dest: /10.251.203.246:50010	Receiving block <*> src: <*> dest: <*>
PacketResponder 2 for block blk_5340239390217577926 terminating	PacketResponder 2 for block <*> terminating
Received block blk_8536267618657319551 of size 67108864 from /10.251.43.115	Received block <*> of size <*> from <*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.30.85:50010 is added to blk_5832902949595110627 size 67108864	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
Received block blk_-1741380712255639743 src: /10.251.39.64:59842 dest: /10.251.39.64:50010 of size 67108864	Received block <*> src: <*> dest: <*> of size <*>
10.251.106.37:50010:Transmitted block blk_63372494289877390 9 to /10.250.7.230:50010	<*>:Transmitted block <*> to <*>
10.251.122.79:50010 Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010, 10.251.107.19:50010	<*> Starting thread to transfer block <*> to <*>
10.250.15.101:50010 Served block blk_7026260299180212879 to /10.250.15.101	<*> Served block <*> to <*>
Deleting block blk_2461548657948649351 file /mnt/hadoop/dfs/data/current/subdir5/blk_2461	Deleting block <*> file <*>
Receiving empty packet for block blk_7690041582726258279	Receiving empty packet for block <*>

IPLog

Log Message	Log Template
Receiving block blk_-251869386118078153 src: /10.251.203.246:38466 dest: /10.251.203.246:50010	Receiving block <*> src: <*> dest: <*>
PacketResponder 2 for block blk_5340239390217577926 terminating	PacketResponder <*> for block <*> terminating
Received block blk_8536267618657319551 of size 67108864 from /10.251.43.115	Received block <*> of size <*> from <*>
BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.30.85:50010 is added to blk_5832902949595110627 size 67108864	BLOCK* NameSystem.addStoredBlock: blockMap updated: <*> is added to <*> size <*>
Received block blk_-1741380712255639743 src: /10.251.39.64:59842 dest: /10.251.39.64:50010 of size 67108864	Received block <*> src: <*> dest: <*> of size <*>
10.251.106.37:50010:Transmitted block blk_63372494289877390 9 to /10.250.7.230:50010	<*>:Transmitted block <*> to <*>
10.251.122.79:50010 Starting thread to transfer block blk_4565119633233737252 to 10.251.107.50:50010, 10.251.107.19:50010	<*> Starting thread to transfer block <*> to <*>, <*>
10.250.15.101:50010 Served block blk_7026260299180212879 to /10.250.15.101	<*> Served block <*> to <*>
Deleting block blk_2461548657948649351 file /mnt/hadoop/dfs/data/current/subdir5/blk_2461	Deleting block <*> file <*>
Receiving empty packet for block blk_7690041582726258279	Receiving empty packet for block <*>

Figure 5. Comparison of parsing results of different parsing methods for ten randomly selected log messages in HDFS (green log templates represent correct parsing).

Comparison of the parsing results of 10 randomly selected log messages in the HDFS dataset with different parsing methods illustrates that IPLog can identify log templates and parameters more accurately, confirming the powerful ability that IPLog shows in log parsing tasks.

IPLog can maintain a high accuracy primarily because it learns semantic information from logs, thus achieving precise predictions of templates and parameters, a capability not found in rule-based methods. For example, in Figure 5, the log "Receiving block blk_-251869386118078153 src:/10.251.203.246:38466 dest:/10.251.203.246:50010" pertains to a data transfer process in HDFS. In this specific log, the "/" serves not merely as a character but also as a part of the network address, thereby ensuring the integrity and correctness of the parameters. However, apart from IPLog and LogPPT, which determine through learned semantic information that "/" is part of a network address, other methods merely split the

log by characters without understanding the role of "/" in representing addresses, leading to incorrect field extraction.

To further study the grouping effect of IPLog on specific log templates, we randomly selected seven log templates from the Hadoop dataset and studied and compared the grouping results of the seven methods on these templates. The results are shown in Table 4, where the numbers in the table indicate how many groups each method actually grouped the current template into after parsing was completed. The correct grouping situation is that each template should correspond to one group. Through this comparative analysis, we can evaluate the accuracy and effectiveness of each method in grouping templates. As can be seen from the table, both IPLog and Drain perform best by correctly grouping all the chosen templates into their respective original groups. The other methods, however, show large differences in their grouping results, especially the SLCT, Spell, and LenMa methods, which group into a large number of groups on multiple templates, suggesting that these methods have significant deficiencies in grouping log templates together.

