Hindawi
International Journal of Intelligent Systems
Volume 2024, Article ID 5961993, 17 pages
https://doi.org/10.1155/2024/5961993

Research Article

DLLog: An Online Log Parsing Approach for Large-Scale System

Hailong Cheng,[1] Shi Ying,[1] Xiaoyu Duan,[2] and Wanli Yuan[1]

[1]*School of Computer Science, Wuhan University, Wuhan 430072, China*
[2]*College of Software Engineering, Zhengzhou University of Light Industry, Zhengzhou 450002, China*

Correspondence should be addressed to Shi Ying; 2018282110412@whu.edu.cn

Received 7 March 2023; Revised 21 March 2024; Accepted 26 March 2024; Published 16 April 2024

Academic Editor: Vasudevan Rajamohan

Syslog is a critical data source for analyzing system problems. Converting unstructured log entries into structured log data is necessary for effective log analysis. However, existing log parsing methods demonstrate promising accuracy on limited datasets, but their generalizability and precision are uncertain when applied to diverse log data. Enhancements in these areas are necessary. This paper proposes an online log parsing method called DLLog, which is based on deep learning and has the longest common subsequence. DLLog utilizes the GRU neural network to mine template words and applies the longest common subsequence to parse log entries in real-time. In the offline stage, DLLog combines multiple log features to accurately extract the template words, creating a log template set to assist online log parsing. In the online stage, DLLog parses log entries by calculating the matching degree between the real-time log entry and the log template in the log template set. This method also supports the incremental update of the log template set to handle new log entries generated by systems. We summarized the previous works and validated DLLog using real log data collected from 16 systems. The results demonstrate that DLLog achieves high parsing accuracy, universality, and adaptability.

1. Introduction

Log data serves as a valuable and reliable source for operations staff to monitor systems, detect abnormalities, and locate faults [1]. Log data, easily obtainable from systems, contains a wealth of information, including system status, performance, and resource usage. However, log data is inherently unstructured, while most system analysis tasks require structured data as input [2–4]. Therefore, parsing unstructured log data into structured data becomes essential [5, 6]. This paper aims to develop a log parsing method characterized by high accuracy, universality, and adaptability. The goal is to enable the accurate extraction of log templates from log data without manual intervention.

Traditional log parsing methods require considerable human resources and time. Moreover, as system scale and complexity increase, data volume expands rapidly. Importantly, developers have not established a unified standard for log format, making traditional manual log parsing methods impractical. Static code-based parsing methods exhibit high limitations [7–9] because obtaining system source code during the parsing process is challenging. While frequent pattern mining-based log parsing methods demonstrate competitive parsing efficiency, they struggle to match rare logs with low frequency to any log template, resulting in suboptimal parsing results [10–12]. Clustering-based log parsing methods often suffer from low parsing accuracy due to their simplistic parsing patterns (e.g., dividing log groups based on word frequency or different word types) [2, 13–15]. In comparison to static code or frequent pattern mining-based methods, clustering methods have slower parsing speeds and require numerous iterations.

Current log parsing methods often exhibit limitations in terms of parsing accuracy and universality. While a specific log parsing method may demonstrate high detection accuracy for a particular dataset, it frequently struggles to maintain comparable accuracy when applied to a broader range of datasets. It is imperative for log parsing methods to incorporate incremental update capabilities, as systems undergo sporadic updates or optimizations postdeployment, resulting in the generation of new log data that needs to be matched with novel log templates. Log parsing methods lacking incremental update functionality require substantial computational resources to build a new parsing model. Undoubtedly, a log parsing method with the ability to update its model during the parsing process is of paramount importance.

To address these challenges, we propose an online log parsing method called DLLog, based on GRU neural networks and the longest common subsequence. Our method outperforms existing approaches by accurately mining log template words using multiple log features, thus achieving high universality. Prior to template matching, DLLog preclassifies log templates to reduce incorrect template matching time. Moreover, our method supports log template set updates to accommodate new log data generated by the system.

DLLog parses logs by utilizing the structural, frequency, and association features of logs entries. It combines offline log template word mining and online log parsing to enhance the universality and parsing accuracy of DLLog for large-scale log datasets. In the offline mining stage, DLLog initially employs common regular expressions to clean and remove obvious parameter words from the logs. Then, it transforms the log entry into a sequence of word frequencies based on log word frequency and log structural feature. Subsequently, DLLog employs a GRU neural network to identify potential relationships between log words and extract log template words based on these relationships. Due to log sequences including the log structural feature, log template words from rare logs are more easily and accurately mined by DLLog. Finally, log entries with the same log template words are categorized into the same log group. Each log group corresponds to a log template. Different log templates form a pre-classified log template set based on the log structural feature. This process does not require manual intervention. Besides, this grouping pattern can effectively avoid the problem that rare logs cannot match any log template.

During the online parsing stage, DLLog processes logs by calculating the matching degree, defined as the length of the longest common subsequence between real-time log entries and the existing log template set. Based on the matching results, DLLog determines whether to update the log template set. By adopting incremental updates to the log template set, DLLog eliminates the need for retraining models, ensuring the efficient operation of log parsing methods and enhancing the method's universality when applied to large-scale log datasets. This paper evaluates DLLog on several extensive log datasets, demonstrating its success in achieving high parsing accuracy, universality, and adaptability.

The primary contributions of this paper are summarized as follows:

(i) This paper introduces an offline log template word mining approach that utilizes a GRU neural network to extract log template words and partition log data into distinct log groups.

(ii) This paper proposes an online log parsing method that leverages the longest common subsequence, enabling updates to the log template set to accommodate newly generated log data from the system.

(iii) We conducted comprehensive experiments and evaluations on various large-scale log datasets, demonstrating the superior performance of DLLog in terms of accuracy, universality, and adaptability.

The rest of this paper is organized as follows: Section 2 presents the related work of log parsing. Section 3 presents the basic structure of log data. Section 4 presents the detailed design of DLLog. Section 5 evaluates the performance of DLLog through experiments. Finally, Section 6 presents the final remarks.

2. Related Work

System logs are invaluable data resources extensively utilized in system operation and maintenance, fault analysis and detection, and various practical applications [16–19]. Since log messages typically consist of semi-structured text strings, log parsing is essential for converting unstructured logs into structured data [20]. Log parsing preserves the essence of log entries, removes parameter words, and minimizes log entry dimensions, making it easier to map diverse unstructured logs to standard log templates. We have categorized and summarized recent research in log parsing into the following categories.

2.1. Static Code-Based Log Parsing Methods. Liang et al. [8] introduced an MTS-DCGAN log parsing method based on source code analysis. This approach involves querying class names, call relationships, and object names associated with system behaviors. By traversing the syntax tree, log templates are constructed. Kabinna et al. [21] proposed the Cox models, which follow similar principles as the MTS-DCGAN, identifying format strings in the code to create log parsing templates. While these methods accurately generate log templates, they are dependent on access to system source code, limiting their applicability to closed-source systems.

2.2. Heuristic-Based Log Parsing Methods. He et al. [22] developed Drain, a log parsing method that utilizes parsing trees. Drain constructs parsing trees and then compares variances between log entries and log event groups within the parsing tree for log parsing purposes. While Drain provides high parsing accuracy, its versatility is limited and, and it requires domain-specific knowledge.

Zhang et al. [23] presented the FT-tree method for log parsing, which creates a log template tree by analyzing log words and their combinations. The process involves pruning the log template tree by removing branches that do not satisfy certain constraints. Consequently, all log words along the path from the root node to any leaf node in the pruned log template tree constitute a log template. However, a drawback of this method is its tendency to overlook infrequent log templates, potentially leading to reduced parsing accuracy.

2.3. Clustering-Based Log Parsing Methods. Sedki et al. [10] proposed the unified log parsing tool, which identifies frequent phrases in log data to form frequent candidate itemsets. These itemsets are then clustered to generate class clusters along with their corresponding templates. Fu et al. [13] proposed the LKE method, a log parsing technique based on K-means clustering. LKE extracts log templates from initial log groups obtained by segmenting clusters using cluster midpoints and parameter distances. However, due to log data imbalance, clustering-based methods might misclassify low-frequency log template words as parameter words, leading to lower parsing accuracy.

2.4. Other Log Parsing Methods. Makanju et al. [24] proposed the iterative partition log mining (IPLoM) method, which employs iterative partitioning to categorize log entries into distinct groups. IPLoM further refines partitions based on log identifiers and location information to extract log templates for each log group. AEL [25] employs a clone detection method for log parsing, assuming significant text similarity among log entries within the same log event. AEL employs the "Adjust" step to consolidate similar log execution events and resolve all log templates. Du and Li [26] presented Spell, an online log parsing method based on the longest common subsequence, which updates and maintains the longest common subsequence library (LCSMap) of log event sequences.

3. Log Structure Overview

System logs are unstructured data stored as free text, recording various events, states, errors, or interaction behaviors generated by systems or components. Typically, there is no unified standard for defining log entry formats and syntax structures across different systems. Each log entry consists of a constant part and a variable part. The constant part, also referred to as the log template, comprises fixed plain text information generated by the printout code, containing semantic information in the form of log template words. The variable part, including dynamic parameter information such as IP addresses, port numbers, and file names, changes with log events. The words that make up the variable parts are referred to as log parameter words and generally lack valuable semantic information. Although the formats of log data vary greatly among different systems, these log data typically include the following important components: timestamp, log level, components, and log events.

(1) Timestamp: The time when the system generated the log entry.

(2) Log level: Also known as log type, it indicates the severity of log events (such as info, error, and warn).

