PILAR: Studying and Mitigating the Influence of Configurations on Log Parsing

Hetong Dai*, Yiming Tang*, Heng Li†, Weiyi Shang*
*Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
†Polytechnique Montréal, Montreal, Canada
Email: *{he_da, t_yiming, shang}@encs.concordia.ca, †heng.li@polymtl.ca

***Abstract*—The significance of logs has been widely acknowledged with the adoption of various log analysis techniques that assist in software engineering tasks. Many log analysis techniques require structured logs as input while raw logs are typically unstructured. Automated log parsing is proposed to convert unstructured raw logs into structured log templates. Some log parsers achieve promising accuracy, yet they rely on significant efforts from the users to tune the parameters to achieve optimal results. In this paper, we first conduct an empirical study to understand the influence of the configurable parameters of six state-of-the-art log parsers on their parsing results on three aspects: 1) varying the parameters while using the same dataset, 2) keeping the same parameters while using different datasets, and 3) using different samples from the same dataset. Our results show that all these parsers are sensitive to the parameters, posing challenges to their adoption in practice. To mitigate such challenges, we propose *PILAR* (Parameter Insensitive Log Parser), an entropy-based log parsing approach. We compare *PILAR* with the existing log parsers on the same three aspects and find that *PILAR* is the most parameter-insensitive one. In addition, *PILAR* achieves the second highest parsing accuracy and efficiency among all the state-of-the-art log parsers. This paper paves the road for easing the adoption of log analysis in software engineer practices.**

I. Introduction

Logging is widely used, and sometimes the only source, to record run-time information for modern software systems [1]. Logs are produced by logging statements within the source code. As illustrated in Table I, which is an example of a real-world project *ZooKeeper*, a typical logging statement is composed of a logging object (`LOG`), a logging level (`warn`), developer-defined text (the quoted text) and dynamic information generators (e.g. variable or method invocation) [2]–[4].

TABLE I
An example logging statement extracted by LogPai [5] from ZooKeeper and its run-time log instance

Logging statement	`LOG.warn("Connection request from old client {}; will be dropped if server is in r-o mode", cnxn.getRemoteSocketAddress());`
Log instance	2015-07-30 16:12:01,554 - WARN [NIOServerCxn.Factory:0.0.0.0/0.0.0.0:2181:ZooKeeperServer@793] - Connection request from old client /10.10.34.19:33442; will be dropped if server is in r-o mode
Log template	Connection request from old client <*>; will be dropped if server is in r-o mode

The significance of logging has been increasingly acknowledged in recent years and logs are used for a variety of software engineering tasks, such as bug detection [1], [6]–[9], system behavior understanding [3], [10]–[12], as well as performance observation and improvement [13]–[15]. Most of these log analysis techniques require structured logs as input. However, as we can see in Table I, raw logs generated by logging statements are usually lines of unstructured text with flexible format which are difficult to be directly leveraged for sophisticated analysis. Thus, log parsing is a very common first step of the majority of log analysis. Automated log parsers, such as Drain [16], AEL [17] and Logram [18] have been proposed by prior research. The key goal of these parsers is to identify the dynamic part and static parts of raw logs and to generate log templates (a log template is shown in Table I) [5], [18]. In general, the static parts refer to the developer-defined text and the dynamic part refer to the dynamic information generated by method invocations or variables at runtime.

However, the configuration of automated log parsers' parameters plays a critical role in generating accurate parsing result. For example, Drain [16], a log parser that provides almost the top accuracy [5], requires developers to choose 2 configuration parameters for different datasets. However, with sub-optimal configuration parameter values, Drain's parsing results may degrade to one of the worst (cf. Section III). In this paper, we first study the influence of configuration parameter values in six state-of-the-art log parsers on 16 datasets from the LogPai benchmark [5]. These six log parsers have a total of ten parameters with 79 possible values to tune. We find that the choice of configuration parameter values, the dataset under analysis and even the different samples of the same dataset may lead to significant differences in the accuracy of parsing results.

Understanding the challenges of tuning log parsers' configuration parameters, we propose *PILAR* (**P**arameter **I**nsensitive **L**og P**ar**ser), an entropy-based log parser which can parse logs accurately while being insensitive to the variations in parameters. The main idea of *PILAR* is that **the dynamic parts of raw logs that are generated by variables or method invocations in the source code tend to have a higher entropy compared to the static parts of raw logs that are written in static text by developers**. In other words, the static parts are more predictable (thus lower entropy) as they are written by developers, while the dynamic parts are less predictable (thus higher entropy) as they are produced by software system in running time. Therefore, an entropy based probability is used to distinguish dynamic components from static components of logs. Besides, as the static parts of the logs are natural language text written by developers, their entropy is likely to be stable [19]. Thus the entropy based probability threshold used to distinguish dynamic and static components is relatively stable across different log datasets The evaluation of *PILAR* shows that *PILAR* is the most parameter-insensitive log parsing approach among all the studied log parsers. Moreover, the *PILAR* itself has a high accuracy and efficiency (both second best among all parsers). In other words, practitioners do not need to spend effort on fine tuning *PILAR* to achieve almost the most accurate and most efficient log parsing[1]. In summary, our paper has the following three main contributions:

- We conduct a comprehensive empirical study to investigate the influence of configuration parameters on log parsers.
- We propose a novel entropy-based log parsing approach *PILAR* that is parameter insensitive while achieving high accuracy and efficiency compared to state-of-the-art log parsers.
- We publish *PILAR* as an off-the-shelf log parsing tool which allows practitioners to perform log parsing in their daily log analysis tasks without considering the configuration.

Paper organization The paper is organized as follows. Section II introduces the background and prior research related to log parsing. Section III conducts an empirical study on the influence of configurations of log parsers. Section IV proposes the approach of *PILAR*. Section V conducts an evaluation of *PILAR*. Section VI discusses the threats to the validity and Section VII concludes our paper.

II. Background and Related Work

In this section, we present the background and related work of this paper.

A. Log parsing

Log parsing has been widely studied in recent years. The development of log parsing approaches can be divided into three phases: 1) heuristic rule based log parsing, 2) source code based log parsing and 3) data driven log parsing.

However, heuristic rules based log parsing need developers to maintain these rules as the logging statements evolve, which needs great efforts from developers. Though source code based approach can achieve a high accuracy, source code is often inaccessible in many practical log analysis scenarios.

To address the challenges faced by the first two types of log parsing approaches, data driven log parsing approaches are proposed. Drain [16], Logram [18] and Spell [20] are the representative of this type of log parsing approaches. These approaches utilize data mining techniques to extract log templates from raw logs or cluster the logs into different groups. For example, Drain [16] utilizes a fixed depth parse tree to encode specially designed rules for log parsing. Logram [18] utilizes n-gram dictionaries and relies on the frequencies of the n-grams to parse logs. Spell [20] utilizes a longest common subsequence based approach to parse logs. Data driven log parsing approaches only need the raw logs as input. This type of log parsing approaches can avoid the effort for maintaining ad hoc rules and the need for source code.

Log parsing is usually a prerequisite of various log analysis tasks, such as anomaly detection [21]–[25], failure diagnosis [10], [26] and system comprehension [10], [27]. Prior work [28] also shows that log parsing result is critical to the success of log analysis tasks. The keystone-role of accurate log parsing results motivate us to study the influence of configurations in automated log parsers.

B. Configurations of log parsers

Most automated log parsing approaches rely on configurable parameters to optimize the parsing performance for specific log datasets. For example, Drain [16] requires specifying the depth of the parse tree and a similarity threshold. Spell [20] uses a threshold for the minimum similarity of the log instances (in the paper, we also use the term "log line" to mention logs if we are concerned with lines) of a template before conducting log parsing process. Logram [18] requires bi-gram and tri-gram frequency thresholds to determine whether there exists a dynamic token inside a 2-gram or a 3-gram.

