2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE) | 978-1-6654-5701-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE48619.2023.00204

Log Parsing with Prompt-based Few-shot Learning

Van-Hoang Le[1] and Hongyu Zhang[2†]
[1]School of Information and Physical Sciences, The University of Newcastle, Australia
[2]School of Big Data and Software Engineering, Chongqing University, China
vanhoang.le@uon.edu.au, hyzhang@cqu.edu.cn

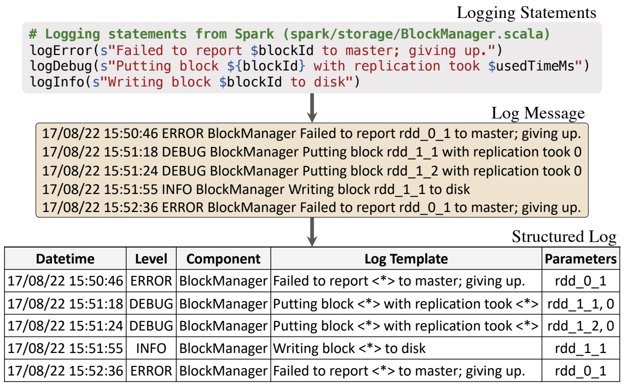
***Abstract*—Logs generated by large-scale software systems provide crucial information for engineers to understand the system status and diagnose problems of the systems. Log parsing, which converts raw log messages into structured data, is the first step to enabling automated log analytics. Existing log parsers extract the common part as log templates using statistical features. However, these log parsers often fail to identify the correct templates and parameters because: 1) they often overlook the semantic meaning of log messages, and 2) they require domain-specific knowledge for different log datasets. To address the limitations of existing methods, in this paper, we propose LogPPT to capture the patterns of templates using prompt-based few-shot learning. LogPPT utilises a novel prompt tuning method to recognise keywords and parameters based on a few labelled log data. In addition, an adaptive random sampling algorithm is designed to select a small yet diverse training set. We have conducted extensive experiments on 16 public log datasets. The experimental results show that LogPPT is effective and efficient for log parsing.**

***Index Terms*—log parsing, few-shot learning, prompt-tuning, deep learning**

I. Introduction

Large-scale software-intensive systems often produce a large volume of logs to record runtime status and events for troubleshooting purposes. Logs play an important role in the maintenance and operation of software systems, which allow engineers to better understand the system's behaviours and diagnose problems. The rich information included in log data enables a variety of log analytics tasks, such as anomaly detection [1], [2], [3], [4], root cause analysis [5], [6], failure prediction [7], [8], and log compression [9], [10]. Among them, the first and foremost step is log parsing, which parses free-text raw log messages into a structured format [11]. The structured log data from log parsing are fed to various machine learning (ML) or deep learning (DL) models to perform many downstream analysis tasks.

Log parsing is the task of converting a raw log message into a specific log template. As shown in Figure 1, log messages are generated from logging statements in the source code. A log message usually contains a header that is automatically produced by the logging framework and includes information such as component and verbosity level. The log message body (log message for short) typically consists of two parts: 1) *Template* - constant strings (or keywords) describing the system events; 2) *Parameters* - dynamic variables, which vary during runtime and reflect system runtime information. For example, in the first log message in Figure 1, the header (i.e., "`17/08/22 15:50:46`", "`INFO`", and "`BlockManager`") can be easily distinguished through regular expressions. The log message consists of a template "`Putting block <*> with replication took <*>`" and the parameters including "`rdd_1_1`" and "`0`".



```
# Logging statements from Spark (spark/storage/BlockManager.scala)
logError(s"Failed to report $blockId to master; giving up.")
logDebug(s"Putting block ${blockId} with replication took $usedTimeMs")
logInfo(s"Writing block $blockId to disk")
```

```
17/08/22 15:50:46 ERROR BlockManager Failed to report rdd_0_1 to master; giving up.
17/08/22 15:51:18 DEBUG BlockManager Putting block rdd_1_1 with replication took 0
17/08/22 15:51:24 DEBUG BlockManager Putting block rdd_1_2 with replication took 0
17/08/22 15:51:55 INFO BlockManager Writing block rdd_1_1 to disk
17/08/22 15:52:36 ERROR BlockManager Failed to report rdd_0_1 to master; giving up.
```

Datetime	Level	Component	Log Template	Parameters
17/08/22 15:50:46	ERROR	BlockManager	Failed to report <*> to master; giving up.	rdd_0_1
17/08/22 15:51:18	DEBUG	BlockManager	Putting block <*> with replication took <*>	rdd_1_1, 0
17/08/22 15:51:24	DEBUG	BlockManager	Putting block <*> with replication took <*>	rdd_1_2, 0
17/08/22 15:51:55	INFO	BlockManager	Writing block <*> to disk	rdd_1_1
17/08/22 15:52:36	ERROR	BlockManager	Failed to report <*> to master; giving up.	rdd_0_1

Fig. 1. An example of log parsing from Spark

To achieve automated log parsing, many data-driven approaches [12], [13], [14], [15] have been proposed over the years to extract the common parts that constantly occur among log messages as templates and the dynamic parts that vary during runtime as parameters. Although making progress, existing log parsers still suffer from unsatisfactory accuracy, which may significantly affect the follow-up analysis such as log-based anomaly detection [16]. For example, the existing state-of-the-art log parsers Drain [12] and AEL [17] only achieve an average Parsing Accuracy of 0.34 and 0.28 on 16 log datasets [18]. We have identified the following limitations of the existing log parsers:

- **Accuracy:** Existing log parsers extract common parts as templates using statistical features (e.g., word length, log length, frequency) and ignore the semantic meaning of log messages. Without considering the semantic information, traditional log parsers tend to misidentify parameters as keywords [19] in many cases (e.g., when encountering previously unseen log templates).
- **Robustness:** Existing log parsers are not robust across different types of logs because they require domain-specific knowledge for different datasets [20]. The domain-specific knowledge includes data pre-processing (e.g.,

[†]Hongyu Zhang is the corresponding author.

1558-1225/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE48619.2023.00204

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

defining regular expressions) and hyper-parameter settings (e.g., the number of clusters or similarity threshold). The accuracy of these log parsers could be significantly affected by the input domain-specific knowledge. For example, without the pre-processing step, the parsing accuracy can decline by 6.1%-73.5% [21]. When applying the existing log parsers to a new log dataset, due to different logging formats and behaviours, time-consuming adjustment of hyper-parameters and regular expressions are needed [19].

To overcome the above-mentioned limitations, in this paper, we propose LogPPT, a novel log parser with prompt-based few-shot learning. LogPPT is able to capture the semantic information of log messages to identify keywords and parameters in log messages by learning from only a few labelled log messages. First, we design an Adaptive Random Sampling algorithm that can sample a small and diverse set of log messages to label as the training data. The training data is a set of labelled logs that contain raw log messages and the corresponding ground truth templates. Second, to effectively train a model with a few labelled log data, we tune a pre-trained language model (e.g., RoBERTa [22]) to predict a specific virtual label token (“PARAM”, an acronym for parameters) at the position of parameters in the log message in a few-shot learning manner. The embedding vector for the virtual label token “PARAM” is generated based on the word distribution from language model predictions and the unlabelled log dataset. After training, LogPPT can be directly applied to parse new log data. Our proposed method does not require any pre-processing step and uses the same set of hyper-parameter values for different datasets, which is robust across different logging formats and behaviours, and more generalised than existing approaches.

We have evaluated LogPPT on 16 public log datasets [11]. LogPPT achieves over 0.9 average Group Accuracy [11] and Parsing Accuracy [18], [19] when using only 32 labelled samples. The experimental results show that LogPPT is effective and efficient. It outperforms state-of-the-art parsers by 16% on Group Accuracy [11] and about 84% on Parsing Accuracy [19]. Moreover, LogPPT is also robust across different log datasets.

To summarise, our main contributions are as follows:

- We propose LogPPT, a prompt-based few-shot log parser that can precisely capture the patterns of templates and parameters in log messages. LogPPT uses a novel prompt tuning method to effectively learn the semantic information from a few labelled log samples. The proposed approach does not require manually-defined regular expressions for pre-processing and uses the same set of hyper-parameter values for every dataset, thus can quickly adapt to new log datasets.
- We evaluate LogPPT on 16 public log datasets, and the results demonstrate that LogPPT outperforms existing approaches. The experimental results confirm the effectiveness and efficiency of our proposed method.