(3) Component: The name of the component (software module or server) that generates log events.

(4) Log event: Describe the system interaction event information under specific time and environment. Generally, a log entry contains only one log event.

In log data, the log event serves as the core of each log entry. Log parsing extracts the constant part (common field) of log events to create a log template representing each log entry. Table 1 displays log samples from eight different types of original log data, including distributed systems, supercomputer systems, operating systems, and mobile systems.

We take the HDFS log entry (081109 203521 146 INFO dfs. DataNode$Packet Responder: Received block blk_7503483334202473044 of size 233217 from/10.251.71.16) as an example. The log event part is generated by the system printout code "LOG.info ("Received block" + block + "size" + block. getNumBytes() + "from" + inAddr)." The fixed parts are "Received block," "of size," and "from," which remain unchanged regardless of the event object. Simultaneously, these words also constitute the log template for the log event. Table 2 displays the classification results of the aforementioned original log entry based on Timestamp, Log level, Component name, and Log events. This table also presents the Log template words, Parameter words, and Log template.

In HDFS log data format Table 2, the symbol "$*$"denotes a placeholder. In fact, a log template can be used to represent multiple log entries. Figure 1 provides twelve examples of HDFS raw log data.

In these examples, the log template "Received block* of size* from*"can also represent the second log entry (081109 205412 832 INFO dfs. DataNode$PacketResponder: Received block blk_-5704899712662113150 of size 67108864 from / 10.251.91.229). Each log entry in log data can be characterized by only one log template, but one log template can represent multiple log entries. Table 3 displays the corresponding log templates for all log data examples in Figure 1.

As shown in Table 3, we can convert 12 different types of unstructured log entries into 5 types of structured data by transforming log data into log templates. Indeed, a log template is a standardized format for representing a group of original log entries. Log entries with the same log template represent the same type of log events. In essence, the core of log parsing lies in converting each log entry into a specific log template. During log parsing, a parser must explicitly distinguish between the constant and variable parts of the log event, extract the constant log part (log template words) to compose the log template, and then use the log template to represent the log entry, thereby completing the log data parsing task.

4. DLLog Architecture and Overview

This section will provide a detailed overview of the proposed online log parsing method, DLLog, which is based on GRU deep learning and has the longest common subsequence. The fundamental concept behind DLLog is that log templates typically consist of the longest combinations of frequently occurring words. DLLog comprises three main modules: log data vectorization, offline log template word mining, and online log parsing. Figure 2 illustrates the framework of the DLLog. Table 4 illustrates notations with their explanatory terms of The DLLog.

Table 1: Sample of raw log data.

Category	Name	Log data
Distributed system	HDFS	081109 203521 146 INFO dfs. DataNode$Packet responder: received block blk_7503483334202473044 of size 233217 from/10.251.71.16
	Hadoop	2015-10-18 18:01:53,713 INFO [main] org. Apache. Hadoop. mapreduce.v2.app.rm.RMContainerAllocator: maxContainerCapability: <memory: 8192, vCores: 32>
	Spark	17/06/09 20:10:45 INFO executor.Executor: running task 2.0 in stage 0.0 (TID2)
	OpenStack	Nova-compute.log.1.2017-05-16_13:55:31 2017-05-16 00:00:04.500 2931 INFO nova.compute.manager [req-3ea4052c-895d-4b64-9e2d-04d64c4d94ab - - - - -] [instance: b9000564-fela-409b-b8cc-1e88b294cd1d] VM started (lifecycle event)
Supercomputer system	BGL	- 1134748483 2005.12.16 R60-M1-N7-C:J17-U01 2005-12-16-07.54.43.245870R60-M1-N7-C:J17-U01 RAS KERNEL INFO total of 20 ddr error(s) detected and corrected over 22070 seconds
Operating system	Windows	2016-09-28 04:30:31, info CBS SQM: queued 0 file(s) for upload with pattern: C: \ windows\ servicing\ sqm* _all.sqm, flags: 0 × 6
	Linux	Jul 27 14:41:58 combo kernel: PCI: PCI BIOS revision 2.10 entry at 0xfc0ce, last bus = 1
Mobile system	Android	03-17 16:16:07.635 1702 1820 D DisplayPowerController: animating brightness: target = 38, rate = 200

Table 2: HDFS log data format.

Timestamp	081109 203521 146
Log level	INFO
Component name	dfs. DataNode$PacketResponder
Log event	Received block blk_7503483334202473044 of size 233217 from/10.251.71.16
Log template words	"Received," "block," "of," "size," "from"
Parameter words	"blk_7503483334202473044," "233217," "/10.251.71.16"
Log template	Received block * of size * from *

```
081109 203521 146 INFO dfs.DataNode$PacketResponder: Received block blk_7503483334202473044 of size 233217 from
/10.251.71.16
081109 205412 832 INFO dfs.DataNode$PacketResponder: Received block blk_-5704899712662113150 of size 67108864 from
/10.251.91.229
081110 092459 8650 INFO dfs.DataNode$DataXceiver: 10.250.9.207:50010 Served block blk_4355450627202483068 to/10.250.9.207
081110 092815 8872 INFO dfs.DataNode$DataXceiver: 10.250.6.191:50010 Served block blk_5952254363678329024 to/10.250.6.191
081110 093020 8827 WARN dfs.DataNode$DataXceiver: 10.251.70.211:50010:Got exception while serving blk_-66793317148508522
to/10.251.203.246:
081110 093643 13 INFO dfs.DataBlockScanner: Verification succeeded for blk_9188832735514090334
081110 093831 8571 INFO dfs.DataNode$DataXceiver: 10.251.106.10:50010 Served block blk_2273334621242106674 to/10.251.106.10
081110 094019 8808 WARN dfs.DataNode$DataXceiver: 10.251.125.193:50010:Got exception while serving blk_3790492230047189408
to/10.251.199.159:
081110 094657 7835 INFO dfs.DataNode$DataXceiver: 10.251.107.50:50010 Served block blk_-2285729896739318683 to/10.251.70.5
081110 103026 34 INFO dfs.FSNamesystem: BLOCK* NameSystem.delete: bk_-1233005817943453613 is added to invalidSet of
10.251.75.49:50010
081110 103027 34 INFO dfs.FSNamesystem: BLOCK * NameSystem.delete: blk_166171721314010075 is added to invalidSet of
10.251.30.85:50010
081110 103027 34 INFO dfs.FSNamesystem: BLOCK* NameSystem.delete: blk_3362838757940877177 is added to invalidSet of
10.250.5.161:50010
```

Figure 1: HDFS raw log data.

Table 3: HDFS log data template.

Template no.	Data size
1	Received block * of size * from *
2	**Got exception while serving * to *
3	**Served block * to *
4	BLOCK * NameSystem.delete: *is added to invalidSet of **
5	Verification succeeded for *

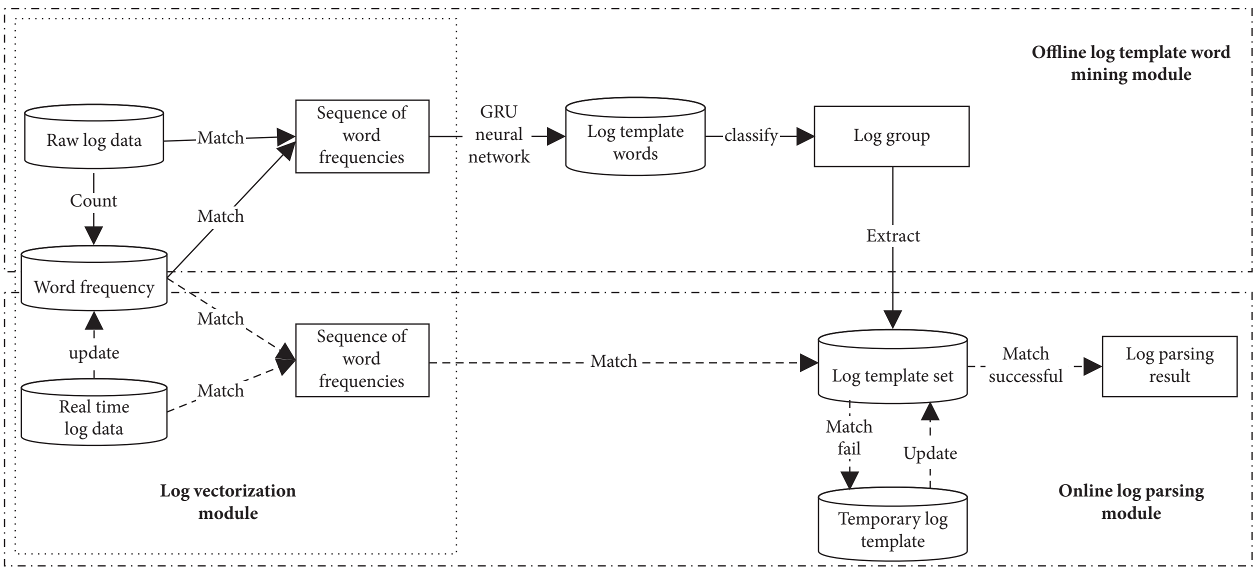


Figure 2: The architecture of DLLog.

4.1. Offline Log Template Word Mining. The solid arrow in Figure 2 illustrates DLLog's offline log template word mining process. This module initially scans and cleans the entire log dataset. It counts the frequency of each word that makes up the log level, component name, and log event. Using this frequency information, we construct a log word frequency table. DLLog vectorizes each log entry based on the word frequency ID in the word frequency table, converting the log entry into a vector to create the log word frequency sequence. During the training stage, the GRU neural networks are employed to learn the relationships between log words, enabling DLLog to extract log template words from log sequences based on the learned associations. Finally, DLLog categorizes log entries into different log groups depending on whether the log template words are identical. Each log group's log entries share the same log template, and the log templates from different log groups constitute the set of log templates.