In order to achieve a (near-)optimal log parsing result, one often needs to carefully choose the configuration parameter values. We examine the choices of the configuration parameter values of the five most accurate log parsers in the LogPai benchmark [5] and a recently published log parser that is not included in the benchmark, i.e., Logram [18]. Table II shows the parameter values used for the different log parsers in the LogPai benchmark and Logram. We find that most log parsers have more than one configuration parameter. Each configuration parameter can be specified with many possible values. For example, the threshold of LenMa varies from 1 to 0.5 and there are 10 different unique values chosen for different log datasets in the LogPai benchmark [5]. In addition, the choices of configuration parameter values may have a large difference between different log datasets. For example, the *Bi_threshold* (the bi-gram frequency threshold) of Logram varies from 115 to 8. Besides, the threshold can grow with the size of the log dataset, as the frequency of the n-grams increases along with the log size. Such a large number and large difference of configuration parameter values illustrate the importance and challenges of choosing the right ones in practice. It is needed to mention that although Logram [18] proposes an automated method to generate configuration parameters, this method can only provide an approximate value for the parameter which cannot provide the best parsing accuracy. Besides, Logram needs to traverse all log lines for calculating the configuration

[1]Our data and source code can be found in the replication package at https://github.com/senseconcordia/PILARData_ICSE2023.git

TABLE II
CONFIGURATIONS USED IN DIFFERENT LOG PARSERS. THE LAST THREE COLUMNS INDICATE THE RANGE AND NUMBER OF VALUES CHOSEN FOR EACH PARAMETER FOR THE DIFFERENT DATASETS IN THE LOGPAI BENCHMARK [5] AND LOGRAM [18].

Log parsers	#Parameters	Parameters	Highest	Lowest	#Unique Values
Drain	2	st	0.7	0.2	5
		depth	6	3	4
AEL	2	minEventCount	10	2	4
		merge_percent	0.7	0.4	4
IPLoM	2	CT	0.9	0.25	7
		lowerBound	0.7	0.01	6
LenMa	1	threshold	1	0.5	10
Spell	1	tau	0.95	0.5	10
Logram	2	Bi_threshold	115	8	14
		Tri_threshold	95	5	15

parameters, which can be a burden when logs are input in a streaming manner.

Despite the importance and challenges of choosing configuration parameters for log parsers, there exists no prior research that focuses on the impact of these parameters to log parsing results. Therefore, to fill in this gap, in the next section (Section III), we conduct an empirical study of the influence of choosing different configuration parameter values on log parsing results.

III. STUDYING THE INFLUENCE OF CONFIGURATION PARAMETER VALUES OF LOG PARSERS

TABLE III
THE SUBJECT LOG DATA USED IN OUR EVALUATION.

Dataset	Description	Size
Android	Android system log	183.37MB
Apache	Apache server error log	4.90MB
BGL	Blue Gene/L supercomputer log	708.76MB
Hadoop	Hadoop client-server job log	48.61MB
HDFS	Hadoop distributed file system log	1.47GB
HealthApp	Android Health app log	22.44MB
HPC	High performance cluster log	32.00MB
Linux	Linux system log	2.25MB
Mac	Mac OS log	16.09MB
OpenSSH	OpenSSH server log	70.02MB
OpenStack	OpenStack log	58.61MB
Proxifier	Proxifier log	2.42MB
Spark	Spark server-client log	2.75GB
Thunderbird	Thunderbird supercomputer log	29.60GB
Windows	Windows system log	26.09GB
Zookeeper	ZooKeeper service log	9.95MB

According to Design Science [29], in this section, we first conduct an empirical study to understand the influence of configuration parameters on log parsing results. We perform experiments on six existing log parsers (listed in Table II) and 16 log datasets (listed in Table III). A research question emerges as to whether the existing log parsers are configuration-insensitive, i.e., if one changes the parameter value of a log parser, whether the parsing results are stable or not. To answer this question, the study is performed in three aspects as listed below:

- **Varying parameters on the same dataset.** As mentioned in Section II-B, log parsers have a variety of parameters with different values assigned to them. Such variations in parameters could have an impact on the parsing results. As a result, in this aspect, we assess the log parsers with different parameter values while using the same dataset to examine if the parsers are parameter-insensitive.
- **Fixed parameters on different datasets.** When practitioners adopt a log parser on their own dataset, they typically do not have the knowledge of the best parameter values for their dataset and may need to depend on the choice of parameter values from existing benchmarks. Therefore, to investigate the effect of parameters on different datasets, we use fixed parameters with different datasets to study this aspect.
- **Fixed parameters on different samples of the same dataset.** Logs are changing from time to time. Even from the same software system, the logs produced during different periods of a day may differ significantly; while it is often not practical or realistic to tune parameter values of a log parser frequently during a day. Therefore, we would like to know with the same parameter values, whether different samples of the same log dataset produce consistent parsing results and how different parameter values influence the parsing results on different samples.

A. Varying parameters on the same dataset

Approach. To evaluate the sensitivity of log parsers' parameters, in this aspect, we tweak the parameter values on the same dataset to observe how the results change. Specifically, we choose three types of parameter values for the study: the highest, lowest and default values that are chosen in the LogPai [5] benchmark, as well as the ones provided by Logram [18]. If the log parsers have multiple parameters, we only change one of them while leaving the others as default for each experiment. Since the analyzed log parsers only have a maximum of two parameters, we run each log parser three times for the one-parameter log parsers (i.e., the default value, the highest value, and the lowest value) and five times for the two-parameter log parsers (i.e., the default values and the four combinations of the highest and lowest values of the two parameters).

Influence on parsing accuracy. We gain insights based on the analysis of the resulting log parsing accuracy. The parsing accuracy is calculated by an automated accuracy evaluation approach used in the LogPai benchmark [5]. This approach examines the percentage of log instances that are correctly grouped into their corresponding log templates. If all log instances of the same template are grouped together and the group only contains the logs instances from the same template, the log instances associated with this template are considered correctly parsed.

Influence on the ranking of parsing accuracy. In addition to parsing accuracy, we also examine the impact of configuration parameters on the rank of a log parser. For each log dataset, we

TABLE IV
PARSING ACCURACY AND RANKING OF SELECTED LOG PARSERS WHEN ALTERING PARAMETER VALUES OF THEIR CONFIGURATIONS.

		Android		Apache		BGL		Hadoop		HDFS		HealthApp		HPC		Linux	
		Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)
AEL	Highest	0.75	4(-1)	1.00	1(0)	0.96	1(0)	0.95	1(-5)	1.00	1(0)	0.63	5(0)	0.90	2(0)	0.68	3(0)
	Lowest	0.66	5(0)	1.00	1(0)	0.82	3(+2)	0.54	6(0)	0.84	5(+4)	0.27	5(0)	0.83	4(+2)	0.67	3(0)
	Default	0.68	5	1.00	1	0.96	1	0.54	6	1.00	1	0.57	5	0.90	2	0.67	3
	Gap	0.09		0.00		0.14		0.41		0.16		0.36		0.07		0.01	
Drain	Highest	0.91	2(0)	1.00	1(0)	0.97	1(0)	0.96	1(0)	1.00	1(0)	0.78	3(0)	0.89	3(0)	0.69	2(0)
	Lowest	0.61	5(+3)	1.00	1(0)	0.82	3(+2)	0.88	4(+3)	0.65	6(+5)	0.18	5(+2)	0.83	4(+1)	0.24	5(+3)
	Default	0.91	2	1.00	1	0.96	1	0.95	1	1.00	1	0.78	3	0.89	3	0.69	2
	Gap	0.30		0.00		0.15		0.08		0.35		0.60		0.06		0.45	
IPLoM	Highest	0.79	4(0)	1.00	1(0)	0.94	3(0)	0.95	1(0)	1.00	1(0)	0.83	2(0)	0.83	4(-1)	0.67	3(0)
	Lowest	0.70	4(0)	1.00	1(0)	0.93	3(0)	0.63	5(+4)	0.83	5(+4)	0.82	2(0)	0.82	5(0)	0.67	3(0)
	Default	0.71	4	1.00	1	0.94	3	0.95	1	1.00	1	0.82	2	0.82	5	0.67	3
	Gap	0.09		0.00		0.01		0.32		0.17		0.01		0.01		0.00	
LenMa	Highest	0.91	3(0)	1.00	1(0)	0.69	6(0)	0.89	4(0)	1.00	1(0)	0.17	6(0)	0.84	4(0)	0.70	1(0)
	Lowest	0.88	3(0)	1.00	1(0)	0.58	6(0)	0.53	6(+2)	0.51	6(+5)	0.16	6(0)	0.79	5(+1)	0.18	5(+4)
	Default	0.88	3	1.00	1	0.69	6	0.89	4	1.00	1	0.17	6	0.83	4	0.70	1
	Gap	0.03		0.00		0.11		0.36		0.49		0.01		0.05		0.52	
Logram	Highest	0.63	6(0)	1.00	1(0)	0.78	5(0)	0.92	3(0)	0.81	6(0)	0.99	1(0)	0.99	1(0)	0.14	6(0)
	Lowest	0.57	6(0)	1.00	1(0)	0.74	5(0)	0.90	3(0)	0.81	6(0)	0.93	1(0)	0.97	1(0)	0.10	6(0)
	Default	0.58	6	1.00	1	0.75	5	0.91	3	0.81	6	0.99	1	0.97	1	0.10	6
	Gap	0.06		0.00		0.04		0.02		0.00		0.06		0.02		0.04	
Spell	Highest	0.92	1(0)	1.00	1(0)	0.79	4(0)	0.78	5(0)	1.00	1(0)	0.64	4(0)	0.65	6(0)	0.61	5(0)
	Lowest	0.59	5(+4)	0.31	6(+5)	0.27	6(+2)	0.26	6(+1)	0.32	6(+5)	0.16	6(+2)	0.53	6(0)	0.17	5(0)
	Default	0.92	1	1.00	1	0.79	4	0.78	5	1.00	1	0.64	4	0.65	6	0.61	5
	Gap	0.33		0.69		0.52		0.52		0.68		0.48		0.12		0.44	