Table 4. Comparison of different parsing methods for grouping seven randomly selected log templates in Hadoop (the number represents how many groups the current template is actually divided into after parsing is completed, of which only the number 1 represents that the current template is correctly divided into a group, and the other number n represents that the current log template is incorrectly divided into n).

	LFA	SLCT	Spell	Drain	LenMa	LogPPT	IPLog
Finished spill <*>	1	8	8	1	8	3	1
(EQUATOR) <*> kvi <*>(<*>)	1	2	197	1	197	2	1
soft limit at <*>	1	1	1	1	1	1	1
Assigning <*> with <*> to <*>	2	5	5	1	1	3	1
<*> Task Transitioned from <*> to <*>	1	4	4	1	609	15	1
Progress of TaskAttempt <*> is: <*>	1	37	887	1	1	2	1
mapreduce.cluster.local.dir for child: <*>	1	1	1	1	1	1	1

IPLog processes all system logs using a fixed set of hyperparameters, without the need for readjustments for each dataset or the setting of domain-specific regular expressions for different systems. However, similarity-based clustering, frequent pattern mining, and heuristic methods need to be adjusted according to different systems. Here, we demonstrate the regular expressions and the "support" parameter settings for the SLCT algorithm tailored to different systems. As shown in Table 5, a suitable regular expression is required to match IP addresses in all four types of system logs. For example, in HDFS logs, an additional match for block ID "blk_-?\d+" is necessary, while in Zookeeper logs, a leading "/" must be handled. These requirements vary by system. Additionally, the "support" parameter is passed to the SLCT method, specifying the threshold for template extraction support needed during the log parsing process. This parameter determines how many times a template must appear in the log data to be considered valid or representative. Adjusting the "support" parameter significantly affects the results of log parsing, including the number, type, and specifics of the templates. Table 5 shows the settings of the "support" parameter for each system under optimal conditions, reflecting their differences. These differences hinder the methods' generality.

Importantly, as systems are updated and log templates and formats change, rule-based log parsing methods like SLCT struggle to match new log formats. To further validate our idea, we simulated the process of log changes: relatively simple and few logs from loghub-2k to represent the log output before system upgrades, and a large number of complex logs from loghub2.0 to represent the log output after system updates. As Table 6 shows, IPLog experienced a minimal decrease in GA and PA metrics, only 0.30% and 0.76%, and was virtually unaffected. However, other methods, except for LFA, showed a more significant decrease in these metrics. For example, SLCT saw the largest drop in GA and PA metrics, at 25.08% and 22.61%, respectively. The neural network-based LogPPT

experienced decreases of 7.10% and 9.63% in GA and PA metrics. The experimental results confirmed that these methods struggle to adapt to changes in log formats, resulting in severe performance degradation.

Table 5. Comparison of regular expressions and parameters between IPLog and SLCT on different systems.

	SLCT		IPLog	
	Regular Expressions	**Support**	**Regular Expressions**	**Support**
HDFS	[r'blk_-?\d+', r'(\d+\.){3}\d+(:\d+)?']	120	None	None
Hadoop	[r'(\d+\.){3}\d+']	125		
Zookeeper	[r'(/ |)(\d+\.){3}\d+(:\d+)?']	10		
Apache	[r'(\d+\.){3}\d+']	5		

Table 6. Comparison of different parsing methods on GA and PA in loghub-2k and loghub2.0.

	LFA		SLCT		Spell		Drain		LenMa		LogPPT		IPLog	
	GA	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**	**GA**	**PA**
loghub-2k	0.906	0.363	0.606	0.345	0.935	0.436	0.981	0.724	0.931	0.279	0.803	0.945	0.990	0.921
loghub2.0	0.804	0.392	0.454	0.267	0.849	0.361	0.978	0.682	0.911	0.225	0.746	0.854	0.987	0.914
% Chg.	−11.26%	**+7.99%**	−25.08%	−22.61%	−9.20%	−17.20%	−0.31%	−5.80%	−2.15%	−19.35%	−7.10%	−9.63%	**−0.30%**	**−0.76%**

Bold values indicate non-significant decreases.