II. Background and Motivation

A. Log Parsing

Log parsing is one of the first steps for log analysis tasks [11]. It is a process to extract the static log template parts and the corresponding dynamic parameters (or variables) from free-text raw log messages. For example, Figure 1 shows an example of logs of the Spark system, where Datetime, Component, and Level fields are the log header generated by the logging framework and are generally easy to extract. The log template “Putting block <*> with replication took <*>” associated with parameters (e.g., “rdd_1_1”, “0”), in contrast, is often difficult to identify. The goal of log parsing is to convert each log message into a specific log template and extract the corresponding parameters [11], [19].

The straightforward way of log parsing relies on handcrafted regular expressions or grok patterns to extract log templates and parameters [11]. However, manually writing regular expressions to parse a huge volume of logs is time-consuming and error-prone [11]. Some studies [23], [24] extract the log templates from logging statements in the source code to compose regular expressions for log parsing. However, it is not applicable in practice since the source code is often unavailable, especially for third-party libraries [11]. Therefore, regular expression matching often serves as a pre-processing step to (1) separate headers and content (which contains log templates and dynamic parameters) from raw log messages, and (2) abstract some special information such as IP address and ID to improve parsing accuracy. To achieve the goal of automated log parsing, many data-driven approaches have been proposed to identify log templates as the frequent part of log messages. Data-driven log parsing approaches can be divided into three main groups:

1) Frequent pattern mining. Some approaches, including SLCT [25], LFA [15], and Logram [14], find frequent patterns which emerge constantly across the entire log dataset. They leverage the token position or n-gram information to extract log templates based on frequent pattern mining.

2) Similarity-based clustering. These approaches apply various clustering algorithms to group similar logs and consider logs under the same group belonging to the same template. Representative methods include LKE [26], LogSig [27], and LenMa [28], which compute distances between two log messages or their signature to cluster them based on similarity.

3) Heuristics-based parsing. AEL [17], Spell [13], or Drain [12] propose heuristics-based log parsing methods that leverage unique characteristics from log messages to extract common templates efficiently.

Although making progress, traditional log parsers are still criticized for unsatisfactory parsing accuracy due to the omission of semantic information or improper evaluation metrics. Recent studies [18], [19] show that traditional approaches focus more on grouping logs and fail to identify the correct templates and parameters. For example, in Figure 1, some tokens (such as “rdd_0_1” and “0”) are identified as keywords by traditional log parsers because they do not vary in different log messages. However, these tokens should be classified

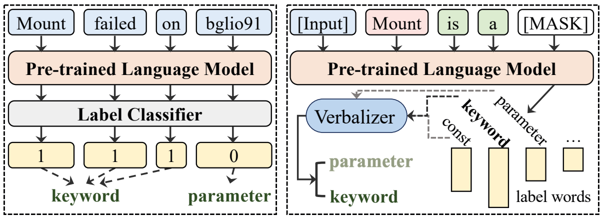
Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

as parameters considering their semantic meanings. Besides, existing log parsers are not robust across different log datasets. They require domain-specific knowledge to define regular expressions for pre-processing of different log data [20]. For example, on the HDFS dataset [21], [29], *block_id* (e.g., "`blk_-6670958622368987959`") information is abstracted from logs by using a regular expression "`blk_-?\d+`". For a new dataset such as BGL [21], [30], this regular expression must be changed to match the *core_id* such as "`core.2275`" (i.e., "**`blk`**`_-?\d+`" → "**`core.`**`\d+`"). Moreover, existing log parsers require specific hyper-parameters (e.g., number of clusters or similarity threshold) for different datasets to optimize the performance. For example, Drain [12] uses a low *similarity threshold* of 0.2 for the HealthApp dataset and a high *threshold* of 0.6 for the Proxifier dataset [11]. Due to different logging formats and behaviours, when facing a new log dataset, existing log parsers have to adjust the hyper-parameters and reconfigure the regular expressions for pre-processing [19].

B. Language Models

1) Pre-training and Fine-tuning: Pre-trained models have been shown effective in many natural language processing (NLP) tasks. These language models (LM), such as BERT [31] and T5 [32], are generally pre-trained using the Masked Language Modelling (MLM) objective. During the pre-training phase, the model learns to predict randomly masked input tokens. Based on the idea that log is actually a natural language sequence [2], some studies [16], [33], [34] have leveraged pre-trained language models such as BERT [31] to analyse log data. Language models are pre-trained on large-scale unlabelled corpus and then fine-tuned to perform downstream tasks.

Fine-tuning a pre-trained model for downstream tasks [31], [35] is a prevalent paradigm in the NLP field that further trains the model in a supervised way. As shown in Figure 2(a), a straightforward way to apply fine-tuning for log parsing is to convert the log parsing task into the token classification problem. The model can easily extract keywords and form log templates by classifying whether a token in log messages is a keyword or parameter (binary classification) using an additional classifier. However, the inconsistency between pre-training objectives and the fine-tuning objective (i.e., classification) restrains the use of rich knowledge distributed in pre-trained models [36], [37], leading to sub-optimal results. Besides, the performance of fine-tuning significantly depends on the scale of downstream data.



a) Fine-tuning b) Standard prompt tuning

Fig. 2. An illustration of fine-tuning and prompt tuning for log parsing

2) Prompt Tuning: Recently, prompt tuning [37], [38], [39], [40] has been proposed to close the gap between pre-training and downstream tasks. Figure 2(b) illustrates the concept of prompt tuning. Instead of designing a new training objective for each downstream task, prompt tuning rewrites the input by adding a natural language instruction such as "[S] is a [MASK]" to reuse the masking objective for downstream tasks. Formally, standard prompt tuning employs a prompt template $T_{prompt}(.)$ to convert the input X to prompt input $X_{prompt} = T_{prompt}(X)$. The prompt template is a textual string with unfilled slots to fill the input X and a label slot [MASK].

For log parsing, a standard prompt template consists of three unfilled slots to fill the input log message, the token needed to be identified, and the label for the processing token. For example, in Figure 2(b), the prompt template is in the form of "[X] [S] is a [MASK]", where [X], [S], and [MASK] are the unfilled slots for the input log message, token, and label, respectively. The LMs then try to fill the label slot [MASK] with label words such as *keyword* or *variable*. After that, a verbalizer is used to map each predicted label word to a class for the input token. In Figure 2(b), the verbalizer contains label words sets of "[*const, keyword*]" for keywords and "[*parameter*]" for parameters. By enumerating over all tokens in a log message, we can extract the corresponding template and parameters.

According to the flexibility of the prompt template, standard prompt tuning techniques can be categorized into two types: hard prompt and soft prompt. We briefly introduce each prompt type in the following.

Hard Prompt. Hard prompt or discrete prompt [37], [38] is a technique that modifies the input by adding fixed natural language instructions. Hard prompt templates usually correspond to natural language phrases [41], in which each token in prompt templates is meaningful and understandable. Although hard prompt has shown promising performance [38], the template design and the label word choices are challenging because it requires task-specific knowledge.

Soft Prompt. Soft prompt [42], [43] is an alternative to hard prompt. Instead of using fixed discrete words as in hard prompt, soft prompt uses *virtual tokens*, which are in the form of continuous vectors and can be learnt during the tuning stage, to construct prompt templates. The soft prompt is proposed to remove the constraints of manually selecting a prompt template in the hard prompt.

Although achieving promising results in various NLP tasks, standard prompt tuning is insufficient for the log parsing task because (1) it needs to enumerate all span candidates, which is inelegant and time-consuming [40], and (2) it is sensitive to noises (see Section VI-A for details).

In this paper, we apply prompt tuning to achieve the goal of log parsing with a few labelled training data. However, instead of using standard prompt tuning, we leverage the paradigm of template-free prompt [39] for log parsing, which does not require prompt templates as the instruction. In template-free prompt [39], an additional *virtual label token* is generated and plays the role of prompt instructions as in standard prompt tuning. Then, the model learns to predict the *virtual label token*

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

at the positions of parameters and the original token at the positions of keywords using a custom MLM objective. The template-free prompt tuning method [39] addresses the major limitations of standard prompt by (1) relaxing the burden of manually selecting prompt templates [39] and (2) performing one-pass decoding to process all tokens simultaneously, which is more efficient compared to the time-consuming enumeration process of standard prompts [39], [40].