		Mac		OpenStack		OpenSSH		Proxifier		Spark		Thunderbird		Windows		Zookeeper	
		Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)	Acc.	rank (Δ)
AEL	Highest	0.85	1(-1)	0.79	2(0)	0.54	5(0)	0.52	3(0)	0.92	1(-2)	0.95	2(0)	0.69	4(0)	0.92	5(0)
	Lowest	0.74	3(+1)	0.25	5(+3)	0.52	5(0)	0.50	5(+2)	0.91	3(0)	0.93	3(+1)	0.57	4(0)	0.78	6(+1)
	Default	0.76	2	0.76	2	0.54	5	0.52	3	0.91	3	0.94	2	0.69	4	0.92	5
	Gap	0.11		0.54		0.02		0.02		0.01		0.02		0.12		0.14	
Drain	Highest	0.86	1(0)	0.73	5(0)	0.79	3(0)	0.53	1(0)	0.92	1(0)	0.96	1(0)	1.00	1(0)	0.99	1(0)
	Lowest	0.71	3(+2)	0.24	5(0)	0.72	3(0)	0.03	5(+4)	0.54	6(+5)	0.66	4(+3)	0.57	3(+2)	0.96	1(0)
	Default	0.79	1	0.73	5	0.79	3	0.53	1	0.92	1	0.96	1	1.00	1	0.97	1
	Gap	0.15		0.49		0.07		0.50		0.38		0.30		0.43		0.03	
IPLoM	Highest	0.67	5(0)	0.87	1(0)	0.80	2(0)	0.52	3(0)	0.92	1(0)	0.66	5(0)	0.68	5(0)	0.98	1(-1)
	Lowest	0.67	5(0)	0.31	5(+4)	0.33	5(+3)	0.04	5(+2)	0.76	6(+5)	0.64	5(0)	0.57	5(0)	0.96	2(0)
	Default	0.67	5	0.87	1	0.80	2	0.52	3	0.92	1	0.66	5	0.57	5	0.96	2
	Gap	0.00		0.56		0.47		0.48		0.16		0.02		0.11		0.02	
LenMa	Highest	0.70	4(0)	0.74	4(0)	0.93	1(0)	0.52	3(-2)	0.88	6(0)	0.94	2(0)	0.70	4(-1)	0.96	2(-4)
	Lowest	0.66	6(+2)	0.33	5(+1)	0.72	3(+2)	0.03	5(0)	0.57	6(0)	0.24	6(+4)	0.57	5(0)	0.74	6(0)
	Default	0.70	4	0.74	4	0.93	1	0.51	5	0.88	6	0.94	2	0.57	5	0.84	6
	Gap	0.04		0.41		0.21		0.49		0.31		0.70		0.13		0.22	
Logram	Highest	0.71	4(-1)	0.33	6(0)	0.51	6(0)	0.03	6(0)	0.92	1(-4)	0.48	6(0)	0.97	3(0)	0.96	2(0)
	Lowest	0.59	6(+1)	0.22	6(0)	0.33	6(0)	0.00	6(0)	0.89	5(0)	0.46	6(0)	0.70	3(0)	0.91	5(+3)
	Default	0.67	5	0.23	6	0.33	6	0.03	6	0.89	5	0.47	6	0.97	3	0.96	2
	Gap	0.12		0.11		0.18		0.03		0.03		0.02		0.27		0.05	
Spell	Highest	0.76	2(0)	0.76	2(0)	0.55	4(0)	0.53	1(0)	0.92	1(-2)	0.84	4(0)	0.99	2(0)	0.96	2(0)
	Lowest	0.36	6(+4)	0.14	6(+4)	0.25	6(+2)	0.49	5(+4)	0.03	6(+3)	0.19	6(+2)	0.40	6(+4)	0.69	6(+4)
	Default	0.76	2	0.76	2	0.55	4	0.53	1	0.91	3	0.84	4	0.99	2	0.96	2
	Gap	0.40		0.62		0.30		0.04		0.89		0.65		0.59		0.27	

Column **Acc.** denotes the parsing accuracy of a studied log parser. Each log parser has three or five parsing results on a single dataset depending on the number of parameters. Row **Gap** denotes the difference between the highest and lowest accuracy. Column **rank (Δ)** denotes the parsing accuracy ranking of the analyzed log parser with different parameter values across all six log parsers. In this column, the number in the parenthesis indicates the rank changes compared to the parsing accuracy ranking using the default parameters.

first rank all log parsers that use their default parameter values. Then, we examine how adjusting a log parser's parameter values affects its ranking against other parsers when keeping the default parameter values.

Results. Table IV presents the results of this aspect.

All studied log parsers can obtain significantly different parsing accuracy using different parameter values. According to Table IV, in most of the cases, the Gap values are considerably larger than 0, indicating the difference in the parsing accuracy when different parameter values are used. Overall, the six studied log parsers have the maximum Gap values of 0.27 to 0.89 across the studied datasets. In particular, the Gap values are fairly large in many cases. For example, Drain has a maximum gap of 0.60 on HealthApp, LenMa has a maximum gap of 0.70 on Thunderbird, Spell has a maximum gap of 0.89 on Spark. Logram has the most stable parsing results in terms of the parsing accuracy, with a maximum Gap value of 0.27 across the different log datasets. The average Gap value of all log parsers on all log datasets is 0.23, which means that altering parameter values can cause parse accuracy to fluctuate by 23% on average. Such a huge accuracy drop may lead to an accurate log parser becoming inaccurate.

The choice of configuration parameter values can have a profound effect on ranking of log parsers in terms of parsing accuracy. As shown in Table IV, in the majority of cases (58/96), altering parameters changes the parsing accuracy ranking across all log parsers (i.e, at least one rank Δ is not equal to 0 in a Table cell). A significant rank shift could be found in all log parsers. Apart from Logram, which has a maximum of four rank slot adjustment, the other five log parsers can have a highly extreme ranking change, with the best becoming the worst or vice versa. Furthermore, Table IV shows that some of the default parameter values chosen by developers cannot achieve the best accuracy, such as using AEL to parse the Android dataset. This observation reveals that it is difficult in practice to select an optimal parameter for log parsers.

The existing log parsers are sensitive to parameters, as the parsing accuracy fluctuates significantly while tweaking the parameter values. Developers may have difficulties in selecting optimal parameter values as the default parameters may not result in best results.

B. Fixed parameters on different datasets

Approach. This aspect intends to examine if the existing log parsers' configuration parameters are insensitive to datasets. To achieve this goal, we conduct the study on various datasets with fixed parser parameters. Considering that studies with just one combination of parameter values could result in data bias, we use a comprehensive set of parameter value combinations which comprise of all possible parameter values found in real-world implementations.

The experiment consists of three steps. First, we extract all parameter values that developers used in the LogPai benchmark [5]. The set of parameter value combinations for this experiment is established by the value combinations used in the real-world implementations. Then, for each combination of parameter values, we parse all 16 datasets to get accuracy results using the same approach as in Section III-A. Lastly, we summarize and analyze the results.

Evaluation metrics. To analyze experiment results, we use three metrics: Inter-Quartile Range (IQR), number of top accuracy, and average accuracy. IQR refers to the difference between the first quartile and the third quartiles of the accuracy distribution across the log datasets. Since there are 16 datasets involved in this study, we can acquire 16 data points for each combination of log parser parameter values. Some datasets may have an overall lower accuracy (e.g., Proxifier), leading to relatively smaller IQR. Therefore, we use the highest accuracy of each dataset to normalize the corresponding accuracy before calculating IQR. We leverage the length of IQR on these 16 data points to gauge the variation in log parsing accuracy on different datasets (the higher the IQR length, the more varied the results). Number of top accuracy denotes the number of datasets where the log parser using the combination of parameters achieves the highest accuracy when compared to the parsing accuracy using the other combinations of parameter values. We would like to use this metric to examine if the parameters of the log parsers behave consistently across datasets. To be more specific, if a log parser using a combination of parameter values achieves the best accuracy on a dataset when compared to all other available parameter values, we expect to know if it still can reach the best accuracy on other datasets using the same parameter values. In addition, we calculate the average accuracy of a log parser using a combination of parameter values across all 16 datasets.