In the experimental results, we observed that only LFA showed an increase in PA performance. A detailed analysis of LFA's parsing results showed that it accurately parsed only a few log templates, such as "PacketResponder <*> for block <*> terminating". As the proportion of such templates increased in loghub2.0, LFA's PA metric correspondingly rose. From the comparison between IPLog and six other methods across loghub-2k and loghub2.0, it is inferred that in real production environments, when log templates change, IPLog is almost unaffected, whereas the other six methods experience a significant decline in performance, struggling to adapt to these changes.

5.2. Ablation Experiments

This section critically evaluates the performance of the fundamental components and parameters of our innovative approach. We first remove the manual feedback strategy, and as can be seen from the results in Table 7, after removing the manual feedback, the average GA of IPLog on the four datasets decreases by 17.2% (from 0.987 to 0.817), with little change in PA. We analyzed this result and designed the manual feedback merge query to correct mainly the errors and omissions of templates in the grouping process, so the improvement for GA is significant. For templates that are grouped incorrectly, it is possible for the incorrect template to become the original correct template during the merge process, but since the number of templates in this category is small, the impact on PA is not significant.

Table 7. Results of ablation studies for the manual feedback module and the prompt tuning module.

	GA	**PA**
Full IPLog	0.987	0.914
$w/o_{ManualFeedback}$	0.817	0.912
$w/o_{PromptTuning}$	0.971	0.890
$w/o_{PromptTuning\&ManualFeedback}$	0.659	0.885

In addition, in order to verify the importance of prompt tuning for IPLog, we first replaced the prompt tuning with fine-tuning, and the results in Table 7 show a relative decrease in both GA and PA. In addition, during the postprocessing merging process, we

found that the increase in parsing errors led to a significantly higher number of requests during merging, which undoubtedly increased the operational difficulty of postprocessing. In order to ensure the fairness of the comparison, we additionally dropped the manual feedback strategy at the same time on the basis of removing the prompt tuning, and the results in the table clearly show that IPLog based on fine-tuning and dropping the manual feedback decreased by 19.3% (from 0.817 to 0.659) in the metric GA and 2.9% (from 0.912 to 0.885) in the metric PA. The results of the ablation experiments in Table 7 confirm that the manual feedback strategy and prompt tuning can effectively improve the performance of IPLog.

Our adaptive system sampling algorithm can effectively select the best training sample data. In order to confirm the effectiveness of our sampling algorithm, we compare the effect of two other sampling algorithms with the results: random sampling and is DPP sampling. Note that for the fairness of the comparison, we uniformly cancelled the postprocessing operation in our experiments. Random sampling is a basic data sampling method that constructs subsets by randomly selecting samples from historical log datasets. The method does not take into account the intrinsic structure or characteristics of the data, and all samples have the same probability of being selected. DPP sampling is a sampling method based on the determinantal point process (DPP), aiming to select samples with diversity and representativeness. The DPP captures the similarity between samples by defining a kernel matrix, thus tending to select samples that are not similar to each other.

From the results in Figure 6, it is clear that the adaptive system sampling algorithm performs well in selecting samples, resulting in optimal model results. This result demonstrates the effectiveness of our sampling algorithm in maximally generalizing the features of the entire history log while using a small amount of data. Specifically, these sample data are not only optimized in terms of quantity but also balanced in terms of representativeness and diversity, thus ensuring the efficiency and accuracy of the model in subsequent training and analysis.