4.2. Online Log Parsing. The dashed arrow in Figure 2 illustrates the online parsing process of the DLLog. Unlike offline log template word mining, the log sequence in online log parsing does not require sorting based on word frequency. Each real-time log entry only needs to undergo a cleaning process before being input into the log vectorization module to generate a sequence of log word frequencies. Then, DLLog calculates the matching degree between the current log sequence and the log template within the existing log template set. By comparing the matching degree with the predefined threshold, DLLog determines whether the parsing is successful or if the log template set needs updating.

4.3. Log Vectorization. We define the log dataset as $L = <\log_1, \log_2, \ldots, \log_n>$, where $\log_i$ represents a log entry, and we define the log event set $E = <\text{event}_1, \text{event}_2, \ldots, \text{event}_n>$, where event_i represents a log event. Let $W = <\text{word}_1, \text{word}_2, \ldots, \text{word}_n>$ be the set of words constituting the log event. These words are also referred to as tokens. If the log word_i appears frequently (that is, it has a high-frequency), then word_i has a high probability of being a log template word; We define the set of log templates as $M = <\text{tem}_1, \text{tem}_2, \ldots, \text{tem}_n>$, where tem_i represents a log template composed of multiple log template words word_i, arranged in a specific manner. It should be noted that each log template tem_i in the log template set M corresponds to multiple log entries, and a log entry can only be represented by one log template.

Log vectorization is the first step in the DLLog. Its objective is to convert unstructured log entries into vectorized sequences, which are then used for offline log template word mining and online log parsing. The process for vectorizing log entries consists of three steps:

(1) The first step is to scan the entire log dataset, break down the log into words, and employ regular expressions to filter obvious log parameter words (such as IP address and file path) with a fixed format. This log vectorization process in Section 4.1 processes log datasets using the log data filtering rules provided by the FT-tree [23], spell [26] and drain [22], which is widely adopted in the Log parsing domain.

(2) The second step is to count the frequency of log words. In this step, the module fully considers the structural and frequency features of the log. The frequency is derived from the statistics of log level word (F_{level}), log component word ($F_{\text{component}}$) and log event word (F_{word}). Next, we categorize the frequency information by the word type, sort it in descending order, and store in the word frequency table, denoted as F, which is defined as $F = F_{\text{level}} \cup F_{\text{component}} \cup F_{\text{word}}$. Each word is assigned a unique word frequency ID. Within the word

Table 4: Notations with their explanatory terms.

Notation	Definition
L	Log dataset
E	Log event set
W	Words set
M	Log template set
F_{level}	Log level word
F_{word}	Log event word
ID	Word frequency ID
LSTM	Long short term memory
r_t	Reset gate
h_t	Hidden state
σ	Activation functions
$\oplus$	Addition
W_u	Weight value
h	Size of the sliding window
y_i	Actual label
k	Number of categories
P	Probability distribution of output
ρ	Probability threshold
$\text{tem}_{\text{total}}$	Log template set
temlength	Optimal template length
new_logs	Temporary log set
$\mathscr{L}_i$	Matching degree
newtem	New log template
log_i	Log entry
event_i	Log event
word_i	Log word
tem_i	Log template
$F_{\text{component}}$	Log component word
F	Word frequency table
GRU	Gated recurrent unit
x_t	Input
u_t	Renewal gate
$\tilde{h}_t$	Candidate hidden state
tanh	Activation functions
$\otimes$	Point multiplication
W_R	Weight value
S_h	Log token subsequence
p_i	Probability value
N	Number of total samples
p_i	Log word probability
tem_i	Log template i
best	Optimal matching degree
w	Number
Tem_{same}	Special log template set
$\text{LCS}(S, \text{tem}_i)$	Longest common subsequence

frequency table F, the ID s corresponding to the F_{level} are positioned at the beginning of the frequency table, followed by $F_{\text{component}}$ in the middle, and F_{word} at the end. Figure 3 provides a structure sample of the word frequency table F.

We use the HDFS original log dataset in Figure 1 as an example to further illustrate the process of creating the word frequency table F. This log dataset includes two log levels: "INFO" and "WARN." Therefore, the word frequency sorting result for log levels can be expressed as $F_{\text{level}} = <(1\text{: INFO: }10), (2\text{: WARN: }2)>$. Similarly, the frequency sorting results for the four log components can be expressed as $F_{\text{component}} = <$ (3: dfs.DataNode *DataXceiver*. The processing of F_{word} follows the same procedure as $F_{\text{component}}$.

The final word frequency table F corresponding to the log dataset can be expressed as $F = <(1\text{: INFO: }10), \ldots, (25\text{: } blk - 3362838757940877177\text{: }1)>$. In the table F, each row is represented as a triple (ID: Word: Frequency), where the first unit represents the word frequency ID, the second unit is the word itself, and the third unit is the frequency (the number of times the words appear in the dataset). By categorizing log words into F_{level}, $F_{\text{component}}$ and F_{word}, this method helps prevent the incorrect categorization of low-frequency log template words as log parameter words, mitigating issues arising from unbalanced log data features.

(3) The third step is to replace log words with word frequency IDs, constructing the log word (token) frequency sequence in ascending order.

It is important to note that the online log entry vectorization process only requires the first and third steps. Since the word frequency table F has already been constructed, we simply need to follow step (1) to clean the online log entry. If a log word appears in the real-time log but is not present in the word frequency table F, we incrementally update the word frequency table F with the newly encountered log word. Then, according to the word frequency table F, we construct the cleaned log data into a log word sequence. Figure 4 illustrates the example of log vectorization.

4.4. Offline Log Template Word Mining.

Offline log template word mining aims to create an accurate log template set. During the log vectorization module, DLLog converts each log entry into a sequence of log word frequencies based on the log structural features and log frequency features. In the offline log template word mining module, DLLog learns the relationship between log words through GRU neural network. It determines whether words are log parameter words or log template words, enabling the accurate extraction of log templates.

The core method of offline log template word mining is GRU neural network [27]. GRU neural network is a well-known variant of recurrent neural network (RNN) and was introduced by Cho et al. [27]. It has found wide application in various fields, including text classification [28, 29], machine translation [30], emotion analysis [31].

Compared with LSTM neural network, the GRU neural network has a forgetting and updating mechanism, both of which excel at tracking long-term dependencies. These mechanisms address the challenge of gradient vanishing or exploding that often occurs in recurrent neural networks during multiple propagations. Unlike LSTM neural network, the GRU neural network simplifies the internal network structure, resulting in more efficient state information updates. The internal structure of GRU unit is depicted in Figure 5.

	ID	Log word	Word frequency
	1	$level_{word_1}$	200
F_{level}	2	$level_{word_2}$	100
	3	$level_{word_3}$	50
	4	$component_{word_1}$	150
$F_{component}$	…	…	…
	20	$component_{word_{20}}$	7
	21	$word_{21}$	200
F_{word}	…	…	…
	n	$word_n$	1

Figure 3: Structured log word frequency table.

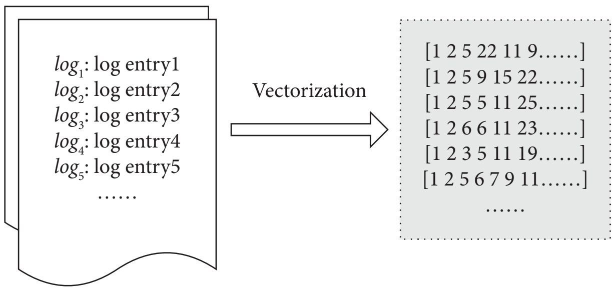


Figure 4: Example of log vectorization.

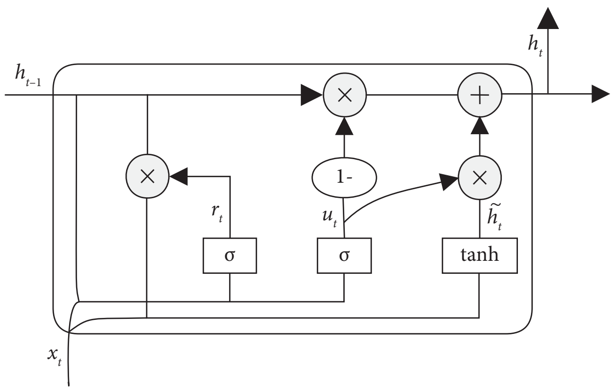


Figure 5: The internal structure of GRU unit.

In Figure 5, x_t represents input at time t (current time), $r_t \in [0, 1]$ is the reset gate, $u_t \in [0, 1]$ is the update gate, h_t is the hidden state of the current GRU unit, h_{t-1} is the hidden state from the previous GRU unit, $\tilde{h}_t$ is the candidate hidden state. σ And tanh are activation functions, ⊕ represents addition and ⊗ represents point multiplication.

