

# Log parsers' performance on raw logs from Android devices


João Alfredo Bessa   [ Universidade Federal do Amazonas | [joao.bessa@icomp.ufam.edu.br](mailto:joao.bessa@icomp.ufam.edu.br) ]

Ricardo Miranda Filho  [ Universidade Federal do Amazonas | [ricardo.filho@icomp.ufam.edu.br](mailto:ricardo.filho@icomp.ufam.edu.br) ]

Girlana Souza  [ Universidade Federal do Amazonas | [girlana.santos@icomp.ufam.edu.br](mailto:girlana.santos@icomp.ufam.edu.br) ]

Raimundo Barreto  [ Universidade Federal do Amazonas | [rbarreto@ufam.edu.br](mailto:rbarreto@ufam.edu.br) ]

Rosiane de Freitas  [ Universidade Federal do Amazonas | [rosiane@icomp.ufam.edu.br](mailto:rosiane@icomp.ufam.edu.br) ]

 Institute of Computing, Universidade Federal do Amazonas, Av. General Rodrigo Octávio Jordão Ramos, 6200, Coroado I, Manaus, AM, 69080-005, Brazil.

**Received:** 04 October 2024 • **Accepted:** 22 February 2025 • **Published:** 01 May 2025

**Abstract** Enhancing log file structure for improved analysis, commonly referred to as “Log Parsing”, holds significant importance in deciphering pertinent insights from software-generated records. This study undertakes a comprehensive comparison of ten parsing tools and models available within the Logpai collection, namely AEL, Brain, Drain, LFA, LogCluster, Logram, NuLog, SHISO, SLCT, and ULP focusing on raw logs sourced from Android Devices, extending a previous work. Our findings underscore a notable precision deficit in models lacking preprocessing steps, as existing tools encounter considerable challenges in managing untreated logs. Consequently, these tools exhibit suboptimal performance levels when analyzing information gleaned from raw Android Logs of the same origin as the reference logs. When analyzing other blocks, such as wifi networks, the difficulty of dealing with small variations in format was persistent.

**Keywords:** Data mining, embedded systems, log analysis, parsing, regular expression.

## 1 Introduction

With the continuous advancement of technology and the widespread utilization of Android devices, the proliferation of system records, or Logs, has become integral to mobile computing. Effective analysis of these Logs is imperative for early fault detection, performance optimization, and bolstering device security. However, prevailing analytical tools encounter formidable challenges stemming from the diverse log formats, extensive data breadth, and escalating data volume Zhu *et al.* [2023]. This article presents findings from a study investigating the effectiveness and efficiency of log parsers, tools designed to structure raw data into organized formats to analyze and monitor these records.

The Android logging system is managed by logd, which organizes logs from the system and the device's applications into eight distinct buffers. Each log provider has its own rules for formatting, information, and structure, with these data segments referred to as blocks. These buffers are temporarily stored in a file called BugReport. This file unifies the entire collection of device logs into one or more files (depending on the manufacturer).

The study focuses on the Logpai collection, which furnishes a Python platform for primary parsers. This collection evaluates datasets sourced from diverse systems, which are pre-processed to ensure uniformity. The primary objective is to compare the performance of various parsers, particularly emphasizing their efficacy with raw data originating from Android devices manufactured by entities such as Samsung, Motorola, and Google. Data acquisition occurs through BugReport, a tool for logging on Android devices. Owing to the constraints of the Logpai tool, which processes one log

structure at a time, two log types are selected for detailed comparative analysis, SystemLog/LogCat and Wake Lock, and a compilation of the performance of the 3 best performing models in the context of wifi related log blocks.

The analysis encompasses both quantitative and qualitative assessments of parsing models, leveraging benchmarks to gauge processing capacity while considering factors like accuracy, robustness, and efficiency. This work expands a previously developed research Bessa *et al.* [2024], containing more parsing models, log blocks, and more robust analysis. The article is structured into the following sections: Background, Android log parsing, Design of experiments, Benchmark results, and Concluding remarks.

## 2 Background

This section provides the theoretical support for the analysis presented in this research, focusing on the structure of log generation within Android processes and Log Parsing.

### 2.1 Mobile log data

Logs refer to the systematic recording of events, activities, and system-generated information in the context of mobile devices. These logs serve as valuable resources for understanding mobile device behavior, performance, and diagnostics. They encompass diverse types of logs, including system logs capturing system-level activities, application logs detailing app-specific events, and error logs providing information about malfunctions or anomalies within the device. Through the meticulous analysis of logs, developers, system administrators, and researchers gain insights into user interactions,

app usage patterns, resource utilization, and potential issues encountered on mobile platforms. These logs often contain timestamps, event details, error codes, and contextual information, forming a rich repository for diagnosing problems and optimizing mobile device functionalities Theys [1999]; Boase and Ling [2013].