III. Approach

In this section, we describe the proposed LogPPT approach. To overcome the limitations of existing approaches, we train a model to capture the patterns of templates and parameters based on the context information of log messages using rich knowledge derived from language models pre-trained on large corpora. Specifically, we apply the paradigm of prompt tuning [39] to enable few-shot log parsing to better transfer the knowledge from pre-trained language models to log parsing. To make the best use of prompt tuning, it is essential to select an optimized labelled training set for our method. Therefore, we introduce an Adaptive Random Sampling algorithm to effectively select a small number of samples for training.

The overview of the proposed approach is shown in Figure 3. In the following, we first present the problem formulation in Section III-A. Then, we describe the few-shot data sampling method in Section III-B. Section III-C describes the training process, which consists of three modules, including a pre-trained language model, a virtual label token generation module, and a training objective. Finally, we describe how to apply LogPPT for online parsing in Section III-D.

A. Problem Definition

In this work, we transform the log parsing task into a parameter recognition problem where only a small number of labelled examples are used for training by adopting a novel prompt tuning method [39]. Specifically, for a new dataset $\mathcal{D}$, we tune a pre-trained language model, $\mathcal{M}$, to recognise keywords and parameters in a log message through prompt tuning. The model takes the input of a raw log message consisting of n tokens, $X = \{x_1, x_2, \dots, x_n\}$ and predicts a virtual label token “`PARAM`” at the position of parameters. For keywords, the model remains to predict the original tokens. Formally, the model $\mathcal{M}$ is trained to generate the output, $Y = \{y_1, y_2, \dots, y_n\}$, where:

$$y_i = \mathcal{M}(x_i) = \begin{cases} \text{“}\texttt{PARAM}\text{”} & \text{if } x_i \text{ is a parameter} \\ x_i & \text{if } x_i \text{ is a keyword} \end{cases} \qquad (1)$$

For example, as shown in Figure 3, the model is trained to predict the parameter “`blgio91`” as a label token “`PARAM`”. For keywords such as “`failed`”, the model will predict the original words. “`PARAM`” is a specific virtual token that does not have any linguistic meaning. It indicates parameters in log messages and guides the model to recognise those parameters based on their relations with the “`PARAM`” token. The embedding vector of “`PARAM`” is calculated based on the most frequent

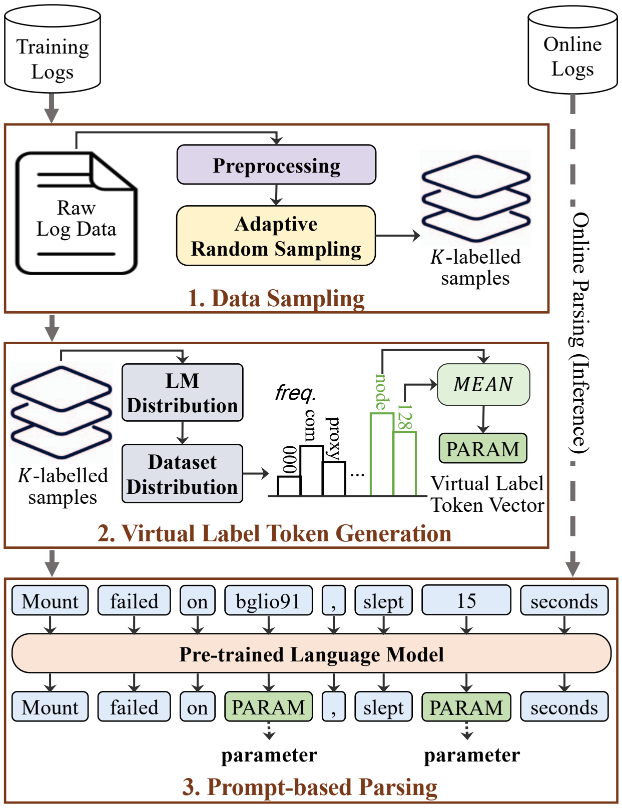


Fig. 3. An overview of LogPPT

parameters in log messages. “`PARAM`”, therefore, is generated using both labelled training data and unlabelled data to better represent the meaning of parameters in log messages. In the online parsing (inference) phase, all tokens with y_i = “`PARAM`” are considered parameters, and other tokens are included in the log template.

B. Few-shot Data Sampling

During the training phase, our proposed method requires a small amount of labelled log data as the training dataset. To collect accurately labelled samples with low manual effort, we propose a simple yet effective approach to select a small number (K) of labelled samples. Firstly, training log messages are cleaned by applying some commonly-used pre-processing techniques [2], [16], such as removing all non-character tokens, stop words or camel case. Then, we propose to use an Adaptive Random Sampling algorithm from Adaptive Random Testing [44] to obtain a diverse and evenly distributed sample set. Algorithm 1 describes the adaptive random sampling based algorithm for few-shot data selection.

Algorithm 1 takes a raw log dataset $\mathcal{D}$ and a desired number of samples in training set K. At line 1, all log messages in $\mathcal{D}$ are pre-processed by applying commonly-used pre-processing techniques [2], [16]. The result of this step is a set $\widehat{L} = \{\dots, (cln, orig), \dots\}$ in which each element contains a clean log message and an original log message. At lines

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Few-shot Data Sampling

```
    Data: $\mathcal{D}$: Log dataset
          $K$: The number of collected samples
    Result: $\mathcal{D}_{train}$: a set of $K$-labelled samples
 1  $\widehat{L} \leftarrow$ pre-process($\mathcal{D}$) // $\widehat{L}_i = \{cln, org\}$: clean and original logs
 2  $\mathcal{D}_{train} \leftarrow \emptyset$                              // initialize the training set
 3  $\mathcal{S} \leftarrow \{l \mid l \in \widehat{L}$ and $l.cln$ is the shortest cleaned log$\}$
 4  while $K > 1$ do
 5      $\widehat{C} \leftarrow \emptyset$                                  // initialize candidate set
 6      for $i = 1 \rightarrow \eta$ do                                    // $\eta = 32$
 7          $\widehat{C}$.add({random $c \in \widehat{L} | c.cln \notin \widehat{C}$ & $c.org \notin \mathcal{S}$})
 8      end
        /* compute the similarities between logs in $\widehat{C}$ and $\mathcal{S}$          */
 9      $\Delta \leftarrow \emptyset$
10      for $c = \{cln, org\} \in \widehat{C}$ do
11          $\delta \leftarrow 0$
            /* find the nearest neighbour of $c$ in $\mathcal{S}$ and calculate the
               similarity between $c$ and its nearest neighbour              */
12          foreach $l = \{cln, org\} \in \mathcal{S}$ do
13              $\delta$ = MAX($\delta$, similarity($c.cln, l.cln$))
14          end
15          $\Delta$.add($\delta$)
16      end
        /* select the candidate with the longest distance/smallest
           similarity to its nearest neighbour in $\mathcal{S}$                   */
17      $\mathcal{S}$.add({$c \in \widehat{C} \mid \Delta_c$ is smallest})
18      $K \leftarrow K - 1$
19  end
    /* label the sample set $\mathcal{S}$ of $K$ samples as the training set       */
20  foreach $s = \{cln, orig\} \in \mathcal{S}$ do
21      $\mathcal{D}_{train}$.add({$s.orig, template(s.orig)$})
22  end
23  return $\mathcal{D}_{train}$
```

2-3, the algorithm initializes the following two components: (1) an empty set $\mathcal{D}_{train}$, which is the result of the algorithm; (2) a set $\mathcal{S}$, which contains the shortest log message at first, to store selected log messages to label. At lines 4-19, the algorithm iteratively selects one log message per iteration based on their similarities until $\mathcal{S}$ contains K samples. From lines 5-8, η random candidate logs from $\widehat{L}$ are selected and stored in $\widehat{C}$. Then, for each candidate in $\widehat{C}$, the algorithm finds and calculates the similarity with its nearest neighbour in S (lines 9-16). At line 17, the algorithm finds a candidate c in $\widehat{C}$ which has the smallest similarity with its nearest neighbour (i.e., smallest Δ_c) and inserts it to the sample set $\mathcal{S}$. The outer loop repeats until $\mathcal{S}$ contains K elements. From lines 20-23, the algorithm collects the templates for all original log messages in $\mathcal{S}$ from user feedback and returns $\mathcal{D}_{train}$ as the final output.