Each GRU block in a GRU neural network consists of an update gate and a reset gate. The reset gate determines which part of the information in the hidden state is "forgotten," while the update gate decides how much of the current input information is incorporated and temporarily stored in the hidden state $\tilde{h}_t$. The formulas for reset gate, update gate, and hidden state are as follows:

$$u_t = \sigma\left(W_u \cdot \left[h_{t-1}, x_t\right]\right) \tag{1}$$

$$r_t = \sigma\left(W_r \cdot \left[h_{t-1}, x_t\right]\right) \tag{2}$$

$$\tilde{h}_t = \tanh\left(W \cdot \left[r_t * h_{t-1}\right], x_t\right) \tag{3}$$

$$h_t = \left(1 - u_t\right) * h_{t-1} + u_t * \tilde{h}_t \tag{4}$$

where, W_u, W_R, and W represent the weight value. When $u_t = 1$, it means retaining the state from the past time to the current state. When $u_t = 0$, it signifies forgetting the past status information.

4.4.1. Training Stage of DLLog. The DLLog log template word mining model employs a two-layer GRU neural network. Compared with a single-layer GRU neural network, the two-layer GRU neural network exhibits superior learning and generalization capabilities, making it better at preserving long-term dependencies within sequences. The log template word mining model based on the GRU neural network consists of four layers: the word embedding layer, the GRU neural network layer, the fully connected layer, and softmax layer. Figure 6 illustrates the network structure of the DLLog log template word mining model.

We input a log token subsequence of length h (where h represents the size of the sliding window) $S_h = < s_{t-h}, \ldots, s_{t-2}, s_{t-1} >$ into the model, where $\forall s_i \in F$. First, the log token subsequence s_t is passed through the word vectorization layer, which maps each token to a computationally recognizable vector. These word vectors then serve as input to the first layer of the GRU neural network. Both the first and second layers of the GRU neural network comprise h GRU units, matching the length of the input data.

In each GRU cell, the input consists of the hidden state h_{t-1} from the previous time step and the external input data at the current time step. The currently embedded word vector and hidden state h_{t-1} are both weighted in the update gate using their respective weights. The result of this weighted sum, obtained using formula (1), is then passed through a sigmoid activation function to calculate the final value of the update gate. The input for the reset gate is identical to that of the update gate, with both being multiplied by their corresponding weights. Formula (2) is applied to calculate the value of the reset gate. The reset gate determines how much information from the previous hidden state will be updated to the current candidate hidden state $\widetilde{h}_t$, while the update gate decides how much information from the previous hidden state will be updated to the current hidden state h_t. The candidate hidden state $\widetilde{h}_t$ and the hidden state h_t are computed using formulas (3) and (4), respectively. Subsequently, the retained information (hidden state h_t) is passed to the next GRU unit.

For the double-layer GRU neural network, each GRU unit in the second layer corresponds to a GRU unit in the first layer. The hidden state produced by each GRU unit in the first layer serves as the input for the connected GRU unit in the second layer. Finally, the fully connected layer and softmax function are employed to transform the final hidden state of the second-layer GRU neural network into a probability distribution for predicting the next log word. During the training phase, the model utilizes the cross-entropy as the loss function and employs stochastic gradient descent (SGD) to iteratively update the weight parameters. The calculation formula for the cross-entropy loss function is given by

$$\mathscr{L} = -\frac{1}{N}\sum_{1}^{N}\sum_{i=1}^{k} y_i \log(p_i) \tag{5}$$

where y_i represents the actual label, p_i is the probability value, k is the number of categories (the number of words in the word frequency table F), and N is the total number of samples.

4.4.2. Log Template Word Mining Stage. In the log template word mining stage, the input method for log data remains the same as in the training phase. The input consists of a log token subsequence $S_h = < s_{t-h}, \ldots, s_{t-2}, s_{t-1} >$, where h is the size of the sliding window, and $\forall s_i \in F$. The output of model is a probability distribution denoted as $P = (p_1, p_2, \ldots, p_{n-1}, p_n)$, which includes the probabilities associated with all words in the word frequency list F. Assuming that p_i represents the probability corresponding to the target log word s_t, we use p_i to indirectly indicate the association between s_t and input sequence $S_h = < s_{t-h}, \ldots, s_{t-2}, s_{t-1} >$. If s_t exhibits a strong association with the input sequence, it is determined to be a log template word; otherwise, it is the log parameter word. Figure 7 illustrates the sample log template mining process.

In fact, the final output of the model can be considered a binary classification problem. Based on prior experience, the target log word following an input sequence is not unique. Therefore, it is essential to manually set an appropriate probability threshold ρ when mining log template words. If the probability value of the target word exceeds the threshold ρ, the target word is considered to have a strong correlation with the input sequence, and it it is identified as a log template word. Conversely, Conversely, if the probability of the target word is below the threshold ρ, it is categorized as a log parameter word. To prevent mistakenly identifying log parameters as log template words, the extraction of template words for that sequence is halted when any target word in the sequence is identified as a parameter word (the first occurrence of a log parameter word within the sequence). Subsequently, processing continues with the next log word sequence until the entire log data has been processed.

After extracting the log template words corresponding to each log entry, the log entries should be divided into different log groups based on the log level, component name, and log template words. Log entries within each log group share the same log template words. For each log group, a data structure named "tem_i"is created to store the corresponding log template of that log group. A data structure named $\text{tem}_{\text{total}}$ is initialized as empty, which will store the final log template set, and $\text{Tem}_{\text{total}} = \text{tem}_1 \cup \text{tem}_2 \cup \ldots \cup \text{tem}_n$. Figure 8 illustrates the sample structure of the log template set.

4.5. Online Log Parsing Module. In the online log parsing stage, when a new log entry $\log_i$ arrives, DLLog first cleans, divides the original log entry and vectorizes to construct the log word frequency sequence S_i. This process has been

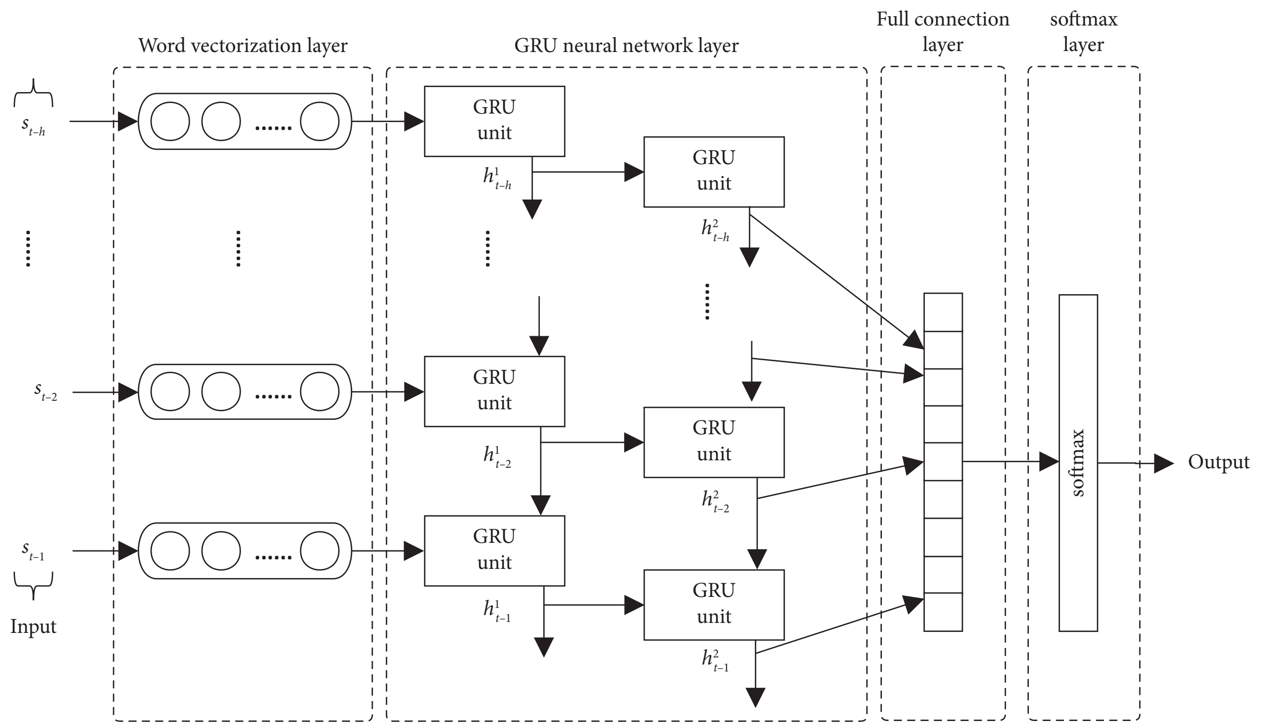


Figure 6: Structural of deep GRU network model.

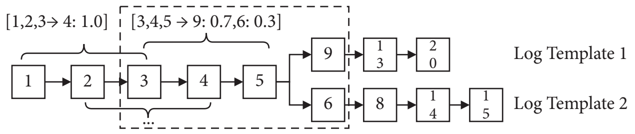


Figure 7: Example of log template mining process.