Results. The experiment results are presented in Table V. **None of the existing parsers with a fixed setting of parameters can consistently achieve accurate parsing results across the different datasets.** According to Table V, the **Ave. Acc** values range from 0.41 to 0.78, with an average of 0.68. On the 16 datasets analyzed, none of the log parsers with a fixed setting of parameters produced an average accuracy greater than 0.8, indicating that these log parsers with fixed parameter values have a limitation in their ability to accurately parse logs. Column **IQR** in Table V exhibits a major fluctuation in parsing accuracy (with an average IQR value of 0.18) when log parsers parse different datasets with fixed parameters. In particular, Spell has an average IQR of 0.39 with a low average accuracy of 0.57.

The optimal parameter value of a parser varies across different datasets. The column **#Top Acc.** in Table V indicates the number of datasets on which a parser with a fixed parameter value performs the best among all parameter values of the same parser. As shown in the column, a setting of parameters typically achieves the best parsing accuracy for a few log datasets (e.g., each parameter value achieves the best performance on three to five datasets). The highest **#Top Acc.** values of all the log parsers only range from 5 to 9 (out of 16 log datasets). Our results indicate that different datasets tend to favor different parameter values. In other words, the setting of the parser parameters are sensitive to the datasets to be parsed.

Although there may exist some mild cases where log parsers appear to be less sensitive to datasets, none of these cases can reach good results on the three metrics that we looked at. For example, although the log parser IPLoM, which generally gives the most steady IQR values, has an acceptable average IQR value (0.04), its average accuracy (0.72) is barely adequate. Even in the best scenario, where IPLoM yields the highest value in column **#Top Acc.** with CT=0.58 and lb=0.25, there are still seven datasets where it fails to achieve the highest parsing accuracy.

The results of this aspect indicates that the parameters of the existing log parsers are sensitive to datasets: no single parameter value can consistently achieve optimal results across different datasets; the optimal parameter value of a parser varies across different datasets.

C. Fixed parameters on different samples of the same dataset

Approach. In this aspect, we investigate if existing log parsers are sensitive to different data samples. To achieve this goal, we conduct study on various data samples of the same datasets using fixed parser parameters.

Generating sample datasets. For each data sample, we extract 10,000 continuous log lines from the original dataset.

TABLE V
IQR, AVERAGE ACCURACY AND NUMBER OF TOP ACCURACY FOR LOG PARSERS WITH FIXED PARAMETERS ACROSS DIFFERENT DATASETS

Parser	Para. Values	IQR	#Top Acc.	Ave. Acc.
Drain	st=0.2, d=6	0.20	3	0.76
	st=0.5, d=4	0.30	4	0.72
	st=0.2, d=4	0.23	5	0.75
	st=0.39, d=6	0.19	4	0.75
	st=0.7, d=6	0.32	3	0.72
	st=0.5, d=5	0.19	3	0.75
	st=0.6, d=5	0.12	5	0.77
	st=0.6, d=3	0.17	4	0.76
IPLoM	CT=0.25, lb=0.3	0.11	2	0.70
	CT=0.3, lb=0.4	0.05	4	0.71
	CT=0.4, lb=0.01	0.04	5	0.72
	CT=0.4, lb=0.2	0.03	6	0.72
	CT=0.35, lb=0.25	0.01	7	0.72
	CT=0.58, lb=0.25	<0.01	9	0.74
	CT=0.3, lb=0.3	0.05	5	0.71
	CT=0.3, lb=0.25	0.05	5	0.71
	CT=0.9, lb=0.25	0.03	6	0.73
	CT=0.78, lb=0.25	0.02	9	0.74
	CT=0.35, lb=0.3	0.01	7	0.72
	CT=0.3, lb=0.2	0.05	5	0.71
	CT=0.4, lb=0.7	0.07	7	0.71

Parser	Para. Values	IQR	#Top Acc.	Ave. Acc.
LenMa	threshold=0.86	0.06	8	0.71
	threshold=0.91	0.02	9	0.73
	threshold=0.7	0.08	7	0.71
	threshold=0.9	0.02	9	0.73
	threshold=0.5	0.14	6	0.70
	threshold=0.8	0.08	8	0.71
	threshold=0.88	0.02	9	0.73
	threshold=1	0.33	4	0.60
	threshold=0.6	0.08	8	0.71
	threshold=0.78	0.08	7	0.71
Spell	tau=0.95	0.48	2	0.41
	tau=0.6	0.33	4	0.66
	tau=0.75	0.36	4	0.64
	tau=0.7	0.33	3	0.65
	tau=0.5	0.42	2	0.56
	tau=0.65	0.36	4	0.62
	tau=0.55	0.33	0	0.64
	tau=0.9	0.52	2	0.45
	tau=0.8	0.41	1	0.55
	tau=0.85	0.40	1	0.48

Parser	Para. Values	IQR	#Top Acc.	Ave. Acc.
AEL	ec=2, mp=0.6	0.10	6	0.76
	ec=2, mp=0.4	0.04	4	0.74
	ec=2, mp=0.5	0.06	5	0.76
	ec=5, mp=0.4	0.10	6	0.76
	ec=6, mp=0.5	0.09	4	0.78
	ec=10, mp=0.7	0.07	5	0.77
Logram	bi=14, tri=13	0.20	5	0.66
	bi=75, tri=32	0.17	3	0.66
	bi=18, tri=10	0.27	3	0.67
	bi=9, tri=6	0.29	3	0.65
	bi=15, tri=15	0.20	5	0.66
	bi=23, tri=5	0.31	5	0.65
	bi=13, tri=11	0.26	4	0.67
	bi=32, tri=24	0.18	5	0.65
	bi=11, tri=10	0.26	4	0.67
	bi=16, tri=9	0.29	4	0.67
	bi=47, tri=80	0.12	3	0.66
	bi=115, tri=95	0.32	4	0.62
	bi=37, tri=37	0.11	5	0.67
	bi=8, tri=7	0.30	3	0.65
	bi=16, tri=16	0.24	6	0.67
	bi=9, tri=9	0.29	5	0.67

Column **Para. Values** denotes possible combinations of parameter values for log parsers. Column **IQR** denotes the length of IQR. Column **# of Top Acc.** displays the number of top accuracy. Column **Ave. Acc.** is the average accuracy.

As we do not have the parsing ground truth data for each sample, we use a sliding window sampling approach such that each two consecutive samples share 5,000 overlapping log lines. The shared 5,000 log lines of two consecutive samples can be used to compare the parsing results of the two samples (i.e., to determine the agreement of parsing). From each log dataset, we extract 11 such samples with each two consecutive samples sharing 5,000 overlapping log lines. We then use all six log parsers to parse each of the samples. Every 5,000 overlapping log lines shared by two consecutive samples are parsed twice in the two samples, thus we can determine the matching between the parsing results (i.e., the agreement of parsing). We use the overlaps between the sliding window samples instead of the 2,000 labeled log lines in the LogPai benchmark, because the 2,000 labeled log lines are sampled randomly and not sufficient (too small) for evaluating the impact of choosing different samples of the same dataset. We vary the parameter values of the log parsers to examine the influence of the parameters on the parsing agreement. The experiment is conducted on 11 datasets as the rest five datasets (i.e., Android, Apache, Linux, OpenStack and Proxifier) do not have sufficient log lines for sampling.

Measuring agreement of parsing results. For each log dataset, there are in total 55,000 log lines (10 pairs of consecutive samples, each with 5,000 overlapping log lines) that are parsed in two samples. The agreement of the parsing results on a dataset is then the percentage of the 55,000 log lines which have identical parsing results from the two samples. Thus, for each log parser with each parameter value and each log dataset, we obtain an overall agreement value for the 55,000 overlapping log lines. To understand the distribution of the agreement across different parameter values, we use four metrics: highest agreement, lowest agreement, agreement gap and average agreement. For each log parser and each log dataset, the highest/lowest/average agreement refers to the highest/lowest/average agreement achieved from the different parameter values. The agreement gap indicates the difference between the highest and lowest agreement across different parameter values.

Results. Table VI illustrates, for each log parser and each log dataset, the agreement achieved by different parameter values.

The log parsers may not achieve stable parsing results on different samples of the same dataset. According to Table VI, in many cases (14/66), the average agreement values are lower than 0.9 (as highlighted in the table), which indicates that using the same parser on two samples of the same dataset can achieve inconsistent results. In particular, in some datasets (e.g., when AEL is used to parsing the HPC samples), the average agreement values can be as low as 0.5, which means that nearly half of the parsing results mismatch when the parser is used to parse different samples of the same dataset.