C. Prompt-Tuning for Log Parsing

In this work, we take advantage of prompt-tuning, which recently set the state-of-the-arts for many NLP tasks, by applying the entity-oriented LM objective [39]. The essence behind this idea is that (1) most keywords in log statements are valid words and readable, which can be looked up in a dictionary [33], thus are easier to be predicted by the language model; and (2) parameters, in contrast, are constantly changing, which are hard to be predicted by the language model. In view of this, we transform the log parsing task into a label token prediction problem. Specifically, for parameters, we force the model to predict the virtual label token "PARAM", while for keywords, the model is trained to predict the original words.

1) Pre-trained Language Model: Pre-trained language models [22], [31], [45], [35] have been shown to be effective in many NLP tasks. These models are pre-trained on large-scale unlabelled corpus and then usually fine-tuned on downstream tasks. Recent studies [16], [33] demonstrate that these pre-trained models can be applied to understand the semantic meanings of log messages, thus favouring many downstream log analytics tasks. In this paper, we choose RoBERTa [22] as the studied pre-trained model since it is one of the most widely-used models. RoBERTa is an encoder-only model and uses the same transformer architecture as BERT [31]. Different from BERT, RoBERTa is trained to predict the mask token with a large byte-level Byte-Pair Encoding (BPE) [46]. One of the main reasons we choose RoBERTa over BERT is that the use of BPE allows RoBERTa to tokenize any input text without introducing any "unknown" tokens by tokenizing out-of-vocabulary words into subwords. This makes RoBERTa more suitable for log parsing because parameters created by developers are far beyond the scale of common English words and constantly changing, which would incur the out-of-vocabulary problem [19]. Several studies also found that RoBERTa is effective for log analysis [16], [33], [47].

2) Virtual Label Token Generation: Given an input sequence, $X = \{x_1, x_2, ..., x_n\}$, we adopt the template-free prompt tuning method [39] to predict a virtual label token "PARAM" at the position i via the pre-trained language model $\mathcal{M}$, where x_i is a parameter. Since all parameters are converted to the same token, it is essential to find a pivot token that can properly represent the parameters.

From the training set $\mathcal{D}_{train} = \{(X_i, Y_i)\}_{i=1}^{K}$, we leverage the pretrained language model $\mathcal{M}$ to get the probability distribution of predicting each token t at each position i. Specifically, we feed each sample (X, Y) into $\mathcal{M}$ and get the probability distribution $p(\widehat{x}_i = t|X)$ of predicting each token t in the log message X. Then, for each position i which is indicated as a parameter, we select the top$k$ predicted tokens of x_i as the initial parameters indication set $\mathcal{V}_{ini}$. This step aims to select top$k$ tokens having a similar meaning to the original parameter tokens to enrich the parameters indication set.

From the initial label-words set $\mathcal{V}_{ini}$, we simply search for the most frequent word in the unlabelled data. Specifically, we calculate the frequency $\phi(x = t|D)$ of each token $t \in \mathcal{V}_{ini}$ and select the most frequent words by ranking:

$$\mathcal{V} = \underset{t}{\text{argmax}}\ \phi(x = t|D), \forall t \in \mathcal{V}_{ini} \qquad (2)$$

After obtaining the set $\mathcal{V}$, we assign the embedding vector for the virtual label token "PARAM" by calculating the mean vector of all tokens in $\mathcal{V}$ and add it to the language model $\mathcal{M}$.

3) Training: Given the input log message $X = \{x_1, x_2, \ldots, x_n\}$, we construct a target sequence $Y = \{y_1, y_2, \ldots, y_n\}$ by replacing the parameter at the position

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

j with the virtual label token "PARAM", and maintaining the original words at keyword positions using Equation 1. Then, the LM model is trained to maximize the probability $P(Y|X)$ of the target sequence Y:

$$\mathcal{L} = -\frac{1}{K}\sum_{K}^{i=1}\left(\frac{1}{n}\sum_{j=1}^{n} logP(x_j = y_j|X_i)\right) \tag{3}$$

where K is the number of labelled training samples.

Note that we reuse the whole pre-trained model during the tuning process. The entity-oriented objective is similar to the LM-based (i.e., mask token prediction) objective, which can reduce the gap between pre-training and fine-tuning, thus allowing our model to keep the knowledge learned by the pre-trained LM model.

D. Online Parsing

During online parsing (inference), we directly feed the log messages into the trained model, which will first tokenize the input to a set of tokens and then predict their corresponding target tokens. If a token is predicted as "PARAM", it will be integrated into the parameter list; otherwise, it will be kept in the log template. Finally, we follow [18] to post-process log templates by replacing consecutive parameters with a single parameter. Note that we only need a one-pass decoding process to parse a log message, which is efficient when scaling to a large volume of logs.

IV. Experimental Design

A. Research Questions

We evaluate our approach by answering the following research questions (RQs):

RQ1: How effective is LogPPT?

RQ2: How efficient is LogPPT?

RQ3: How do different modules contribute to LogPPT?

RQ4: How does LogPPT perform with different tuning techniques?

B. Datasets

We conduct experiments based on datasets initially collected from the *LogPai* benchmark [11], [48], which consists of log data of 16 different systems, including distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software. To determine the ground truth log templates, Zhu et al. [11] randomly sampled 2,000 log messages for each dataset and manually labelled them. However, recent studies [18], [19] point out that there are multiple errors from these original datasets. Therefore, Khan et al. [18] applied some heuristic rules such as Double Space or User-defined String to fix incorrect templates in the original datasets. In this study, we use the corrected version of these 16 datasets from [18] in our evaluation.

C. Baselines

We compare our proposed method with five state-of-the-art methods, including AEL [17], LenMa [28], Spell [13], Drain [12], and Logram [14]. These approaches apply many techniques such as similarity-based clustering (i.e., LenMa), frequency-based mining (i.e., AEL and Logram), or heuristics-based searching (i.e., Drain and Spell). We choose these five approaches in our evaluation since they have their source code publicly available; and a prior study [11] finds that these approaches have the highest accuracy and efficiency among all the evaluated log parsers. We adopt the implementation of these methods from their replication packages [49], [50].

For a fair comparison, we extend baseline methods to include the labelled data from the data sampling phase. We transform the message-level labels into token-level labels by splitting log messages using default separators of each method.

D. Evaluation Metrics

Following recent studies [11], [18], [19], [20], we apply three metrics in our evaluation, including:

Group Accuracy (GA): Group Accuracy [11] is the most commonly used metric for log parsing. Group Accuracy considers template identification as a clustering process in which log messages with different log events are clustered into different groups [18]. The GA metric is defined as the ratio of "correctly parsed" log messages over the total number of log messages, where a log message is considered "correctly parsed" if and only if it is grouped with other log messages consistent with the ground truth. However, recent studies [18], [19] show that GA only accounts for how the parsed templates support the log message grouping activity instead of considering whether the templates and parameters are correctly identified or not.

Parsing Accuracy (PA): The Parsing Accuracy (or Message-Level Accuracy [19]) metric is defined as the ratio of "correctly parsed" log messages over the total number of log messages, where a log message is considered to be "correctly parsed" if and only if every token of the log message is correctly identified as template or variable. This metric is much stricter than Group Accuracy since any incorrectly parsed token will lead to the wrong parsing result for the whole log message. We found that this metric is useful when evaluating the performance of log parsers when dealing with unseen log events compared to Group Accuracy. For example, for those log events that only appear once, GA always considers them as correctly identified since they belong to the correct groups. In contrast, PA could mark this identification as incorrect if some variables are incorrectly recognised as keywords.

Edit Distance (ED): Edit Distance is proposed in [20]. Different from GA and PA, Edit Distance is used to evaluate the template extraction in terms of string comparison. Specifically, Edit Distance (or Levenshtein edit distance) is computed by counting the minimum number of operations required to transform one template into the other [20]. The score of Edit Distance for a dataset is computed as the median edit distance of all parsed template and ground truth template pairs. By computing the distance between parsed templates and ground

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

truth templates, this metric can measure the accuracy of log parsers in terms of meaning similarity (i.e., lexical similarity in our evaluation) between parsed results and ground truth. Note that the smaller the distance between two templates, the more similarity between them.