Log level	Component name	Log template
level$_1$	*component*$_1$	*tem*$_1$
		tem$_2$
	component$_2$	*tem*$_3$
level$_2$	*component*$_3$	*tem*$_4$
	component$_4$	*tem*$_5$
…	…	…
level$_n$	*component*$_n$	*tem*$_n$

Figure 8: Example of log template set structure.

described in Section 4.1. Then, DLLog compares the current log word frequency sequence S_i with the log templates in $\text{Tem}_{\text{total}}$ to determine whether S_i matches any existing log template tem_i, or if it should create a new log template to extend the log template set $\text{Tem}_{\text{total}}$. This section utilizes the longest common subsequence (LCS) to calculate the matching degree. Algorithm 1 shows the pseudo-code for online log parsing.

```
     Input: log word frequency sequence $S$ and log template set $\mathrm{Tem}_{\mathrm{total}}$
     Output: log template
 (1) Initialize optimal matching degree best = 0, optimal template length temlength = 0 and number $w = 0$
 (2) Initialize temporary log set new_logs
 (3) Go through all log templates in $\mathrm{Tem}_{\mathrm{total}}$ with the same loglevel and componentname as $S$ to form the log template set $\mathrm{Tem}_{\mathrm{same}}$
 (4) for $\mathrm{tem}_i$ in $\mathrm{Tem}_{\mathrm{same}}$ do
 (5)   $\mathscr{L}_i = |LCS(S, \mathrm{tem}_i)|$
 (6)   if $(\mathrm{best} < \mathscr{L}_i)$ or $(\mathrm{best} == \mathscr{L}_i$ and $|\mathrm{tem}_i| < h)$ then
 (7)     best = $\mathscr{L}_i$
 (8)     temlengh = $|\mathrm{tem}_i|$
 (9)     $w = i$
(10)   end if
(11) end for
(12) if $(\mathrm{best} < |S|/2)$ or $(\mathrm{best} < |tem_w|/2)$ then
(13)   newtem, $\mathrm{Tem}_{\mathrm{total}} = \mathrm{update}_{\mathrm{logtem}}(S, \mathrm{Tem}_{\mathrm{total}}, \mathrm{new\_logs})$//Call the log template set update function
(14)   if newtem == NULL then
         return NULL
(15)   else
         return newtem
(16)   end if
(17) else
         return $\mathrm{tem}_w$
(18) end if
```

Algorithm 1: Algorithm of online log parsing.

For a given current log word frequency sequence S, the first step is to search for log templates in the existing log template set $\mathrm{Tem}_{\mathrm{total}}$ with the same log type and component name as the current log word frequency sequence S. These matched log templates form a new set, $\mathrm{Tem}_{\mathrm{same}}$. Then, we calculate the matching degree between each log template in $\mathrm{Tem}_{\mathrm{same}}$ and the current log word frequency sequence S using the longest common subsequence (LCS) method [32–34]. The matching degree is determined by the length of the longest common subsequence. For instance, if there are three log templates (tem_1, tem_2, and tem_3) in $\mathrm{Tem}_{\mathrm{same}}$ that share the same log type and component name as the current log word sequence. The matching degrees between the current log word sequence and these log templates are denoted as $\mathscr{L}_1$, $\mathscr{L}_2$, and $\mathscr{L}_3$, which are calculated using the $\mathrm{LCS}(S, \mathrm{tem}_i)$.

The second step is to find the log template with the highest matching degree corresponding to the current log word frequency sequence S. If these log templates (tem_1, tem_2, and tem_3) share the same matching degree with the current sequence S, the system selects the log template with the shortest length as the corresponding log template. It is important to note that the matching degree, denoted as $\mathrm{LCS}(S, \mathrm{tem}_i)$, between the selected log template and the current log word sequence S should be greater than or equal to half the length of the current log word sequence and half the length of the selected log template. If, for any reason, the log template set $\mathrm{Tem}_{\mathrm{total}}$ cannot produce a match for the current log word sequence S, a new log template must be generated and added into the log template set $\mathrm{Tem}_{\mathrm{total}}$. If it is impossible to generate a new template based on the existing data, the current log word sequence S is stored in the temporary log set new_logs.

In each case, the examples are as follows:

(i) If the matching degree are ordered as $\mathscr{L}_1 > \mathscr{L}_3 > \mathscr{L}_2$, the system chooses $\mathscr{L}_1$ for the further processing.

(ii) If $\mathscr{L}_1 \geq |S|/2$, and $\mathscr{L}_1 \geq |\mathrm{tem}_1|/2$, $|S|$ is the length of the log word sequence and $|\mathrm{tem}_1|$ is the length of the log template tem_1, then the log template tem_1 is the final log template corresponding to the log word sequence S.

(iii) If $\mathscr{L}_1 < |S|/2$, or $\mathscr{L}_1 < |\mathrm{tem}_1|/2$, then a new template must be created for the current log word sequence S.

(iv) If $\mathscr{L}_1 = \mathscr{L}_3 > \mathscr{L}_2$, and $\mathscr{L}_1 = \mathscr{L}_3 \geq |S|/2$, $\mathscr{L}_1 \geq |\mathrm{tem}_1|/2$, $\mathscr{L}_3 \geq |\mathrm{tem}_3|/2$, DLLog selects the log template with the minimum length as the final log template corresponding to the log word sequence S by comparing the lengths of log templates $(|\mathrm{tem}_1| \mathrm{and} |\mathrm{tem}_3|)$.

The third step is to update the log template set. According to reference [23], when the system begins generating new types of system log entries, it often generates a substantial amount of log data of these new types within a single day. These log data typically contain numerous different parameter words. Consequently, new templates can be directly extracted by computing the longest common subsequence of these new types of logs. The pseudo-code of the log template set update algorithm is presented in Algorithm 2.

For the current log word sequence S, which fails to match any log template within the log template set $\mathrm{Tem}_{\mathrm{total}}$, it becomes necessary to calculate the longest common subsequence between S and each log entry in the temporary log

```
     Input: log word frequency sequence $S$, log template set $\text{Tem}_{\text{total}}$, and temporary log set new_logs
     Output: new log template and new log template set
(1)  Initialize optimal matching value best = 0, optimal matching length temlength = 0 and number $w = 0$
(2)  for $NS_i$ in new_logs do
(3)      if ($S$.level == $NS_i$.level) and ($S$.compoent == $NS_i$.compoent) then
(4)          $\mathscr{L}_i = |\text{LCS}(S, NS_i)|$
(5)          if (best < $\mathscr{L}_i$) or (best == $\mathscr{L}_i$ and temlengh > $|\text{LCS}(S, NS_i)|$) then
(6)              best = $\mathscr{L}_i$
(7)              temlengh = $|\text{LCS}(S, NS_i)|$
(8)              $w = i$
(9)          end if
(10)     end if
(11) end for
(12) if (best < $|S|/2$) or (best < $|NS_w|/2$) then
(13)     Add $S$ to new_logs//Add the log word sequence to the temporary log set
             return NULL
(14) else
(15)     newtem = $\text{LCS}(S, NS_w)$
(16)     Add $\text{LCS}(S, NS_w)$ to $\text{Tem}_{\text{total}}$//Add a new log template to the log template set
             return newtem, $\text{Tem}_{\text{total}}$
(17) end if
```

Algorithm 2: Algorithm of log template set update.

set new_{logs}. Subsequently, the optimal longest common subsequence is selected as the new log template. Similarly, this new template needs to be longer than or equal to half the length of both the current log word sequence and the selected log entry from new_{logs}. Once this condition is met, the log template can be added to the log template set $\text{Tem}_{\text{total}}$, thereby updating the set. The next time a new type of log entry of the same kind appears, the first two steps can be employed to match the log template.

5. Evaluation

This section first introduces the hardware and software environment, the experimental log dataset, and the evaluation metrics. Finally, specific experimental results are presented to demonstrate the superiority of DLLog.

5.1. Experimental Setting

5.1.1. Experimental Dataset. The log datasets used in this section consist of 16 real-world log datasets published by the LogPai team (https://github.com/logpai). In the LogHub data repository, these log data come from different systems, including distributed systems (HDFS, Hadoop, Spark, ZooKeeper, and OpenStack), supercomputers (BGL, HPC, and Thunderbird), operating system (Windows, Linux, and Mac), mobile system (Android, HealthApp), server applications (Apache, OpenSSH) and standalone software (Proxifier). LogHub log dataset can not only be used to measure the accuracy of log parsing methods but also test the robustness and efficiency of parsing methods. These datasets have been widely employed in similar research endeavors [15, 22, 26, 35]. Table 5 provides detailed information about these log datasets.

For each log dataset, Zhu et al. [11] sampled it and manually marked the log template of each log entry. In all experiments in this section, these markers were used as the basic factual basis for evaluation.

5.1.2. Evaluation Index. In the field of log parsing, parsing methods are typically evaluated using the Parsing Accuracy (PA) metric, as defined in reference [11]. PA is calculated as the ratio of correctly parsed log messages to the total number of log messages. Each log message corresponds to a specific log template, and log messages sharing the same log template are grouped into the same cluster, representing a particular type of log message. When assessing the correctness of parsed log messages, it is considered correct only when the log template corresponding to the log message is correctly divided into the log template cluster. In comparison to the evaluation metric (the *RandIndex*) used in prior studies [35–37], PA is considered a more rigorous measure.

5.1.3. Environment and Implementation. We have implemented the methods proposed in this chapter using the open-source Python machine learning library, PyTorch. All experiments were conducted in a consistent experimental environment using Python 3.8 with PyTorch 1.7.0. The hardware platform utilized for the experiments featured an AMD Ryzen 5 3600 6-core processor running at 3.6 GHz, an NVIDIA GTX1660 GPU, 128 GB of memory, and the Windows 10 64 bit operating system. We constructed our model based on the above environment. Specifically, during the offline training process and the log template mining process, it runs on a GPU to accelerate model training. The DLLog online parsing phase runs on a CPU to allow for a fair comparison with other log parsing methods.

Table 5: Summary of datasets.