Different parameter values can cause different levels of disagreement in the parsing results of different samples. The **Gap** rows in Table VI indicate the largest difference between the agreement values achieved by different parameter values. The **Gap** values range from 0 to 0.46 with an average of 0.07, indicating that different parameter values can lead to very different levels of disagreement in the parsing results of two samples. Our results indicate that the parameter values of the log parsers can impact its stability when parsing different samples of the same dataset.

The log parsers may not achieve stable parsing results on different samples of the same log dataset. Furthermore, the parameter values can influence such stability of the log parser on different samples.

TABLE VI
AGREEMENT RESULTS ACROSS VARIOUS DATA SAMPLES WITH IDENTICAL DATASETS AND FIXED PARAMETER VALUES.

		BGL	Hadoop	HDFS	HealthApp	HPC	Mac	OpenSSH	Spark	Thunderbird	Windows	Zookeeper
AEL	Highest	>0.99	0.96	>0.99	0.98	**0.51**	0.94	**0.79**	0.96	0.95	0.95	>0.99
	Lowest	0.99	0.94	0.95	0.90	**0.49**	0.93	**0.76**	0.93	0.95	**0.87**	0.93
	Gap	<0.01	0.02	0.05	0.08	0.02	0.01	0.03	0.03	<0.01	0.08	0.06
	Average	>0.99	0.95	0.97	0.93	**0.50**	0.94	**0.78**	0.94	0.95	0.92	0.95
Drain	Highest	>0.99	0.97	0.95	>0.99	0.90	0.96	0.94	0.97	0.98	0.95	0.93
	Lowest	>0.99	0.96	0.95	0.94	**0.80**	0.95	**0.82**	0.90	0.95	**0.86**	0.93
	Gap	<0.01	0.01	<0.01	0.06	0.10	0.01	**0.12**	0.07	0.03	0.09	¡0.01
	Average	>0.99	0.97	0.95	0.97	**0.81**	0.96	**0.84**	0.95	0.96	0.93	0.93
IPLoM	Highest	>0.99	0.97	0.98	0.96	>0.99	0.92	0.90	0.96	0.97	0.94	0.93
	Lowest	0.90	0.95	0.95	0.95	**0.79**	0.90	**0.81**	**0.89**	0.96	**0.78**	0.92
	Gap	0.10	0.02	0.03	0.01	**0.20**	0.02	0.09	0.07	0.01	**0.16**	0.01
	Average	0.98	0.96	0.96	0.96	**0.82**	0.91	**0.83**	0.94	0.96	**0.89**	0.93
LenMa	Highest	>0.99	0.96	0.95	>0.99	**0.80**	0.98	0.91	0.97	0.97	0.95	>0.99
	Lowest	>0.99	**0.86**	0.95	>0.99	**0.80**	0.96	**0.80**	0.92	0.97	**0.66**	0.93
	Gap	<0.01	0.10	<0.01	<0.01	<0.01	0.02	**0.11**	0.05	<0.01	**0.29**	0.07
	Average	>0.99	0.95	0.95	>0.99	**0.80**	0.97	**0.86**	0.93	0.97	0.91	0.97
Logram	Highest	>0.99	0.98	>0.99	>0.99	>0.99	0.94	>0.99	0.98	0.91	0.98	0.99
	Lowest	0.99	0.91	>0.99	0.96	>0.99	**0.81**	0.99	0.94	**0.84**	0.94	0.99
	Gap	<0.01	0.07	<0.01	0.03	<0.01	**0.13**	<0.01	0.04	0.07	0.04	<0.01
	Average	>0.99	0.94	>0.99	0.97	>0.99	**0.86**	>0.99	0.95	**0.88**	0.98	0.99
Spell	Highest	>0.99	>0.99	>0.99	0.99	>0.99	>0.99	**0.88**	0.98	0.93	>0.99	>0.99
	Lowest	**0.66**	**0.88**	**0.85**	0.90	**0.79**	**0.86**	**0.42**	**0.86**	**0.83**	**0.86**	**0.88**
	Gap	**0.34**	**0.11**	**0.15**	0.09	**0.21**	**0.14**	**0.46**	**0.12**	0.10	**0.14**	**0.12**
	Average	0.96	0.93	0.94	0.95	**0.88**	0.93	**0.70**	0.91	**0.89**	0.93	0.93

Row **Highest/Lowest/Average** indicates the highest/lowest/average agreement achieved by different parameter values. Row **Gap** denotes the difference between the highest and lowest agreement. The **Highest/Lowest/Average** agreement values < 0.9 and the **Gap** values > 0.1 are highlighted.

IV. *PILAR*: A CONFIGURATION PARAMETER IN-SENSITIVE LOG PARSER

In this section, we present *PILAR*, a parameter insensitive approach to automatically parsing logs.

Overall idea of *PILAR*. In Section II and III, we find that the values of the configuration parameters of existing log parsers have a large influence on the parsing results. By examining their configuration parameters, we find that they are often domain specific. For example, clustering goodness and similarities are parameters of IPLoM and Drain, respectively, while the interpretation of these values may differ from domain to domain. Therefore, we would like to leverage a measure that is less prone to be influenced by the domains.

As the static parts of logs are natural language text ((mainly in English for LogPai) written by developers, their entropy would be likely to all follow the same characteristics, i.e., stable to each other [19]. Meanwhile, as logs generated from the same logging statement have the same static parts, the static parts of logs are highly repetitive. These facts suggest the possibility of a log parsing framework that is not sensitive to configuration parameters by utilizing the entropy bias between the dynamic and static parts of logs. Therefore, in our approach, we leverage an entropy based probability to determine whether a token in a log line is generated from the static text or dynamic variables.

Our approach consists of five phases, as illustrated in Figure 1: (i) processing raw logs to produce "cooked" logs (i.e., logs with no automatically generated formatting content) and using coarse-grained parsing to identify some simple dynamic content from "cooked" logs, (ii) generating n-grams dictionaries based on "cooked logs", (iii) calculating the entropy-based probability of each token, (iv) identifying dynamic and static log tokens (i.e., log tokens from dynamic and static content, respectively), and (v) generating log templates.

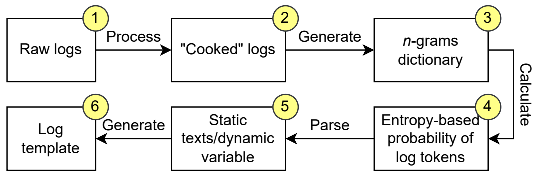

Fig. 1. An overview of *PILAR*'s approach.

A. Processing raw logs

Raw logs typically comprise two types of log content, i.e., the one automatically generated by the logging handler which usually follows the same format in one software system, and the customized content provided by developers. For example, "`2015-07-31 00:19:30,005 - INFO [ProcessThread(sid:2 cport:-1)::PrepRequestProcessor@476] - Processed session termination for sessionid: 0x14edfaa86f60019`" is a raw log line picked up from Zookeeper log data. "`2015-07-31 00:19:30,005 - INFO [ProcessThread(sid:2 cport:-1)::PrepRequestProcessor@476] -` " is the part which is automatically generated by log handler. This part specifies the log timestamp, log level, and environment information and follows the same format. Therefore, we can identify this kind of information with regular expressions simply. The main challenge of log parsing is to identify the dynamic part and static part in customized content provided by developers.

In order to extract the developer provided content for deeper analysis, we first filter out the formatting content (including log timestamp, log level, etc.) with regular expression and only maintain the logging content generated solely by logging statement. "`Processed session termination for sessionid: 0x14edfaa86f60019`" is the "Cooked log" after filtering out the formatting content.

After getting the "cooked" logs, we can extract dynamic content which follows a common template through regular expression, for example, IP address and numbers. We refer to the regular expressions which are freely available from LogPai to identify this kind of information. This kind of dynamic content will be replaced with wildcard characters (i.e., "<*>"). These characters are used in LogPai benchmark [5] to represent dynamic token. This phase is conducted before generating n-gram dictionaries phase. It is worth mentioning that this is a pre-processing step. All of the state-of-the-art log parsers studied in this paper have such a pre-processing step that relies on regular expressions. Developers who are familiar with the log data are expected to configure the regular expressions with minimum effort.