E. Implementation and Environment

We conduct our experiments on a GPU server equipped with NVIDIA Tesla V100 GPU and CUDA 10.2. We implement LogPPT with Python 3.8 and PyTorch 1.7. Followed recent studies for prompt tuning [36], [38], during the training process, we utilize AdamW [51] optimizer and set the initial learning rate to $5e^{-5}$. We set the batch size as 8 and train the model for 200 steps. AdamW optimizer is used with a linear decaying schedule with 10% warm-up steps. During the online parsing phase, we set the batch size to 32. In the Virtual Label Token Generation module, we calculate the embedding of the virtual label token "`PARAM`" from the 8 most frequent label tokens for our experiments. We also evaluate the performance of LogPPT with different numbers of frequent label tokens. We provide the results in our project webpage[1] due to space constraints. The results show that the performance of the proposed method is robust to the number of label tokens. It achieve consistently good results when choosing at least four label tokens. In the Few-shot Data Sampling module, we set $K = 32$ as the default. We also experiment with different values of K (from 4 to 128) in the experiments.

V. Experimental Results

A. RQ1: Parsing Effectiveness

1) Accuracy: In this RQ, we compare LogPPT with five state-of-the-art methods (including AEL [17], LenMa [28], Spell [13], Drain [12], and Logram [14]) on all 16 log datasets. Firstly, we compare the results of LogPPT with baselines using $K = 32$ labelled samples. The results in terms of three metrics (Group Accuracy, Parsing Accuracy, and Edit Distance) are shown in Table I.

From the results, we can see that our model outperforms baseline methods on almost all datasets in the three evaluation metrics. Specifically, in terms of Group Accuracy (GA), LogPPT exceeds the most powerful log parser (Drain) by 15.8% (0.923 versus 0.797 on average) and achieves the best results on 12 out of 16 datasets. It is worth noting that LogPPT achieves the accuracy of over 0.9 on 12 datasets and achieves 1.0 accuracy on four datasets among them, which is significantly superior to existing log parsers. In terms of Parsing Accuracy (PA), LogPPT surpasses baselines by at least 83.9% when achieving an accuracy of 0.916 on average. LogPPT also achieves the best parsing accuracy on 14 out of 16 datasets. The high Parsing Accuracy suggests that LogPPT is able to accurately recognise the templates and corresponding parameters of log messages. The experimental results confirm that LogPPT is effective in grouping logs into the same templates and identifying correct log templates and parameters.

Inspired by recent studies [18], [20], we also evaluate our proposed LogPPT in terms of Edit Distance (ED) to measure the similarity between identified templates and their corresponding ground truth. It can be seen that LogPPT achieves the best average edit distance of 1.130, which is 7 times better than Drain. Besides, LogPPT outperforms baseline approaches on 15 out of 16 datasets and achieves a comparable result on the Apache dataset (0.024 versus 0). The experimental results on Edit Distance show that the parsed templates produced by LogPPT have high textual similarities with the ground truth. The main reason for the high accuracy of LogPPT is that it is capable of learning from semantic information of log messages, thus is able to precisely identify the templates and parameters of log messages.

2) Robustness: Our proposed LogPPT explicitly aims at supporting a broad range of diverse log datasets as employing a general log parser in production environments requires a robust performance [11]. Existing log parsers are sensitive to pre-processing steps, which involve domain-specific knowledge. Therefore, they show low robustness against different logging formats and behaviours [11], [21]. Therefore, we next analyze and compare the robustness against different types of logs of LogPPT with that of the baselines. Figure 4 shows the accuracy distribution of each log parser across different log datasets.

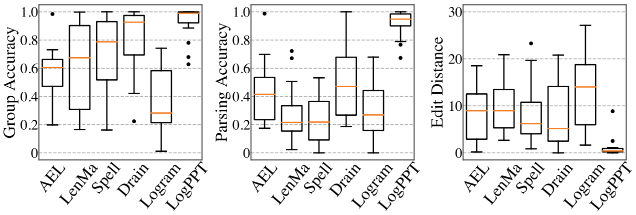


Fig. 4. Accuracy Distribution of Log Parsers with 32-shot

From the results, we can see that LogPPT outperforms the baselines in terms of robustness across different log types. Existing methods require different regular expressions for pre-processing and different hyper-parameter values, thus, performing inconsistently on different datasets. For example, Drain uses different *similarity threshold* (e.g., 0.2 for HealthApp and 0.6 for Proxifier) and different regular expressions (e.g., "`blk_-?\d+`" for HDFS and "`core.\d+`" for BGL) for different datasets. In contrast, LogPPT does not require to manually define regular expressions and achieves the smallest variance over different datasets. LogPPT is robust and performs well on most of the datasets (accuracy higher than 0.9) in terms of group and parsing accuracy. For example, LogPPT yields a median of 0.99 for GA robustness and 0.94 for PA robustness, which exceeds the second best log parser (i.e., Drain) by 6.9%, and 98.5%, respectively. Besides, LogPPT uses the same set of hyper-parameter values for every dataset in the training phase and does not require re-adjustment for each dataset. Overall, the experimental results confirm that LogPPT is robust and can be applied to different log datasets with low effort.

Our method requires a small amount (K) of labelled data

[1] https://github.com/LogIntelligence/LogPPT

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON WITH THE STATE-OF-THE-ART LOG PARSERS WITH 32-SHOT (↑: HIGHER IS BETTER; ↓: LOWER IS BETTER)

	AEL			**LenMa**			**Spell**			**Drain**			**Logram**			**LogPPT**		
	GA (↑)	PA (↑)	ED (↓)	GA (↑)	PA (↑)	ED (↓)	GA (↑)	PA (↑)	ED (↓)	GA (↑)	PA (↑)	ED (↓)	GA (↑)	PA (↑)	ED (↓)	GA (↑)	PA (↑)	ED (↓)
HDFS	0.626	0.630	0.926	0.998	0.125	3.949	**1**	0.487	0.858	0.998	**0.959**	0.452	0.012	0.018	18.665	**1**	0.902	**0.276**
Hadoop	0.677	0.422	12.163	0.667	0.242	16.788	0.533	0.196	9.274	0.948	0.439	7.564	0.283	0.370	19.014	**0.994**	**0.895**	**0.882**
Spark	0.415	0.381	3.088	0.869	0.023	10.130	0.920	0.336	4.192	0.905	0.376	2.568	0.282	0.275	7.433	**0.999**	**0.991**	**0.167**
Zookeeper	0.657	0.527	2.430	0.894	0.457	4.472	0.987	0.453	2.450	0.967	0.498	2.304	0.724	0.516	3.928	**0.994**	**0.990**	**0.338**
BGL	0.491	0.410	4.288	0.316	0.154	7.929	0.850	0.329	5.952	0.955	0.444	3.958	0.218	0.170	8.954	0.954	**0.970**	**0.233**
HPC	0.731	0.698	1.151	0.681	0.671	2.687	0.657	0.532	3.633	0.741	0.672	1.845	0.742	0.679	2.628	**0.943**	**0.947**	**1.147**
Thunderbird	0.650	0.203	13.640	0.943	0.171	7.924	0.856	0.039	12.280	**0.960**	0.191	13.675	0.129	0.128	15.479	0.679	**0.926**	**0.857**
Windows	0.685	0.389	10.475	0.287	0.266	19.132	0.990	0.004	2.961	**0.994**	0.696	4.705	0.694	0.374	6.413	0.991	**0.983**	**0.461**
Linux	0.404	0.239	15.200	0.238	0.132	12.631	0.162	0.109	16.069	0.422	0.194	15.438	0.201	0.185	16.514	**0.934**	**0.949**	**0.279**
Android	0.642	0.559	8.082	0.778	0.722	5.602	0.891	0.241	8.311	0.765	0.730	5.626	0.677	0.428	12.872	**0.885**	**0.767**	**1.143**
HealthApp	0.570	0.175	18.474	0.166	0.289	15.947	0.961	0.152	5.119	0.644	0.241	18.393	0.258	0.263	15.173	**1**	**0.789**	**2.536**
Apache	0.984	0.987	0.189	0.984	0.293	3.524	0.301	0.285	10.275	**1**	**1**	**0**	0.297	0.509	1.658	**1**	0.994	0.024
Proxifier	0.495	0.506	9.980	0.495	0.506	9.168	0.527	0.478	6.457	0.527	0.527	9.982	0.016	0	27.118	**1**	**1**	**0**
OpenSSH	0.198	0.421	4.193	0.927	0.155	8.744	0.488	0.127	5.888	**0.996**	0.534	3.539	0.343	0.482	4.654	0.628	**0.976**	**0.119**
OpenStack	0.266	0.187	9.822	0.213	0.191	11.199	0.245	0	19.663	0.224	0.187	20.801	0.241	0.112	49.110	**0.989**	**0.907**	**0.788**
Mac	0.583	0.223	18.523	0.648	0.155	20.867	0.724	0.033	23.281	0.711	0.277	20.531	0.551	0.252	21.651	**0.780**	**0.673**	**8.856**
Average	0.567	0.435	8.289	0.631	0.284	10.043	0.693	0.237	8.541	0.797	0.498	8.211	0.354	0.297	14.454	**0.923**	**0.916**	**1.130**