Dataset	Dataset size	Logs	Log templates	Explanations
HDFS	1.47 GB	11,175,629	30	Distributed system logs
Hadoop	48.61 MB	394,308	298	Distributed system logs
Spark	2.75 GB	33,236,604	456	Distributed system logs
OpenStack	60.01 MB	207,820	51	Distributed system logs
ZooKeeper	9.95 MB	74,380	95	Distributed system logs
BGL	708.76 MB	4,747,963	619	Supercomputer logs
HPC	32.00 MB	433,489	104	Supercomputer logs
Thunderbird	29.60 GB	211,212,192	4,040	Supercomputer logs
Linux	2.25 MB	25,567	488	Operating system logs
Mac	16.09 MB	117,283	2,214	Operating system logs
Windows	26.09 GB	114,608,388	4,833	Operating system logs
OpenSSH	70.02 MB	655,146	62	Service application logs
Apache	4.90 MB	56,481	44	Service application logs
Android	3.38 GB	30,348,042	76,923	Mobile system logs
HealthApp	22.44 MB	253,395	220	Mobile system logs
Proxifier	2.42 MB	21,329	9	Standalone software logs

The number of training epochs is set to 300, the hidden dimensions of the GRU model are 64, and the number of layers is 2. In the Log Template Word Mining stage, the sliding window size, h, is set to 3, and the probability threshold, ρ, is set to 0.63. The learning rate is set to 0.001.

5.2. Accuracy Evaluation. In our experiments, we aimed to select state-of-the-art log parsing methods as comparison baselines. However, due to the unavailability of the source code for some methods [38, 39], such as Uniparser [38], we attempted to reproduce it for further experiments. Unfortunately, the parsing results of the reproduced model did not yield satisfactory outcomes on certain datasets. Consequently, to assess the accuracy of DLLog, we compared it with five baseline log parsing methods: Drain [22], Spell [26], Nulog [40], IPLoM [24], Logram [41], and Brain [42]. Drain is a tree-based log parsing method, Spell is a log parsing method based on the longest common subsequence, Nulog is a log parsing method based on a deep self-supervised learning model, IPLoM is a log parsing method based on iterative partition, Logram is a log parsing method based on the N-Gram statistical language model, and Brain is a rule-based log parsing method, specifically utilizing the longest common pattern. The 6 log parsing methods have been introduced in detail in Section 2. The comparison results of parsing accuracy are shown in Table 6. We set the best comparison result to bold.

As depicted in Table 6, DLLog achieves the best parsing accuracy in 7 out of the 16 log datasets, with an impressive average parsing accuracy of 0.891. Compared to state-of-the-art log parsing methods, the highest average parsing accuracy demonstrates the superiority of DLLog. DLLog also achieved high parsing accuracy scores on datasets where the optimal parsing accuracy was not attained. DLLog's average parsing accuracy is approximately 4% higher than Brain and Drain. In comparison to the relatively lower accuracy of the Logram method, DLLog's average parsing accuracy is 11% higher. However, we also observed that due to different rules in generating logs for various log systems, no log parsing method can achieve optimal parsing accuracy on all datasets.

Nearly every parsing method can achieve satisfactory parsing results for log datasets with simpler structures, such as HDFS and Apache log datasets; some methods even achieve the optimal parsing accuracy of 1. For log datasets with more complex structures, like HealthApp and HPC log datasets, the accuracy of each parsing method decreases to varying degrees. However, DLLog still attains the highest parsing accuracy on both datasets. The Spell method, which relies solely on the longest common subsequence for log parsing, achieves accuracies of 0.654 for HPC and 0.787 for BGL datasets. But DLLog based on deep learning and the longest common subsequence achieves accuracies of 0.996 for HPC and 0.988 for BGL datasets. This suggests that DLLog can effectively aid the model in parsing logs and enhance log parsing accuracy by fully utilizing the structural, frequency, and associative features of logs.

5.3. Versatility Evaluation. Experiment 2 evaluated the versatility of each log parsing method. The purpose is to verify whether the proposed method can widely support different log data types. Detailed statistics are given in Table 7, including the median (Median), minimum (Min.), standard deviation (STD), and interQuartile Range (IQR). Figure 9 shows the boxplot of the accuracy distribution for each log parsing method. For each box in Figure 9, the line from bottom to top represents the minimum observation value (Lower bound), the lower quartile (Q1), the median (Q2), the upper quartile (Q3), and the maximum observation value (upper bound). The length of each box represents the interQuartile Range of the corresponding log parsing method.

From Table 7 and Figure 9, it is clear that DLLog has the smallest InterQuartile Range of 0.186, and DLLog has the smallest standard deviation of 0.143, which is 2.0%, 11.1%, 17.8%, 7.7%, 21.8%, and 14.8% lower than Drain, Spell, Nulog, IPLoM, Logram, and Brain. This indicates that DLLog has the highest versatility and stability compared with other log parsing methods. The average parsing accuracy of Drain, Nulog, and Brain is basically the same, but

Table 6: Accuracy comparison of parsing results.

Dataset	Drain	Spell	Nulog	IPLoM	Logram	Brain	DLLog
HDFS	0.998	**1**	0.998	**1**	0.981	0.997	**1**
Hadoop	0.948	0.778	0.954	0.954	0.965	0.915	**0.992**
Spark	0.920	0.905	0.927	0.920	0.903	**0.997**	0.995
ZooKeeper	0.967	0.964	**0.990**	0.955	0.725	0.987	0.979
OpenStack	0.733	0.764	0.989	0.871	0.545	**1**	0.831
BGL	0.963	0.787	0.954	0.939	0.788	0.978	**0.988**
HPC	0.887	0.654	0.912	0.824	0.824	0.727	**0.996**
Thunderbird	0.955	0.844	0.912	0.663	0.761	**0.971**	0.783
Windows	**0.997**	0.989	0.990	0.567	0.957	0.718	0.992
Linux	**0.690**	0.605	0.468	0.672	0.461	0.480	0.672
Mac	0.787	0.757	0.826	0.673	0.673	**0.952**	0.856
Android	0.911	**0.919**	0.822	0.712	0.848	0.817	0.905
HealthApp	0.780	0.639	0.876	0.822	0.822	0.875	**0.990**
Apache	**1**	**1**	**1**	**1**	**1**	**1**	**1**
OpenSSH	0.788	0.554	0.603	0.802	0.553	**0.819**	0.751
Proxifier	0.526	**0.527**	0.515	0.515	0.504	0.526	**0.527**
Average	0.865	0.792	0.858	0.806	0.783	0.859	**0.891**

The bold values represent the best comparison result.

Table 7: Comparison of detailed accuracy results.

	Drain	Spell	Nulog	IPLoM	Logram	Brain	DLLog
Median	0.865	0.792	0.858	0.806	0.783	0.859	**0.891**
Min	0.526	0.527	0.468	0.515	0.461	0.480	**0.527**
STD	0.146	0.161	0.174	0.155	0.183	0.168	**0.143**
IQR	0.201	0.295	0.189	0.274	0.343	0.220	**0.186**

The bold values represent the best comparison result.

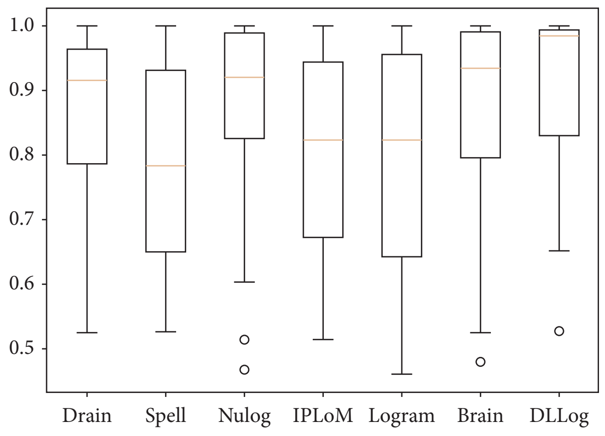


Figure 9: Boxplot comparison of accuracy distribution.

Drain is better than Nulog and Brain in terms of stability. In contrast, the versatility of Logram log parsing methods needs to be improved. Overall, DLLog is superior to the other five comparison methods in accuracy and versatility.

5.4. Efficiency Evaluation. Due to the system producing a large amount of log data in real-time, the online parsing efficiency of log parsing methods must also be considered. Experiment 3 verifies the running time spent by the five methods to parse all HDFS and BGL log entries (2.16 G in total). The experimental results are shown in Figure 10, the ordinate represents the parsing time, and the abscissa represents the size of log data volume.