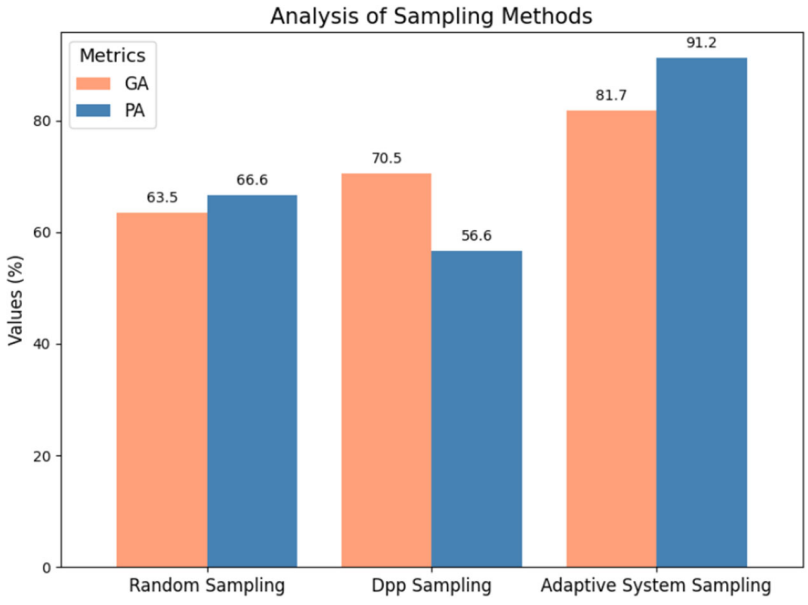


Figure 6. Comparison of different sampling algorithms.

6. Discussion

Based on our analysis, the reason that IPLog attains excellent performance in parsing logs for different systems is that it first utilizes a pretrained language model, which obtains basic sentence comprehension after training on a large amount of text data, and through the prompt tuning techniques, the model can more accurately identify the keywords and parameters in the log statements and complete the accurate extraction of the log templates.

Traditional methods (e.g., SLCT and Drain) usually rely on simple feature extraction and are unable to deeply understand the semantics of the text, so they are not as good as IPLog in terms of parsing accuracy. In addition, we believe that IPLog can easily perform log parsing on different system logs because it does not require domain-specific knowledge of regular expressions. IPLog extracts log templates by learning common patterns of keywords and parameters without the need to set up fixed rules for different systems in advance. When log formats change due to system updates, IPLog is largely immune to such changes, but methods based on domain-specific knowledge (e.g., LFA) are less effective. Methods that use deep learning models (e.g., LogPPT), although also showing a certain degree of superiority, tend to incorrectly group templates belonging to the same log group into multiple groups, which results in poor performance in grouping accuracy. However, IPLog greatly improves the method's grouping performance for templates with a small amount of human involvement through the manual feedback merge query strategy, which is missing for the other methods.

Of course, we also found some problems during our research. For example, we conducted our evaluation using public log datasets, which provide real log templates for all log messages. Despite being commonly utilized in related research, these datasets still include a minor proportion of errors. In addition, in our approach, we split the tokens in the original log messages by BPE in RoBERTa. However, because of the complexity of the log messages, some custom parameters are still difficult to split correctly, which leads to some degradation in the parsing accuracy. In future work, we will try more effective ways to deal with such cases.

In future research, we plan to explore the possibility of deploying IPLog in real production environments to deeply analyze its performance under real production conditions.

7. Conclusions

Log parsing is responsible for converting raw logs into a structured format, which is the basis for system troubleshooting and anomaly detection. In order to make log parsing more accurate and efficient, in this paper, we propose a log parsing method called IPLog. IPLog is based on few-shot learning, which firstly improves the quality and diversity of the training data by parsing historical log data, identifying log templates and adjusting the sampling strategy according to the template types through a novel adaptive system sampling method. Secondly, the template-free prompt tuning method is used to allow the model to fully extract the generic patterns of logs from these sampled labeled log data, which enhances the model's ability to identify log parameters and its parsing performance. In the online parsing phase, the trained model is directly applied to the parsing of new logs, thus enabling accurate extraction of log templates and parameters. At the end of our approach, we also propose a novel manual feedback merge query strategy based on the longest public prefix, which further improves the parsing ability of IPLog on complex logs.