sampled by an adaptive random sampling algorithm as the training set. Therefore, to evaluate the sensitivity of our proposed LogPPT to the amount of labelled data, we conduct an experiment using different numbers of training log messages (i.e., different *shots*). Figure 5 shows the performance of LogPPT with different numbers of shots.

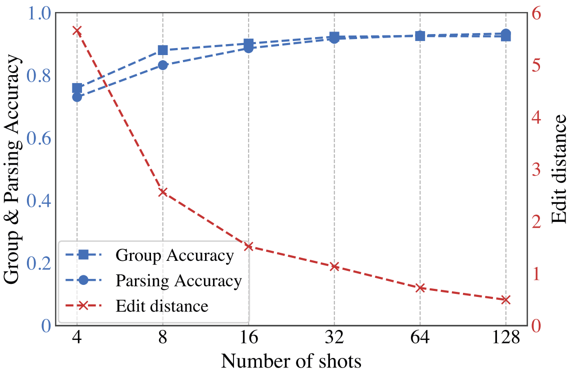


Fig. 5. Results of LogPPT with different shots (K)

The experimental results show that the model's performance witnesses a severe drop when less data is used for training. The low results are reasonable since pre-trained models require task-specific data for better adapting to downstream tasks [36]. However, we observe that LogPPT achieves a good balance between Group Accuracy and Parsing Accuracy. Also, LogPPT performs better than baselines in terms of Parsing Accuracy and Edit Distance even with only four labelled training samples. Moreover, it is noticeable that LogPPT can consistently achieve good results when $K \geq 16$.

In summary, LogPPT significantly outperforms the existing approaches in all three evaluation metrics. The experimental results confirm that LogPPT is capable of recognising log templates and the corresponding parameters.

3) Accuracy with Unseen Logs: Unseen log events occur frequently in logs. In this study, we consider the log events appearing only once in a dataset as previously unseen log events. LogPPT can accurately recognise the templates and corresponding parameters of unseen log events, as reflected by the high Parsing Accuracy. To further evaluate the ability of LogPPT in parsing unseen logs, we measure the Parsing Accuracy of LogPPT on unseen log data and compare it with baseline methods. Specifically, for every dataset, we extract those log messages whose corresponding log templates only appear one time based on the ground truth, then calculate the Parsing Accuracy on these log messages. Table II shows the results. There are 42.64 unseen log events on average on 16 studied datasets. LogPPT achieves the best accuracy of 0.599 when parsing unseen log data, which exceeds existing log parsers by 58.9% (LenMa) to 517.5% (Logram).

TABLE II
PARSING ACCURACY ON UNSEEN LOG DATA

	#Unseen	AEL	LenMa	Spell	Drain	Logram	LogPPT
Parsing Acc.	42.64	0.335	0.377	0.230	0.372	0.097	**0.599**

B. RQ2: Runtime Performance Evaluation

Besides effectiveness, efficiency is another critical metric for log parsers to consider in order to handle large-scale log data. To measure the efficiency of our proposed LogPPT, we record the running time it needs to finish the entire parsing process and compare it with the baseline methods. Specifically, we conduct this experiment on BGL and HDFS datasets, as they are relatively large. Figure 6 reports the results.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

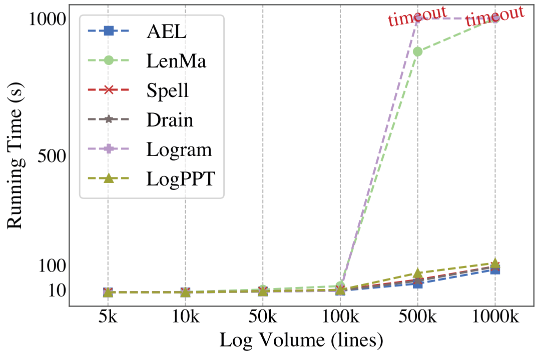


Fig. 6. Running time of different log parsers under different volume

We can see that the running time of LogPPT increases slowly with the increase of log data volume. With the use of GPU acceleration, our model can perform faster than or comparable with traditional log parsers. For example, LogPPT takes about 107 seconds to process one million log messages, which is just slightly slower than Drain (94s), Spell (95s) and AEL (84s) and much faster than LenMa and Logram (cannot finish within 1,000 seconds).

C. RQ3: Ablation Study

In this section, we evaluate the effectiveness of the major components and parameters in our proposed model. Specifically, we exclude the Virtual Label Token Generation module and let the pre-trained model automatically assign the embedding for the virtual label token "`PARAM`". To measure the contribution of the Adaptive Random Sampling module, we remove it from our model and randomly sample the log messages for labelling. We repeat this random process five times to avoid random bias and report the average results in Table III.

TABLE III
ABLATION STUDY RESULTS

	GA	**PA**	**ED**
Full LogPPT	0.923	0.916	1.130
$\text{w/o}_{\text{Virtual Label Token Gen.}}$	$0.879_{(\downarrow 5.8\%)}$	$0.835_{(\downarrow 8.8\%)}$	$3.130_{(\downarrow 177\%)}$
$\text{w/o}_{\text{Adaptive Random Sampling}}$	$0.890_{(\downarrow 3.6\%)}$	$0.704_{(\downarrow 23.1\%)}$	$3.602_{(\downarrow 219\%)}$

We can see that LogPPT performs worse in terms of parsing accuracy and edit distance without Virtual Label Token Generation and Adaptive Random Sampling modules. For example, without the Virtual Label Token Generation module, LogPPT only achieves a parsing accuracy of 0.835, which is 8.8% worse than complete LogPPT, while it can still achieve an acceptable group accuracy (0.879). The reason is that without the Virtual Label Token Generation module, the model cannot find the pivot word that can mostly represent parameters in log messages. Consequently, many parameters are misidentified, leading to a worse parsing accuracy and edit distance. On the other hand, log messages are highly imbalanced under different log templates. Using a naive random sampling technique cannot guarantee the quality of the training set. Therefore, the results significantly decline when we remove the Adaptive Random Sampling module (23.1% decreasing in terms of Parsing Accuracy).

In summary, this comparison demonstrates the usefulness of the proposed Adaptive Random Sampling module and the Virtual Label Token Generation module of LogPPT.

D. RQ4: Comparison with Different Tuning Techniques

LogPPT applies a novel prompt tuning method (i.e., template-free prompt [39]), which relaxes the burden of manually selecting prompt templates and improves the efficiency compared to other prompt tuning methods. In this section, we evaluate the performance of this prompt tuning method. To this end, we replace our prompt tuning module with four different prompt tuning methods (introduced in Section II-B) and a fine-tuning technique. We then compare the performance of LogPPT with that of the variants.