As illustrated in Figure 10, DLLog exhibits a linear growth trend with the increase of log data in both log datasets. Parsing the BGL log dataset takes more time for each log parsing method compared to the HDFS log dataset, as the HDFS log dataset has 30 templates while the BGL log dataset has 619 templates (20 times more than the HDFS log dataset). Logram demonstrates the fastest parsing speed

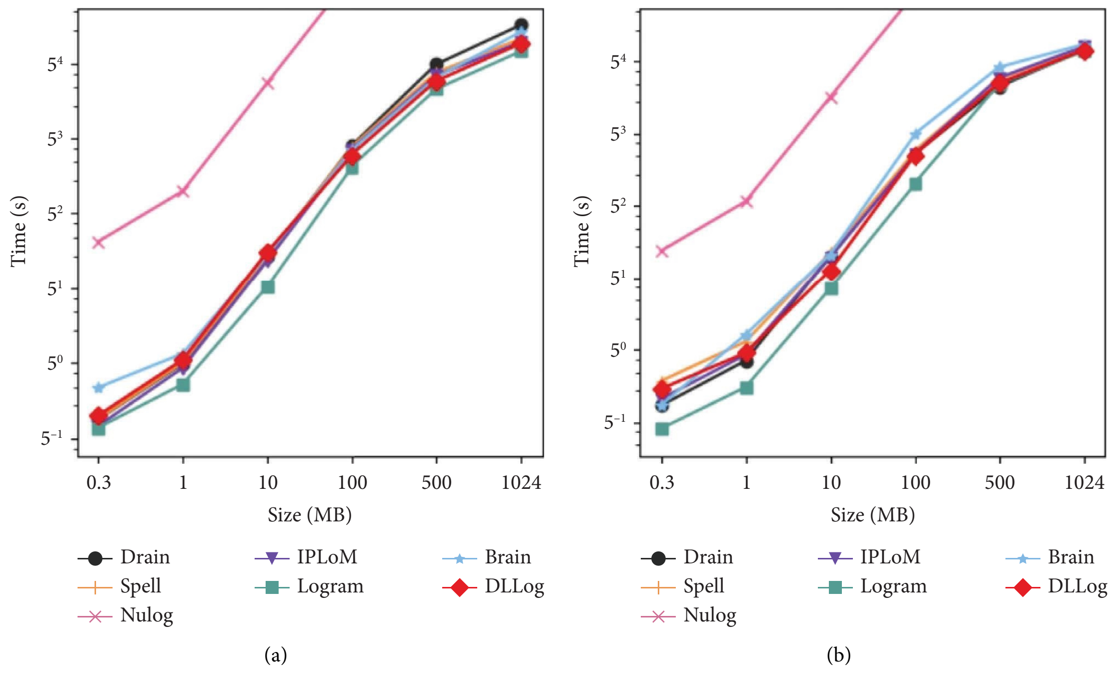


Figure 10: Comparison of log parsing time. (a) HDFS. (b) BGL.

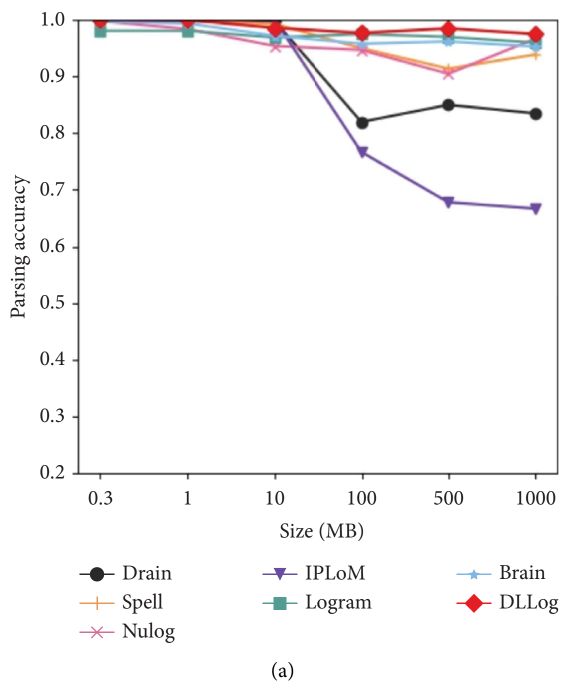
when the dataset size is less than 100 MB. This is because Logram, based on n-gram, calculates frequencies simply by counting, which saves a significant amount of time compared to other parsing methods based on complex rules. We found that Nulog has the slowest parsing speed because throughout the entire parsing process, Nulog, based on deep learning models, continuously needs to retrain the model for parsing.

Although DLLog also requires training deep learning models, it is only used during the initial offline parsing stage (with a data size of 0.3 M). In the online log parsing stage, DLLog pre-classifies log templates before template matching to compare the similarity of newly arrived logs with existing log templates. Thus, for similarity comparison, only a comparison between the newly arrived log and log templates that meet specific categories is required. This strategy significantly reduces template matching time. While DLLog does not achieve the optimal parsing speed compared to the 6 parsing methods, its parsing speed remains within an acceptable range, and it attains the highest parsing accuracy and the best versatility.

5.5. Incremental Update Evaluation. The maintenance and upgrade of systems result in the generation of new log data. Therefore, it is crucial to consider the performance of log parsing methods when dealing with newly emerging log types. Experiment 4 evaluated the update capabilities of 7 parsing methods on the HDFS and Android datasets, chosen due to their volumes and the availability of ground truths for such evaluations. We used an initial data volume of 2 K (approximately 0.3 M) for each dataset for model training. Subsequently, we processed the trained model with data volumes of 1 M, 10 M, 100 M, 500 M, and 1000 M for each dataset. With the increase in the number of logs, new log types may emerge. For instance, the 2 k HDFS logs are generated from 14 log templates, while the 1000 MB HDFS dataset contains 29 log templates. An excellent log parser should exhibit stable accuracy when introducing new logs accurately parsing new log types. The experimental results are shown in Figure 11, the ordinate represents the parsing accuracy, and the abscissa represents the size of log data volume.

We can observe that, with the increase in volume and the introduction of new types of log data, DLLog demonstrates optimal stability, indicating its effective handling of new log types. However, it's worth noting that the parsing performance of all log parsing methods on the HDFS dataset surpasses that on the Android dataset. This difference can be attributed to the significantly larger number of log templates in the Android dataset, making parsing more complex. DLLog's exceptional incremental update capability, derived from the combination of deep neural networks and the longest common subsequence, enables it to effectively process new log types. Consequently, even with an increase in log data volume, the decline in parsing accuracy is not significant.

5.6. Ablation Experiment. In the process of constructing the log word frequency sequence, we conducted ablation experiments using two sequences: one sorted according to our method and the other unsorted, to verify the effectiveness of our approach. We conducted a comparative experiment between DLLog based on LSTM and DLLog based on GRU. The experimental results are illustrated in Table 8.



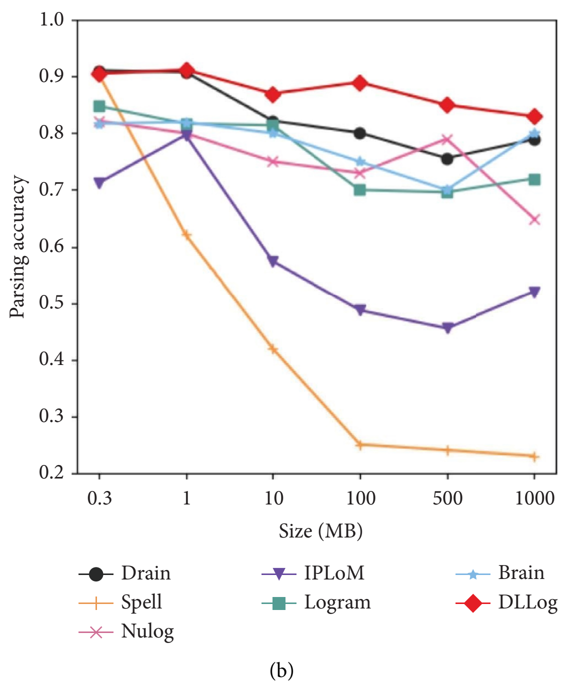


Figure 11: Group accuracy on different volumes of logs. (a) HDFS. (b) Android.

Table 8: Ablation experiment comparison of parsing results.

Dataset	DLLog	DLLog_Unsort	DLLog_LSTM
HDFS	**1**	0.546	**1**
Hadoop	**0.992**	0.563	0.981
Spark	**0.995**	0.545	0.983
ZooKeeper	0.979	0.563	**0.987**
OpenStack	**0.831**	0.471	0.831
BGL	**0.988**	0.735	0.988
HPC	**0.996**	0.323	0.996
Thunderbird	**0.783**	0.357	0.783
Windows	0.992	0.655	**0.997**
Linux	**0.672**	0.243	0.672
Mac	**0.856**	0.478	0.845
Android	0.905	0.493	**0.961**
HealthApp	**0.990**	0.592	0.905
Apache	**1**	0.582	**1**
OpenSSH	**0.751**	0.518	0.619
Proxifier	**0.527**	0.387	**0.527**
Average	**0.891**	0.502	0.872

The bold values represent the best comparison result.

As depicted in Table 8, DLLog is significantly influenced by whether the log word frequency sequence is sorted. DLLog based on the sorted log word frequency sequence exhibits an average parsing accuracy that is 39% higher than the unsorted version. Moreover, substantial improvements are observed on each dataset, such as Thunderbird, Linux, etc. This is attributed to the presence of numerous templates and variable parameter words in these datasets, making the sorting of the log word frequency sequence more impactful on DLLog's training. We processed the sequence of word frequencies transformed based on the frequency table *F* in ascending order, leveraging the characteristics of high-frequency templates appearing more frequently and the construction method of the frequency table. This arrangement ensures that high-frequency template words form a "fixed" combination, with dynamic parameters following, allowing the GRU neural network to accurately learn the log pattern combinations.

On most datasets, the parsing accuracy of DLLog based on GRU is similar to that based on LSTM, but there is a significant difference in parsing accuracy on OpenSSH and Linux. We believe that in the majority of cases, the classifications of the two are consistent. Only in a few log sequences are the classifications by LSTM and GRU are different, leading to some inaccuracies in log grouping. Moreover, these log groups constitute a significant portion, resulting in a substantial difference in parsing accuracy between the two, according to the formula for grouping accuracy calculation. We attribute the difference in parsing accuracy between GRU and LSTM to the fact that GRU, as a simplified variant of LSTM, predicts more accurately in short sequence datasets, and logs are a type of short sequence data. Conversely, for longer log sequence data, LSTM performs slightly better than GRU.