B. Generating n-grams dictionaries

The aforementioned content identification is coarse-grained, and parsing the logs further requires considerable attention and effort. Given the "cooked" logs produced by the initial phase, we first separate them into tokens to prepare for producing n-gram dictionaries. Each "cooked" log is separated by white-space characters, such as a blank space, tabulation, etc. Using this set of tokens, we then build n-gram dictionaries. According to a prior study [19], when $n \leq 3$, the re-occurrence of a n-gram in logs becomes steady. As a result, we build three dictionaries with n equal to 1, 2, and 3. If we take the following example (consisting of 5 "cooked" logs), we can make three dictionaries containing the information of single grams, double grams and triple grams, the key element is the n-gram appears in the log and its corresponding value is the number of times this n-gram appears in the log. For example, "`Processed`" is one key element in the 1-gram dictionary and its corresponding value element is 5; "`for→sessionid`" is one key element in the 2-gram dictionary and its corresponding value is 5; "`for→sessionid→0x14edfaa86f6002d`" is one key element in the 3-gram dictionary with corresponding value 1.

```
Processed session termination for sessionid:
0x14edfaa86f60019z
Processed session termination for sessionid:
0x14edfaa86f6002d
Processed session termination for sessionid:
0x24edfaa8717002d
Processed session termination for sessionid:
0x34edfaa9c22003d
Processed session termination for sessionid:
0x14edfaa86f60047
```

```
tokens  = cookedLog.split() // Get a list of tokens

for(index = 0; index < tokens.size(); index++){
// iterate over all tokens
    first = (index-1) % token.size();
    second = (index-2) % token.size();

    if(tokens[first] in Dynamic)
    // check if the first preceding token is in the list of Dynamic tokens
    // Dynamic list is initialized with the tokens list
    // calculate probability based on 1-gram
        probability = (# of 1-gram)/(# of log lines)
        // calculate probability
       <check probability>
        // check if probability ≥ threshold
        // If so, remove the scanned token from Dynamic
    else if(tokens[second] in Dynamic)
    // calculate probability based on 1-gram and 2-grams
        2-gram = tokens[first] + tokens[index]
        1-gram = tokens[first]
        probability = (# of 2-gram)/(# of 1-gram)
       <check probability>
    else
    // calculate probability based on 1-gram, 2-grams and 3-grams
        3-gram = tokens[second] + tokens[first] + tokens[index]
        2-gram = tokens[second] + tokens[first]
        probability = (# of 3-gram)/(# of 2-gram)
       <check probability>
}
```

Listing 1. Pseudo-code example for token entropy-based probability calculation.

C. Calculating the entropy-based probability of each token

Entropy is a physical concept that originally measures the degree of uncertainty or randomness. In this paper, we apply an entropy based probability to software logs. For a token under parsing, we use the conditional probability of the token's appearance given the appearances of the preceding tokens to measure the entropy of the token. A higher probability represents a lower entropy. Dynamic tokens generally have a lower probability than static tokens since static tokens always appear repeatedly. Thus, the lower a log token's entropy, the more likely it is a static token.

Listing 1 presents the pseudo-code example for calculating the probability for tokens. For each token under identification, we calculate its probability by considering at most two tokens immediately preceding it. Since the first log token in the log token list has no preceding token, we simply consider itself as a static token according to the development experience. Although there exist some logs whose content starts with dynamic tokens, most of these first tokens can be handled by regular expressions in the preprocessing part as they are usually simple numbers, dates and IPs. Only 4 logs among 2,000 in Android dataset, 31 among 2,000 logs in Thunderbird and 17 among 2,000 in Mac could affect the parsing result. We will further explore how to deal with the first token in the future. For the second token in the list, only one preceding token is taken into account. `Dynamic` represents the group of tokens which have been identified as dynamic tokens. It is initialized with `tokens` list as a void list. During the calculating process, only the preceding token which is identified as a static token will be used for calculating the probability. Based on this rule, we calculate the probability by three scenarios, as presented in Listing 1.

D. Identifying dynamic and static log tokens

With the entropy information from the prior step, we can now identify dynamic and static tokens. The method `<check probability>` in Listing 1 denotes a comparison of probability to a threshold, *which is the only configuration parameter of PILAR*. The examined token is regarded as a dynamic token if the probability is less than the threshold. The dynamic token will be appended to `Dynamic` list.

If we take the first "cooked" log in the box from Section IV-B as an example, according to Listing 1, the first token "Processed" will be considered as a static token. The probability of the following tokens (`session`, `termination`, `for`, `sessionid:`) is 1, and they are all static tokens. For the last token (`0x14edfaa86f60019z`), its probability is equal to the number of enclosing 3-grams appearances (i.e., 1) divided by the number of preceding 2-grams appearances (i.e., 5). The probability of the last token (0.2) is significantly less than that of other tokens (1). If the tool threshold is set to a lower value, we may overlook this dynamic token. However, because the logs in Section IV-B are only extracted from a tiny portion of log files to save writing space, the last token could be accurately identified as a dynamic token when used for a real-world software system.

E. Generating log templates

In this phase, we produce a log template based on the identification results from prior phases. Using the same log template structure as the LogPai benchmark [5], the dynamic content in "cooked" logs is replaced by wildcard characters (i.e., "<*>") for further research. Below is the log template for the first "cooked" log in the box from Section IV-B.

```
Processed session termination for sessionid: <*>
```

V. Evaluation of *PILAR*

In this section, we evaluate *PILAR* with two parts:
Part 1: Evaluating the influence of configuration parameter values of the parsing results of *PILAR*. From Section III, we can see that varying parameter can lead to problems with accuracy and stability in log parsing. Therefore, we should ensure *PILAR* is not sensitive to parameters compared to other log parsers.
Part 2: Evaluating *PILAR* with the general quality metrics of log parsers. Accuracy and efficiency are two dominant metrics used to access log parsers [5]. Therefore, in addition to knowing that *PILAR* is parameter insensitive, it is necessary to determine whether *PILAR* outperforms or is comparable to other log parsers based on these two metrics.

A. Evaluating the influence of configuration parameter values of PILAR

In this section, we conduct a study to evaluate the influence of configuration parameter values of *PILAR*. As we did in Section III, we applied *PILAR* on 16 datasets to see if and how altering parameter values, as well as varying datasets and data samples with fixed parameters, influence the parsing results. We follow the same procedure described in Section III. *PILAR* only has one parameter (i.e., probability threshold) that can be configured, as mentioned in Section IV-D. In theory, the range of the possible threshold values is [0, 1]. Since the outcome of parsing each token is binary, we use 0.5 as the maximum value of the threshold. The value 0.5 also agrees with the results of entropy value reported by prior work [30]. In this section, we select 30 thresholds ranging from 0.1 to 0.5 with 0.01 as the step, which span a wide variety of conceivable value ranges.

1) Varying parameters on the same dataset: In this aspect, we alter parameter values, same as Section III, to examine whether the parsing results remain stable. The results of this aspect are presented in Table VII.

Generally, *PILAR* is not sensitive to parameter values. According to Table VII, the minority of the gap values (5/16) are larger than 0.1, indicating that on most datasets, changing parameters will not result in accuracy changes larger than 0.1. Particularly, in many cases (6/16), the accuracy remains completely **constant** regardless of which value is assigned to the parameter. Moreover, when comparing *PILAR* to other log parsers, altering parameters does not result in a change in ranking in half of the cases, based on row **Rank (Δ)**. In the remaining cases, six cases have modest changes (2 or 3 ranking slot changes among a total of 7 slots) and only three cases have large changes (4 or 5 ranking slot changes). Although the rank slot changes appear large for HDFS and ZooKeeper, their gaps are modest (0.06 and 0.05), and the accuracy remains high (their lowest accuracy is 0.94 and 0.92) after experiencing such ranking changes. The only worst-case scenario is parsing OpenStack. The reason is that many repetitive dynamic tokens and varying static tokens exist in this dataset. Nevertheless, *PILAR* is still one of the best parsers for this dataset. However, as we can see, other parsers generate a large average accuracy fluctuation (0.46) on this dataset, and *PILAR* (gap value 0.55) is not far behind. In addition, according to Table IV, other log parsers yield low accuracy on this dataset, generally lower than *PILAR*. Finally, **Small gap** and **Average gap** confirm that *PILAR* is generally more stable than other parsers when it comes to altering parameter values, and on some datasets, *PILAR* is the most parameter-insensitive log parser.