We evaluated IPLog on four public log datasets. IPLog can achieve an average of 0.987 in group accuracy and 0.914 in parsing accuracy, which are the best in both metrics compared to existing methods. In order to deeply analyze the effectiveness of IPLog in the log parsing task, we additionally compared in detail the parsing results of ten randomly selected log messages in the HDFS dataset, as well as the grouping results of seven log templates randomly selected from the Hadoop dataset, and these comparisons also clearly show the superiority of IPLog compared to other methods.

Author Contributions: Conceptualization, L.Y.; methodology, L.Y. and S.L.; software development, L.Y.; validation, L.Y.; formal analysis, S.L. and L.Y.; investigation, L.Y.; resources, L.Y.; data curation, L.Y. and S.L.; original draft preparation, L.Y.; review and editing, L.Y., S.L., S.N., G.Z. and W.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (61762085) and the College Scientific Research Project Plan of Xinjiang Autonomous Region (XJEDU2020Y003): 61762085 and XJEDU2020Y003.

Data Availability Statement: The data are available upon request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Das, A.; Mueller, F.; Siegel, C.; Vishnu, A. Desh: Deep learning for system health prediction of lead times to failure in hpc. In Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, Tempe, AZ, USA, 11–15 June 2018; pp. 40–51.
2. Zhang, S.; Liu, Y.; Meng, W.; Luo, Z.; Bu, J.; Yang, S.; Liang, P.; Pei, D.; Xu, J.; Zhang, Y.; et al. Prefix: Switch failure prediction in datacenter networks. In Proceedings of the ACM on Measurement and Analysis of Computing Systems; Association for Computing Machinery: New York, NY, USA, 2018; Volume 2, pp. 1–29.
3. Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; et al. Robust log-based anomaly detection on unstable log data. In Proceedings of the ESEC/FSE 2019, Tallinn, Estonia, 26–30 August 2019; pp. 807–817.
4. Zhang, B.; Zhang, H.; Moscato, P.; Zhang, A. Anomaly detection via mining numerical workflow relations from logs. In Proceedings of the 2020 International Symposium on Reliable Distributed Systems (SRDS), Shanghai, China, 21–24 September 2020; pp. 195–204.
5. Zhang, B.; Zhang, H.; Le, V.-H.; Moscato, P.; Zhang, A. Semi-supervised and unsupervised anomaly detection by mining numerical workflow relations from system logs. *Autom. Softw. Eng.* **2023**, *30*, 4. [CrossRef]
6. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, Dallas, TX, USA, 30 October–3 November 2017; pp. 1285–1298.
7. He, P.; Zhu, J.; Zheng, Z.; Lyu, M.R. Drain: An online log parsing approach with fixed depth tree. In Proceedings of the 2017 IEEE International Conference on Web Services (ICWS), Honolulu, HI, USA, 25–30 June 2017; pp. 33–40.
8. Dai, H.; Li, H.; Chen, C.S.; Shang, W.; Chen, T.-H. Logram: Efficient log parsing using n-gram dictionaries. *IEEE Trans. Softw. Eng.* **2020**, *48*, 879–892. [CrossRef]
9. He, P.; Zhu, J.; He, S.; Li, J.; Lyu, M.R. An evaluation study on log parsing and its use in log mining. In Proceedings of the 2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN), Toulouse, France, 28 June–1 July 2016; pp. 654–661.
10. Le, V.-H.; Zhang, H. Log-based anomaly detection without log parsing. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 492–504.
11. Ma, R.; Zhou, X.; Gui, T.; Tan, Y.; Li, L.; Zhang, Q.; Huang, X. 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, Seattle, WA, USA, July 2022; Association for Computational Linguistics: USA; pp. 5721–5732. Available online: https://aclanthology.org/2022.naacl-main.420 (accessed on 12 June 2023).
12. Zhu, J.; He, S.; Liu, J.; He, P.; Xie, Q.; Zheng, Z.; Lyu, M.R. Tools and benchmarks for automated log parsing. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, QC, Canada, 25–31 May 2019; pp. 121–130.
13. Vaarandi, R. 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), Kansas City, MO, USA, 3 October 2003; pp. 119–126.
14. Nagappan, M.; Vouk, M.A. Abstracting log lines to log event types for mining software system logs. In Proceedings of the 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), Cape Town, South Africa, 2–3 May 2010; pp. 114–117.
15. Vaarandi, R.; Pihelgas, M. Logcluster-a data clustering and pattern mining algorithm for event logs. In Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 9–13 November 2015; pp. 1–7.
16. Fu, Q.; Lou, J.G.; Wang, Y.; Li, J. Execution anomaly detection in distributed systems through unstructured log analysis. In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, 6–9 December 2009; pp. 149–158.
17. Shima, K. Length matters: Clustering system log messages using length of words. *arXiv* **2016**, arXiv:1611.03213.
18. Jiang, Z.M.; Hassan, A.E.; Flora, P.; Hamann, G. Abstracting execution logs to execution events for enterprise applications (short paper). In Proceedings of the 2008 The Eighth International Conference on Quality Software, Oxford, UK, 12–13 August 2008; pp. 181–186.
19. Du, M.; Li, F. Spell: Streaming parsing of system event logs. In Proceedings of the 2016 IEEE 16th International Conference on Data Mining (ICDM), Barcelona, Spain, 12–15 December 2016; pp. 859–864.
20. Nedelkoski, S.; Bogatinovski, J.; Acker, A.; Cardoso, J.; Kao, O. Self-supervised log parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track*; Springer: Cham, Switzerland, 2020; pp. 122–138.
21. Yu, S.; Chen, N.; Wu, Y.; Dou, W. Self-supervised log parsing using semantic contribution difference. *J. Syst. Softw.* **2023**, *200*, 1116462023. [CrossRef]
22. Le, V.H.; Zhang, H. Log parsing with prompt-based few-shot learning. In Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 14–20 May 2023; pp. 2438–2449.