6. Conclusions

In this paper, we proposed DLLog, an online log parsing method for accurately and incrementally parsing templates without the need for domain-specific knowledge. DLLog leverages the GRU neural network for offline template word mining and leverages the longest common subsequence for parsing log entries in real-time. By utilizing multiple log entry features, DLLog can autonomously extract template words, eliminating the requirement for manual intervention and enhancing its versatility in parsing unstructured log data. Additionally, DLLog supports incremental updates of the log template set, making it adaptable to newly generated log entries in evolving systems. We conducted a comprehensive evaluation of the DLLog parsing method on multiple extensive log datasets, and the experimental results unequivocally demonstrated its remarkable accuracy, universality, and adaptability when parsing large-scale log data. In our future research endeavors, we intend to incorporate location information and character features of log words to assist the log parsing method in distinguishing between log parameter words and log template words. This endeavor aims to further enhance the precision and effectiveness of DLLog.

Data Availability

Data deposited in a repository (https://www.usenix.org/cfdr/).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The work was supported by National Natural Science Foundation of China under grant no. 61672392 and no. 62072342.

References

[1] D. Wu, Y. Deng, and M. Li, "Fl-mgvn: federated learning for anomaly detection using mixed Gaussian variational self-encoding network," *Information Processing & Management*, vol. 59, no. 2, Article ID 102839, 2022.

[2] T. Xiao, Z. Quan, Z.-J. Wang, K. Zhao, and X. Liao, "LPV: a log parser based on vectorization for offline and online log parsing," in *Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM)*, pp. 1346–1351, Sorrento, Italy, November 2020.

[3] Y.-Q. Zhu, J.-Y. Deng, J.-C. Pu, P. Wang, S. Liang, and W. Wang, "Ml-parser: an efficient and accurate online log parser," *Journal of Computer Science and Technology*, vol. 37, no. 6, pp. 1412–1426, 2022.

[4] S. Huang, Y. Liu, C. Fung et al., "Paddy: an event log parsing approach using dynamic dictionary," in *Proceedings of the NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–8, Budapest, Hungary, April 2020.

[5] W. Meng, Y. Liu, Y. Huang et al., "A semantic-aware representation framework for online log analysis," in *Proceedings of the 2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–7, Honolulu, HI, USA, August 2020.

[6] Z. A. Khan, D. Shin, D. Bianculli, and L. Briand, "Guidelines for assessing the accuracy of log message template identification techniques," in *Proceedings of the 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pp. 1095–1106, Pittsburgh, PA, USA, May 2022.

[7] X. Duan, S. Ying, W. Yuan, H. Cheng, and X. Yin, "Qllog: a log anomaly detection method based on q-learning algorithm," *Information Processing & Management*, vol. 58, no. 3, Article ID 102540, 2021.

[8] H. Liang, L. Song, J. Wang, L. Guo, X. Li, and J. Liang, "Robust unsupervised anomaly detection via multi-time scale dcgans with forgetting mechanism for industrial multivariate time series," *Neurocomputing*, vol. 423, pp. 444–462, 2021.

[9] S. Tao, W. Meng, Y. Cheng et al., "Logstamp: automatic online log parsing based on sequence labelling," *SIGMETRICS Performance Evaluation Review*, vol. 49, no. 4, pp. 93–98, 2022.

[10] I. Sedki, A. Hamou-Lhadj, O. Ait-Mohamed, and M. A. Shehab, "An effective approach for parsing large log files," in *Proceedings of the 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 1–12, Limassol, Cyprus, October 2022.

[11] J. Zhu, S. He, J. Liu et al., "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)*, pp. 121–130, Montreal, Canada, May 2019.

[12] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: an update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[13] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *Prooceedings of the 2009 9th IEEE International Conference on Data Mining*, pp. 149–158, Miami Beach, FL, USA, December 2009.

[14] X. Wang, X. Zhang, L. Li et al., "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*, pp. 1198–1208, Association for Computing Machinery, Singapore, November 2022.

[15] M. Raynal, M.-O. Buob, and G. Quénot, "A novel pattern-based edit distance for automatic log parsing: implementation and reproducibility notes," in *Reproducible Research in Pattern Recognition: 4th International Workshop, RRPR 2022, Montreal, Canada, August 21, 2022, Revised Selected Papers*, pp. 43–50, Springer-Verlag, Berlin, Germany, 2023.

[16] C. Zhang, X. Wang, H. Zhang et al., "Layerlog: log sequence anomaly detection based on hierarchical semantics," *Applied Soft Computing*, vol. 132, Article ID 109860, 2023.

[17] R. Tian, Z. Diao, H. Jiang, and G. Xie, "Logdac: a universal efficient parser-based log compression approach," in *Proceedings of the ICC 2022-IEEE International Conference on Communications*, pp. 3679–3684, Seoul, Korea, May 2022.

[18] A. Namavar Jahromi, S. Hashemi, A. Dehghantanha et al., "An improved two-hidden-layer extreme learning machine for malware hunting," *Computers and Security*, vol. 89, Article ID 101655, 2020.

[19] A. Javed, P. Burnap, and O. Rana, "Prediction of drive-by download attacks on twitter," *Information Processing and Management*, vol. 56, no. 3, pp. 1133–1145, 2019.

[20] H. Guo, S. Yuan, and X. Wu, "Logbert: log anomaly detection via bert," in *Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, Shenzhen, China, July 2021.

[21] S. Kabinna, C. P. Bezemer, W. Shang, M. D. Syer, and A. E. Hassan, "Examining the stability of logging statements," *Empirical Software Engineering*, vol. 23, no. 1, pp. 290–333, 2018.

[22] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: an online log parsing approach with fixed depth tree," in *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, pp. 33–40, Honolulu, HI, USA, June 2017.

[23] S. Zhang, W. Meng, J. Bu et al., "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *Proceedings of the 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, Vilanova i la Geltrú, Spain, June 2017.

[24] A. A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, "Clustering event logs using iterative partitioning," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '09*, pp. 1255–1264, Association for Computing Machinery, New York, NY, USA, November 2009.

[25] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, "Abstracting execution logs to execution events for enterprise applications (short paper)," in *Proceedings of the 2008 the 8th International Conference on Quality Software*, pp. 181–186, Oxford, UK, August 2008.

[26] M. Du and F. Li, "Spell: online streaming parsing of large unstructured system logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 11, pp. 2213–2227, 2019.

[27] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, October 2014.

[28] D. Song, A. Vold, K. Madan, and F. Schilder, "Multi-label legal document classification: a deep learning-based approach with label-attention and domain-specific pre-training," *Information Systems*, vol. 106, Article ID 101718, 2022.

[29] Y. Liu, A. Pei, F. Wang et al., "An attention-based category-aware gru model for the next poi recommendation," *International Journal of Intelligent Systems*, vol. 36, no. 7, pp. 3174–3189, 2021.

[30] B. Zhang, D. Xiong, J. Xie, and J. Su, "Neural machine translation with gru-gated attention model," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 11, pp. 4688–4698, 2020.

[31] Y. Liu, Z. Song, X. Xu et al., "Bidirectional gru networks-based next poi category prediction for healthcare," *International Journal of Intelligent Systems*, vol. 37, no. 7, pp. 4020–4040, 2022.

[32] Y. Du, Y. Zhong, F. Chen, Q. Huang, and Q. Hu, "Matching method based on similarity of working trajectories," *International Journal of Intelligent Systems*, vol. 37, no. 12, pp. 11871–11892, 2022.

[33] Y. Xiao, H. Yin, Y. Zhang, H. Qi, Y. Zhang, and Z. Liu, "A dual-stage attention-based conv-lstm network for spatio-temporal correlation and multivariate time series prediction," *International Journal of Intelligent Systems*, vol. 36, no. 5, pp. 2036–2057, 2021.

[34] O. Koren, C. A. Hallin, M. Koren, and A. A. Issa, "Automl classifier clustering procedure," *International Journal of Intelligent Systems*, vol. 37, no. 7, pp. 4214–4232, 2022.

[35] G. Chu, J. Wang, Q. Qi, H. Sun, S. Tao, and J. Liao, "Prefix-graph: a versatile log parsing approach merging prefix tree with probabilistic graph," in *Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pp. 2411–2422, Chania, Greece, April 2021.

[36] W. Meng, Y. Liu, F. Zaiter et al., "Logparse: making log parsing adaptive through word classification," in *Proceedings of the2020 29th International Conference on Computer Communications and Networks (ICCCN)*, pp. 1–9, Honolulu, HI, USA, August 2020.

[37] G. Lee and K. Lee, "Online dependence clustering of multivariate streaming data using one-class svms," *International Journal of Intelligent Systems*, vol. 37, no. 6, pp. 3682–3708, 2022.

[38] Y. Liu, X. Zhang, S. He et al., "Uniparser: a unified log parser for heterogeneous log data," in *Proceedings of the ACM Web Conference 2022, WWW '22*, pp. 1893–1901, Association for Computing Machinery, New York, NY, USA, October 2022.

[39] Y. Fu, M. Yan, J. Xu et al., "Investigating and improving log parsing in practice," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1566–1577, ESEC/FSE 2022, Singapore, November 2022.

[40] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-supervised log parsing," in *Proceedings of the Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020*, pp. 122–138, Ghent, Belgium, September 2021.

[41] H. Dai, H. Li, C.-S. Chen, W. Shang, and T.-H. Chen, "Logram: efficient log parsing using nn-gram dictionaries," *IEEE Transactions on Software Engineering*, vol. 48, no. 3, pp. 879–892, 2022.

[42] S. Yu, P. He, N. Chen, and Y. Wu, "Brain: log parsing with bidirectional parallel tree," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3224–3237, 2023.