2) Fixed parameters on different datasets: This aspect aims at evaluating how different datasets affect *PILAR* across multiple fixed parameters. The experiment, like Section V-A1, is performed on all available parameter values, but we only present 9 of them in Table VIII due to limited space, with each case having a 0.5 threshold interval between them. Table VIII shows that *PILAR* has low IQR values with a maximum of 0.06, which is lower than the IQR in the best case of some other parsers, such as 0.33 for Spell and 0.11 for Logram in Table V. Only the results of IPLoM can catch up with, yet still not as good as *PILAR*; while *PILAR* can generate more accurate results since *PILAR*'s lowest average accuracy (0.79) is greater than IPLoM's highest average accuracy (0.74) based on Tables VIII and V. Therefore, ***PILAR* is not sensitive to datasets while using fixed parameters.**

3) Fixed parameters on different samples of the same dataset: In this aspect, we perform study on different data samples within the same datasets using multiple fixed parameter values. Table IX presents the study results. According to the table, *PILAR* yields high agreements with only three agreements lower than 0.9 (0.89 for HDFS, 0.89 for Mac and 0.82 for Spark). Rows **Average** and **Highest average** show that *PILAR*'s average agreement is even higher or

TABLE VII
PARSING ACCURACY AND RANKING OF *PILAR* WHEN ALTERING ITS PARAMETER VALUES.

		Android	Apache	BGL	Hadoop	HDFS	HealthApp	HPC	Linux	Mac	OpenStack	OpenSSH	Proxifier	Spark	Thunderbird	Windows	Zookeeper
PILAR	Highest	0.78	1.00	0.93	0.96	1.00	1.00	0.99	0.27	0.79	0.91	0.62	0.52	0.94	0.94	0.99	0.97
	Lowest	0.67	1.00	0.83	0.94	0.94	0.98	0.85	0.15	0.69	0.36	0.43	0.52	0.94	0.90	0.99	0.92
	Average	0.77	1.00	0.88	0.94	0.99	1.00	0.91	0.20	0.74	0.59	0.54	0.52	0.94	0.92	0.99	0.96
	Gap	0.11	0.00	0.09	0.02	0.06	0.02	0.14	0.12	0.10	0.55	0.18	0.00	0.00	0.03	0.01	0.05
	Rank Δ	3	0	0	2	5	1	3	0	3	5	2	0	0	2	0	4
Other parsers	Smallest gap	0.03	0.00	0.01	0.02	0.00	0.01	0.01	0.00	0.00	0.11	0.02	0.02	0.01	0.02	0.11	0.02
	Average gap	0.15	0.12	0.16	0.29	0.31	0.25	0.06	0.24	0.14	0.46	0.21	0.26	0.30	0.29	0.28	0.12

Row **Highest/Lowest** denotes the highest/lowest accuracy achieved by *PILAR* when applied to a dataset with different parameter values. Row **Average** denotes the average accuracy. Row **Gap** denotes the difference between the highest and lowest accuracy. Row **Rank (Δ)** indicates the maximum accuracy rank changes caused by parameter value changes compared to the parsing accuracy of other parsers. Row **Smallest gap/Average gap** denotes the smallest/average gap achieved by other log parsers on the same datasets.

TABLE VIII
IQR, AVERAGE ACCURACY AND NUMBER OF TOP ACCURACY FOR *PILAR* WITH FIXED PARAMETERS ACROSS DIFFERENT DATASETS.

Parser	Para. Values	IQR	# of Top Acc.	Ave. Acc.
PILAR	threshold=0.10	0.05	4	0.81
	threshold=0.15	0.04	5	0.82
	threshold=0.20	0.04	6	0.82
	threshold=0.25	0.05	5	0.82
	threshold=0.30	0.06	5	0.79
	threshold=0.35	0.02	8	0.80
	threshold=0.40	0.01	8	0.81
	threshold=0.45	0.03	8	0.80
	threshold=0.50	0.04	5	0.79

Column **Para. Values** denotes possible parameter values for log parsers. Column **IQR** denotes the length of IQR. Column **# of Top Acc.** displays the number of top accuracy. Column **Ave. Acc.** is the average accuracy.

roughly comparable to the highest agreement achieved by other parsers. Though Logram has only two agreements lower than 0.9, the average agreement for Logram is lower than 0.9 on 2 datasets. For *PILAR*, the average agreement on all the datasets are higher than 0.9. Meanwhile, there is only one parameter that will result in an agreement lower than 0.9 for *PILAR* on HDFS(0.5) and Mac(0.5). Moreover, the gap values of *PILAR* are very small, with two cases (0.11) on HDFS and (0.14) on Spark over 0.1. Except for Spark, the gap values of *PILAR* are even smaller or very close to the smallest gap values obtained from other parsers on the same datasets. For HDFS and Spark, *PILAR* does not lag far behind the gaps of other parsers, which range from 0.01 to 0.15 and 0.03 to 0.12 based on Table VI and IX. In addition, *PILAR* has an average agreement that is better or roughly comparable to other parsers as reported by Table VI and IX. Therefore, ***PILAR* is not sensitive to data samples with identical datasets and fixed parameter values.**

PILAR is parameter-insensitive in all scenarios compared to all other log parsers.

B. Evaluating PILAR with the General Quality Metrics of Log Parsers

We evaluate log parsers based on two widely used metrics: accuracy and efficiency [5]. Accuracy measures a log parser's ability to correctly identify static text and dynamic variables in log messages, while efficiency measures a log parser's processing time.

Accuracy. For accuracy, we only consider the highest accuracy among all parameters on each dataset. The average accuracy for *PILAR* is 0.85, which is the second highest average parsing accuracy among all log parsers. The highest average accuracy is achieved by Drain, which is 0.88. For AEL, Lenma, Spell, IPLoM and Logram, the average accuracy is 0.82, 0.80, 0.81, 0.77 and 0.74 separately. Based on the LogPai benchmark, we also count the number of cases where the tools have either the highest accuracy or an accuracy above 0.9. *PILAR* can achieve the highest accuracy or an accuracy higher than 0.9 on 11 datasets, while Drain has 12. For AEL, Lenma, Spell, IPLoM and Logram, the number is 8, 7, 9, 6 and 8 separately. Although Drain has a slightly better accuracy, it may need parameter tuning to achieve the accuracy; while *PILAR* is much less sensitive with its parameter (see Section V-A).

Efficiency. We compare the end-to-end time for parsing 100 MB of Android logs, 500 MB of BGL logs and 1GB of HDFS, Windows and Spark logs, respectively. The smallest size of Android and BGL are due to their relatively smaller data available in the LogPai benchmark. For each dataset of logs, we also measure the time for parsing 300 KB, 1 MB, 10 MB logs, to understand whether the parsing time scales with the growth of log data. Except for Logram, which is designed to have extremely high efficiency, *PILAR* exceeds all other parsers in efficiency (see Figure 2). For example, *PILAR* is almost 2 times faster than the next most efficient log parser Drain on HDFS, and 3 times faster on BGL.

PILAR has the second highest accuracy and efficiency among all log parsers, being only slightly behind the most accurate parser Drain and the most efficient parser Logram.

VI. THREATS TO VALIDITY

Construct Validity. In this paper, we study and compare *PILAR* with six log parsers that are selected based on their high parsing results reported in prior research [5], [18]. Future research may further study the influence of configuration parameters of other log parsers. For the first two aspects in our study, the ground truth is based on the 2,000 manually labeled log lines per dataset from the *LogPai* benchmark. Future research may consider evaluating these aspects on a larger number of manually labeled logs. In addition, Khan et al. [31] present a new metrics to evaluate the accuracy of

TABLE IX

AGREEMENT RESULTS ACROSS VARIOUS DATA SAMPLES WITH IDENTICAL DATASETS AND FIXED PARAMETER VALUES FOR *PILAR*.

		BGL	Hadoop	HDFS	HealthApp	HPC	Mac	OpenSSH	Spark	Thunderbird	Windows	Zookeeper
PILAR	Highest	>0.99	0.98	>0.99	>0.99	>0.99	0.97	>0.99	0.97	0.98	0.98	0.99
	Lowest	>0.99	0.95	**0.89**	>0.99	0.90	**0.89**	0.96	**0.82**	0.95	0.94	0.92
	Average	>0.99	0.97	0.98	>0.99	>0.99	0.94	0.99	0.91	0.96	0.96	0.97
	Gap	<0.01	0.03	**0.11**	<0.01	0.10	0.08	0.04	**0.15**	0.03	0.04	0.06
Other parsers	Highest average	>0.99	0.97	>0.99	>0.99	>0.99	0.97	>0.99	0.95	0.97	0.98	0.99
	Smallest gap	<0.01	0.01	<0.01	<0.01	<0.01	<0.01	<0.01	0.03	<0.01	0.04	<0.01

Row **Highest/Lowest/Average** indicates the highest/lowest/average agreement achieved by different parameter values. Row **Gap** denotes the difference between the highest and lowest agreement. The **Highest/Lowest/Average** agreement values < 0.9 and the **Gap** values > 0.1 are highlighted. Row **Highest average/Smallest gap** is the highest average agreement/smallest gap of other log parsers.