- **FT** (fine-tuning): We add a binary classification layer on top of the pre-trained RoBERTa model and fine-tune the model to perform log parsing as a binary token classification problem.
- **HardPT$_\mathbf{M}$** (hard prompt tuning with manual label words): We use a standard hard prompt [52] with the prompt template of "[X] [S] is a [MASK]", where [X], [S], and [MASK] are the unfilled slots for the input log message, token, and label respectively. The model learns to predict the label word at the [MASK] position. In this setting, we use fixed manual sets of label words, including "*[const, keyword]*" for keyword tokens and "*[variable, parameter]*" for parameter tokens.
- **HardPT$_\mathbf{S}$** (hard prompt tuning with soft label words): We use the same standard hard prompt template as the above setting. However, we use trainable tokens [53] as the label words for this setting.
- **SoftPT$_\mathbf{M}$** (soft prompt tuning with manual label words): In this setting, we follow recent works to use a soft prompt-template of "[X] [S] [SOFT] [SOFT] [SOFT] [MASK]", where [X], [S], and [MASK] are the unfilled slots for the input log message, token, and label respectively. [SOFT] is a trainable token. The embeddings of these [SOFT] tokens are optimized during the tuning stage. We use manual label word sets for this setting as in **HardPT$_\mathbf{M}$**.
- **SoftPT$_\mathbf{S}$** (soft prompt tuning with soft label words): We use the same soft prompt template of "[X] [S] [SOFT] [SOFT] [SOFT] [MASK]" as in the **SoftPT$_\mathbf{M}$** setting. For label words, we adopt the same **HardPT$_\mathbf{M}$** setting to use trainable tokens [53] as the label words.

Table IV shows the results. We can see that LogPPT with our proposed prompt tuning method achieves the best results among all studied methods. For example, with *16shot* setting, LogPPT outperforms others by 6.0%-74.5% in terms of Parsing Accuracy. Our proposed method significantly outperforms other prompt tuning methods because it can leverage both semantic and position information of tokens in log messages. Standard prompt tuning methods overly focus on leveraging the semantic

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
COMPARISON WITH DIFFERENT TUNING METHODS

	4shot			8shot			16shot			32shot		
	GA	PA	ED	GA	PA	ED	GA	PA	ED	GA	PA	ED
FT	0.69	0.70	5.98	0.88	0.74	3.56	0.85	0.84	2.09	0.91	0.91	1.23
HardPT$_{\mathrm{M}}$	0.54	0.54	6.75	0.64	0.62	4.12	0.69	0.64	3.57	0.68	0.68	2.71
HardPT$_{\mathrm{S}}$	0.48	0.44	10.08	0.57	0.51	8.83	0.54	0.51	8.47	0.67	0.67	3.27
SoftPT$_{\mathrm{M}}$	0.56	0.52	6.63	0.54	0.61	5.05	0.54	0.55	8.26	0.66	0.65	5.38
SoftPT$_{\mathrm{S}}$	0.29	0.24	20.34	42	0.46	9.40	0.46	0.48	7.30	0.58	0.64	5.94
LogPPT	**0.76**	**0.73**	**5.66**	**0.88**	**0.83**	**2.56**	**0.90**	**0.89**	**1.51**	**0.92**	**0.92**	**1.13**

meaning of a token and overlook the contextual information which is important in log parsing. Fine-tuning, on the other hand, can achieve better results than standard prompt tuning because it can use the positional information during the training stage. With more labelled training data, fine-tuning can achieve quite similar results with LogPPT.

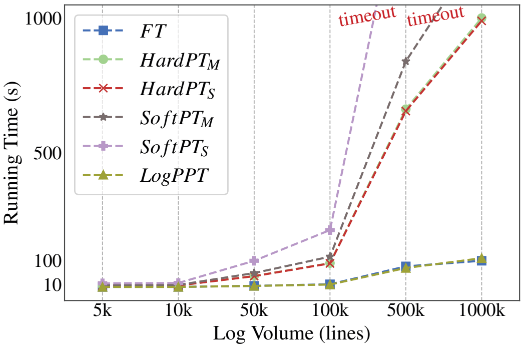


Fig. 7. Parsing time of different tuning methods

Next, we evaluate the parsing time of different tuning methods. As shown in Figure 7, the parsing time of LogPPT and fine-tuning approach is similar because they only need one-pass decoding to parse one log message. On the other hand, other prompt tuning methods need to enumerate all tokens in a log message which is a time-consuming process. For example, with soft prompts, the model cannot finish parsing one million log lines within 1,000 seconds.

In summary, our proposed method is more effective and efficient compared to other tuning techniques and can achieve high accuracy with a few shots of training data.

VI. Discussion

A. Why does LogPPT Work?

There are several reasons that make LogPPT perform better than the related approaches. First, LogPPT predicts keywords and parameters using the semantic information from log messages by tuning a pre-trained language model. Thus, compared to traditional methods using only superficial features, LogPPT is able to indicate the keywords or parameters more precisely. Besides, LogPPT does not require domain-specific knowledge to define regular expressions for each dataset, thus is easy to be applied to a new log dataset.

Second, compared to other few-shot learning techniques, LogPPT applies an effective and efficient prompt tuning method, which can avoid the complex design for prompt instructions and also boost the few-shot performance. LogPPT leverages both semantic and positional information of tokens in log messages, thus can handle the noise in log data compared to other prompt tuning methods. For example, the log message from Proxifier, "`open through` **`proxy proxy`**`.cse.cuhk.edu.hk:5070`", contains two "`proxy`" tokens with different roles. Standard prompt tuning methods fail to distinguish these tokens and predict the same label for them. The reason is that standard prompt tuning methods only consider the semantic meaning of tokens but ignore the position information, which is important for log parsing. In contrast, our method utilizes both semantic and position information of a token in log messages and achieves high parsing accuracy (100% parsing accuracy).

B. Threats to Validity

We have identified the following major threats to validity.

Data Quality. In this paper, we used public log datasets for our evaluation. The ground truth templates of all log messages, including log templates and corresponding parameters, are provided within the datasets. Although these datasets are commonly used by many related works [11], [14], [20], they may also contain a small proportion of errors. To reduce this threat, we leverage the latest version of the benchmark datasets [18] that are corrected with automatic and manually-defined rules.

Tool Comparison. In our evaluation, we compared our results with related approaches. The approaches achieved the best results in a recent benchmark [11] and are used in both industry and academia. We adopt the implementations from their replication packages. We apply the parameters and settings (e.g., number of log templates, similarity threshold, etc.) optimized by the previous work [11].

Labelling Effort. Our proposed method relies on a small number of labelled log data. To reduce the labelling effort, we propose to use an Adaptive Random Sampling algorithm to select a diverse set of K log messages (K from 4 to 128) and attain the templates from user feedbacks.

VII. Related Work

Log Analysis with Language Models: Log analysis is a research area that has attracted lots of attention due to its practical importance. Typical applications of log analysis include anomaly detection [1], [54], [55], [56], failure prediction [7], [8], root cause analysis [5], [6], etc. Recently, inspired by the success of pre-trained models in NLP, many studies have been proposed to apply pre-trained language models to log analysis. SwissLog [33] and NeuralLog [16] utilize the pre-trained BERT [31] model for log-based anomaly detection. Ott et al. [57] studied the use of different pre-trained models such as BERT [31] and XLNet [58] for log anomaly detection. Setianto et al. [59] proposed to fine-tune the GPT2 [45] model for log parsing.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

Data-driven Log Parsing: Log parsing has become an active research topic in recent years [12], [13], [20], [60]. Recently, to address the limitations of traditional log parsers and improve the parsing accuracy, some approaches [34], [19] proposed to use token classification for log parsing. LogStamp [34] converts the log parsing task into a sequence labelling problem. It leverages the BERT [31] model to classify words in log messages. These approaches, however, adopt a traditional log parser to generate pseudo labels for log messages as the training data, which can introduce many noises in training data. Liu et al. [19] proposed UniParser, which is a unified log parser for heterogeneous log data. UniParser is trained with labelled data across multiple log sources to capture the common patterns of templates and parameters. Although effective, UniParser requires a noticeable amount of labelled data to train a classification model, which is not always available in practice. Besides, UniParser requires handcrafted rules to split raw log messages into tokens, which is not suitable to apply on some special dataset [19].

Our LogPPT can effectively leverage semantic information from a few labelled data by using a pre-trained language model. LogPPT does not require any domain-specific knowledge to pre-process log data, thus can adapt to new log dataset with low effort. Besides, by using a novel prompt tuning method, LogPPT can effectively learn the semantic patterns from a few labelled data.