23. Li, X.; Chen, P.; Jing, L.; He, Z.; Yu, G. Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 12–15 October 2020; pp. 92–103.
24. Liu, Y.; Ott, M.; Goyal, N.; Du, J.; Joshi, M.; Chen, D.; Levy, O.; Lewis, M.; Zettlemoyer, L.; Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv* **2019**, arXiv:1907.11692.
25. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. 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)*; Association for Computational Linguistics: Minneapolis, MN, USA, 2019; pp. 4171–4186.
26. Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog* **2019**, *1*, 9.
27. Guo, H.; Yuan, S.; Wu, X. Logbert: Log anomaly detection via bert. In Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN), Shenzhen, China, 18–22 July 2021; pp. 1–8.
28. Xu, J.; Fu, Q.; Zhu, Z.; Cheng, Y.; Li, Z.; Ma, Y.; He, P. Hue: A user-adaptive parser for hybrid logs. In Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, San Francisco, CA, USA, 3–9 December 2023; pp. 413–424.
29. [n. d.]. Public Datasets for Log Parsing. Available online: https://github.com/logpai/loghub-2.0 (accessed on 23 March 2023).
30. Zhu, J.; He, S.; He, P.; Liu, J.; Lyu, M.R. Loghub: A Large Collection of System Log Datasets for AI-driven Log Analytics. In Proceedings of the 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), Florence, Italy, 9–12 October 2023.
31. Loshchilov, I.; Hutter, F. Decoupled weight decay regularization. In Proceedings of the 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, 6–9 May 2019; OpenReview.net. 2019. Available online: https://openreview.net/forum?id=Bkg6RiCqY7 (accessed on 5 January 2024).
32. [n. d.]. Detailed Performance Metrics Table of Log Parsers. Available online: https://github.com/logpai/loghub-2.0/blob/main/RQs_experiments/RQ2/effectiveness_results.csv (accessed on 12 March 2024).
33. Jiang, Z.; Liu, J.; Huang, J.; Li, Y.; Huo, Y.; Gu, J.; Chen, Z.; Zhu, J.; Lyu, M.R. A Large-scale Evaluation for Log Parsing Techniques: How Far are We? In Proceedings of the ISSTA, Vienna, Austria, 16–20 September 2024.
34. Khan, Z.A.; Shin, D.; Bianculli, D.; Briand, L. Guidelines for Assessing the Accuracy of Log Message Template Identification Techniques. In Proceedings of the ICSE '22: 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 21–29 May 2022.