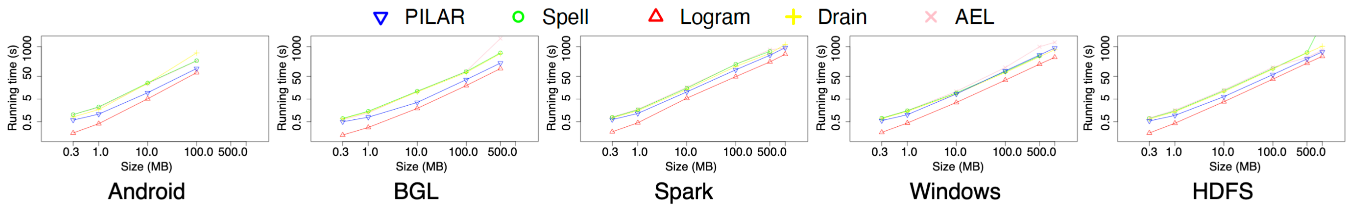

Fig. 2. The elapsed time of parsing five different log data with various sizes. The x and y axes are both in log scale.

parsing results. We plan to integrate this metric in our future work to evaluate the impact of parameters on parsing results.

Internal Validity. We leverage the automated parsing accuracy method from LogPai [5]. Prior research also proposed a stricter accuracy measure [18] that depends on manual examination and requires extensive manual effort, which is not suitable for this large-scale study. Future research may revisit our results with manually verified results or with other accuracy measures. Although being parameter-insensitive, *PILAR* may still further improve its results with choosing the optimal configuration by practitioners. Hence, we opt to keep the ability of manually choosing the parameter value when building the tool.

External Validity. Wieringa [32] provides strategies for generalization, among which we follow the lab-to-lab strategies. Evaluation of *PILAR* and other log parsers on additional log datasets can improve the generalizability of this paper.

VII. CONCLUSION

In this paper, we first conducted an empirical study on the influence of configuration parameters on the results of automated log parsers. We evaluated the influence in three aspects: 1) varying parameters on the same dataset, 2) fixed parameters on different datasets and 3) fixed parameters on different samples of the same dataset. The result shows that the configuration parameters have a strong influence on the parsing result, which illustrates the challenge of achieving optimal parsing results when these log parsers are adopted in practice. Therefore, we propose an entropy-based parameter insensitive log parser named *PILAR*. By evaluating *PILAR* through the same three aspects as our empirical study, we find that *PILAR* is the most insensitive log parser to the variations in parameters. Meanwhile, *PILAR* also has a near-top high accuracy and efficiency. This paper is the first research effort that studies and addresses the influence of configuration parameters in automated log parsers. Our paper can help ease the adoption of automated log parsers in practice.

REFERENCES

[1] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan, "A qualitative study of the benefits and costs of logging from developers' perspectives," *IEEE Transactions on Software Engineering*, 2020.

[2] B. Chen and Z. M. J. Jiang, "Characterizing and detecting anti-patterns in the logging code," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. IEEE Press, 2017, p. 71–81. [Online]. Available: https://doi.org/10.1109/ICSE.2017.15

[3] H. Li, W. Shang, Y. Zou, and A. E. Hassan, "Towards just-in-time suggestions for log changes," *Empirical Softw. Engg.*, vol. 22, no. 4, p. 1831–1865, Aug. 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9467-z

[4] C. Gülcü, *The complete log4j manual*. QOS. ch, 2003.

[5] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, "Tools and benchmarks for automated log parsing," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 121–130. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00021

[6] D. Yuan, H. Mai, W. Xiong, L. Tan, Y. Zhou, and S. Pasupathy, "Sherlog: Error diagnosis by connecting clues from run-time logs," in *Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XV. New York, NY, USA: Association for Computing Machinery, 2010, p. 143–154. [Online]. Available: https://doi.org/10.1145/1736020.1736038

[7] D. Yuan, S. Park, P. Huang, Y. Liu, M. M. Lee, X. Tang, Y. Zhou, and S. Savage, "Be conservative: Enhancing failure diagnosis with proactive logging," in *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'12. USA: USENIX Association, 2012, p. 293–306.

[8] M. D. Syer, Z. M. Jiang, M. Nagappan, A. E. Hassan, M. Nasser, and P. Flora, "Leveraging performance counters and execution logs to diagnose memory-related performance issues," in *2013 IEEE International Conference on Software Maintenance*, 2013, pp. 110–119.

[9] 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*, ser. SOSP '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–132. [Online]. Available: https://doi.org/10.1145/1629575.1629587

[10] Q. Fu, J.-G. Lou, Q. Lin, R. Ding, D. Zhang, and T. Xie, "Contextual analysis of program logs for understanding system behaviors," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. IEEE Press, 2013, p. 397–400.

[11] J. Zhu, P. He, Q. Fu, H. Zhang, M. R. Lyu, and D. Zhang, "Learning to log: Helping developers make informed logging decisions," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. IEEE Press, 2015, p. 415–425.

[12] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. USA: USENIX Association, 2012, p. 26.

[13] K. Yao, G. B. de Pádua, W. Shang, C. Sporea, A. Toma, and S. Sajedi, "Log4perf: suggesting and updating logging locations for web-based systems' performance monitoring," *Empir. Softw. Eng.*, vol. 25, no. 1, pp. 488–531, 2020. [Online]. Available: https://doi.org/10.1007/s10664-019-09748-z

[14] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automated performance analysis of load tests," in *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*. IEEE Computer Society, 2009, pp. 125–134. [Online]. Available: https://doi.org/10.1109/ICSM.2009.5306331

[15] W. Shang, A. E. Hassan, M. N. Nasser, and P. Flora, "Automated detection of performance regressions using regression models on clustered performance counters," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, January 31 - February 4, 2015*, L. K. John, C. U. Smith, K. Sachs, and C. M. Lladó, Eds. ACM, 2015, pp. 15–26. [Online]. Available: https://doi.org/10.1145/2668930.2688052

[16] 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 2017, Honolulu, HI, USA, June 25-30, 2017*, I. Altintas and S. Chen, Eds. IEEE, 2017, pp. 33–40. [Online]. Available: https://doi.org/10.1109/ICWS.2017.13

[17] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "An automated approach for abstracting execution logs to execution events," *Journal of Software Maintenance*, vol. 20, no. 4, pp. 249–267, 2008.

[18] H. Dai, H. Li, C. Chen, W. Shang, and T.-H. P. Chen, "Logram: Efficient log parsing using n-gram dictionaries," *IEEE Transactions on Software Engineering*, vol. PP, 07 2020.

[19] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 178–189.

[20] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *IEEE 16th International Conference on Data Mining, ICDM 2016, December 12-15, 2016, Barcelona, Spain*, F. Bonchi, J. Domingo-Ferrer, R. Baeza-Yates, Z. Zhou, and X. Wu, Eds. IEEE Computer Society, 2016, pp. 859–864. [Online]. Available: https://doi.org/10.1109/ICDM.2016.0103

[21] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009*, 2009, pp. 117–132.

[22] ——, "Online system problem detection by mining patterns of console logs," in *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, 2009, pp. 588–597.

[23] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *2010 USENIX Annual Technical Conference, Boston, MA, USA, June 23-25, 2010*, 2010.

[24] Q. Fu, J. Lou, Y. Wang, and J. Li, "Execution anomaly detection in distributed systems through unstructured log analysis," in *ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009*, 2009, pp. 149–158.

[25] Z. M. Jiang, A. E. Hassan, G. Hamann, and P. Flora, "Automatic identification of load testing problems," in *24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28 - October 4, 2008, Beijing, China*, 2008, pp. 307–316.

[26] "Automated root cause analysis for spark application failures - o'reilly media," https://www.oreilly.com/ideas/automated-root-cause-analysis-for-spark-application-failures, (Accessed on 08/13/2019).

[27] W. Shang, Z. M. Jiang, H. Hemmati, B. Adams, A. E. Hassan, and P. Martin, "Assisting developers of big data analytics applications when deploying on hadoop clouds," in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, 2013, pp. 402–411.

[28] 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)*, June 2016, pp. 654–661.

[29] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, pp. 75–105, 2004.

[30] K. Yao, H. Li, W. Shang, and A. E. Hassan, "A study of the performance of general compressors on log files," *Empirical Softw. Engg.*, vol. 25, no. 5, p. 3043–3085, sep 2020. [Online]. Available: https://doi.org/10.1007/s10664-020-09822-x

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

[32] R. Wieringa and M. Daneva, "Six strategies for generalizing software engineering theories," *Science of computer programming*, vol. 101, pp. 136–152, 2015.