VIII. Conclusion

Log parsing is the foundation step to enabling automated log analytics. To overcome the limitations of existing log parsers, we propose a log parser with prompt-based few-shot learning, namely LogPPT, to capture the patterns of templates and parameters. LogPPT utilises a novel prompt tuning method to recognise keywords and parameters from a few labelled log data selected by an adaptive random sampling algorithm. We have evaluated LogPPT on public log datasets. The results show that LogPPT is effective and efficient, outperforming the state-of-the-art log parsers. In the future, we will deploy LogPPT in a production environment to further evaluate its scalability and effectiveness in practice.

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

Acknowledgment

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

References

[1] M. Du, F. Li, G. Zheng, and V. Srikumar, “Deeplog: Anomaly detection and diagnosis from system logs through deep learning,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[2] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, “Robust log-based anomaly detection on unstable log data,” in *ESEC/FSE 2019*, 2019, pp. 807–817.

[3] B. Zhang, H. Zhang, P. Moscato, and A. Zhang, “Anomaly detection via mining numerical workflow relations from logs,” in *2020 International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2020, pp. 195–204.

[4] B. Zhang, H. Zhang, V.-H. Le, P. Moscato, and A. Zhang, “Semi-supervised and unsupervised anomaly detection by mining numerical workflow relations from system logs,” *Automated Software Engineering*, vol. 30, no. 1, p. 4, 2023.

[5] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, “Log-based abnormal task detection and root cause analysis for spark,” in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 389–396.

[6] N. Gurumdimma, A. Jhumka, M. Liakata, E. Chuah, and J. Browne, “Crude: combining resource usage data and error logs for accurate error detection in large-scale distributed systems,” in *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2016, pp. 51–60.

[7] A. Das, F. Mueller, C. Siegel, and A. Vishnu, “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*, 2018, pp. 40–51.

[8] S. Zhang, Y. Liu, W. Meng, Z. Luo, J. Bu, S. Yang, P. Liang, D. Pei, J. Xu, Y. Zhang *et al.*, “Prefix: Switch failure prediction in datacenter networks,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, pp. 1–29, 2018.

[9] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, “Logzip: extracting hidden structures via iterative clustering for log compression,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 863–873.

[10] J. Wei, G. Zhang, Y. Wang, Z. Liu, Z. Zhu, J. Chen, T. Sun, and Q. Zhou, “On the feasibility of parser-based log compression in large-scale cloud systems,” in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 249–262.

[11] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, “Tools and benchmarks for automated log parsing,” in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2019, pp. 121–130.

[12] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An online log parsing approach with fixed depth tree,” in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 33–40.

[13] M. Du and F. Li, “Spell: Streaming parsing of system event logs,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[14] H. Dai, H. Li, C. S. Chen, W. Shang, and T.-H. Chen, “Logram: Efficient log parsing using n-gram dictionaries,” *IEEE Transactions on Software Engineering*, 2020.

[15] M. Nagappan and M. A. Vouk, “Abstracting log lines to log event types for mining software system logs,” in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. IEEE, 2010, pp. 114–117.

[16] V.-H. Le and H. Zhang, “Log-based anomaly detection without log parsing,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 492–504.

[17] Z. M. Jiang, A. E. Hassan, P. Flora, and G. Hamann, “Abstracting execution logs to execution events for enterprise applications (short paper),” in *2008 The Eighth International Conference on Quality Software*. IEEE, 2008, pp. 181–186.

[18] 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 44th International Conference on Software Engineering (ICSE’22)*. ACM, 2022.

[19] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang *et al.*, “Uniparser: A unified log parser for heterogeneous log data,” in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1893–1901.

[20] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, “Self-supervised log parsing,” in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2020, pp. 122–138.

[21] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, “An evaluation study on log parsing and its use in log mining,” in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2016, pp. 654–661.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.

[22] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[23] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009, pp. 117–132.

[24] M. Nagappan, K. Wu, and M. A. Vouk, "Efficiently extracting operational profiles from execution logs using suffix arrays," in *2009 20th International Symposium on Software Reliability Engineering*. IEEE, 2009, pp. 41–50.

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

[26] Q. Fu, J.-G. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 149–158.

[27] L. Tang, T. Li, and C.-S. Perng, "Logsig: Generating system events from raw textual logs," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, 2011, pp. 785–794.

[28] K. Shima, "Length matters: Clustering system log messages using length of words," *arXiv preprint arXiv:1611.03213*, 2016.

[29] "HDFS dataset," 2022. [Online]. Available: https://github.com/logpai/loghub/tree/master/HDFS

[30] "BGL dataset," 2022. [Online]. Available: https://github.com/logpai/loghub/tree/master/BGL

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "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)*, 2019, pp. 4171–4186.

[32] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[33] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust and unified deep learning based log anomaly detection for diverse faults," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 92–103.

[34] S. Tao, W. Meng, Y. Cheng, Y. Zhu, Y. Liu, C. Du, T. Han, Y. Zhao, X. Wang, and H. Yang, "Logstamp: Automatic online log parsing based on sequence labelling," *ACM SIGMETRICS Performance Evaluation Review*, vol. 49, no. 4, pp. 93–98, 2022.

[35] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," 2018.

[36] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 382–394.

[37] X. Han, W. Zhao, N. Ding, Z. Liu, and M. Sun, "Ptr: Prompt tuning with rules for text classification," *AI Open*, vol. 3, pp. 182–192, 2022.

[38] T. Gao, A. Fisch, and D. Chen, "Making pre-trained language models better few-shot learners," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 3816–3830.

[39] R. Ma, X. Zhou, T. Gui, Y. Tan, L. Li, Q. Zhang, and X. Huang, "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, United States: Association for Computational Linguistics, Jul. 2022, pp. 5721–5732. [Online]. Available: https://aclanthology.org/2022.naacl-main.420

[40] L. Wang, R. Li, Y. Yan, Y. Yan, S. Wang, W. Wu, and W. Xu, "Instructionner: A multi-task instruction-based generative framework for few-shot ner," *arXiv preprint arXiv:2203.03903*, 2022.

[41] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–35, 2023.

[42] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 4582–4597.

[43] G. Qin and J. Eisner, "Learning how to ask: Querying LMs with mixtures of soft prompts," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Online: Association for Computational Linguistics, Jun. 2021, pp. 5203–5212. [Online]. Available: https://aclanthology.org/2021.naacl-main.410

[44] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Annual Asian Computing Science Conference*. Springer, 2004, pp. 320–329.

[45] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[46] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016, pp. 1715–1725.

[47] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via bert," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[48] "A large collection of system log datasets for ai-powered log analytics," 2021. [Online]. Available: https://github.com/logpai/loghub

[49] "A toolkit for automated log parsing," 2022. [Online]. Available: https://github.com/logpai/logparser

[50] "Artifact for ”guidelines for assessing the accuracy of log message template identification techniques”," 2022. [Online]. Available: https://doi.org/10.6084/m9.figshare.18858332

[51] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[52] L. Cui, Y. Wu, J. Liu, S. Yang, and Y. Zhang, "Template-based named entity recognition using bart," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 2021, pp. 1835–1845.

[53] K. Hambardzumyan, H. Khachatrian, and J. May, "Warp: Word-level adversarial reprogramming," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021, pp. 4921–4933.

[54] S. He, J. Zhu, P. He, and M. R. Lyu, "Experience report: System log analysis for anomaly detection," in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 207–218.

[55] X. Li, P. Chen, L. Jing, Z. He, and G. Yu, "Swisslog: Robust anomaly detection and localization for interleaved unstructured logs," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[56] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1356–1367.

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

[58] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *Advances in neural information processing systems*, vol. 32, 2019.

[59] F. Setianto, E. Tsani, F. Sadiq, G. Domalis, D. Tsakalidis, and P. Kostakos, "Gpt-2c: a parser for honeypot logs using large pre-trained language models," in *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2021, pp. 649–653.

[60] W. Ahmad, S. Chakraborty, B. Ray, and K.-W. Chang, "Unified pre-training for program understanding and generation," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021, pp. 2655–2668.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on March 27,2024 at 17:52:17 UTC from IEEE Xplore. Restrictions apply.