Utilizing logs in mobile devices is multifaceted, pivotal in enhancing user experience, troubleshooting technical issues, and improving system performance. Application developers leverage logs to debug software, identify performance bottlenecks, and refine user interfaces based on usage patterns. System administrators utilize logs to monitor device health, track security breaches, and implement preventive measures against potential threats.

Moreover, researchers and data analysts extract valuable insights from logs by employing data mining, machine learning, or statistical techniques to uncover hidden patterns, user preferences, and trends in mobile device usage. The systematic analysis of logs empowers stakeholders to make informed decisions, enhance device functionality, and ensure a seamless user experience in the dynamic landscape of mobile technology Romero and Ventura [2007]; Hwang *et al.* [2016].

However, using logs on mobile devices also raises concerns regarding privacy, data security, and the ethical handling of user information. As logs often contain sensitive data, including user interactions, location information, and personal identifiers, protecting this information from unauthorized access or malicious exploitation is crucial. Adhering to privacy regulations, implementing robust encryption methods, and adopting anonymization techniques are essential to safeguard user privacy while harnessing the insights gleaned from logs. Balancing the benefits of log analysis with stringent data protection measures remains a critical consideration for stakeholders in the mobile ecosystem to maintain user trust Dhanaraj *et al.* [2021]; Zhang *et al.* [2012].

## 2.2 Log parsing

The Log pre-processing phase is essential for preparing raw data for more refined analysis. This process includes filtering out irrelevant data, correcting errors, structuring the data, and imputing missing values. The techniques employed can be categorized into two main groups: transformation and detection and visualization. Transformation techniques aim to modify the Logs to correct, complete, filter and/or format the data before analysis, thus improving data quality. On the other hand, detection and visualization techniques focus on identifying problems in the data, such as problematic message structures, making it easier to understand the data and identifying areas that need transformation Marin-Castro and Tello-Leal [2021].

In this context, transformation techniques are particularly interesting, especially in Android operating system logs with specific formatting rules based on their source code. The importance of the Log Parsing process is highlighted by He *et al.* (2017) He *et al.* [2017a], who describe Log Parsing as an initial step in Log mining to identify abnormal patterns and operational insights. The main information is structured

in this phase, as shown in Figure 1.

Log Message	
01-09 20:56:29.519 1058 771 771   tombstoned: received crash request for pid 2916	
Structured log	
Timestamp	01-09 20:56:29.519
UID	1058
PID	771
TID	771
Event	
Component	tombstoned
Constant	received crash request for pid <*>
Variables	2916

Figure 1. Illustration of a pre-processing example parsing an Android log .

Zhu *et al.* (2019) compiled several Log Parsing techniques, performing benchmarks to evaluate their performance in detecting constants and variables in Logs from different systems Zhu *et al.* [2019]. Other works stood out in these benchmarks, carried out by Loghub, for their high accuracy in processing Logs from Android's Main buffer He *et al.* [2017a]. He *et al.* (2017) use a fixed-depth tree to speed up log analysis, employing a tree structure to categorize log messages based on predefined rules. On the other hand, Du *et al.* (2016) apply the longest common subsequence (LCS) algorithm to analyze unstructured Logs and extract message types and parameters in a structured way.

Xu *et al.* (2009) highlight the complexity of the unstructured nature of logs and the difficulty of filtering out relevant data for analysis. In addition, the requirement for real-time processing to identify faults and respond to security incidents adds complexity to the challenge. Among the specific challenges of analyzing bug report logs are the variety of devices and operating system versions, the presence of confidential information, the complexity of the system, and the lack of standardization, which make it challenging to automate the parsing process.

## 2.3 Log parsing techniques

Log parsing methods easily converse with traditional parsing methods, sharing strategies, and difficulties in rationalizing the information, in this case, contained in event logs. While in the Top-down approach, parsers seek to segment the complete information and establish its priority relationships correctly, the Bottom-up approach starts from the most straightforward types and relationships between data fragments to structure the information Sharma *et al.* [2013]; Jiang *et al.* [2024].

Approaches such as clustering mining patterns in Event Logs present in the Parsing Vaarandi and Pihelgas [2015]; Vaarandi [2003] and Token Frequency Verification Nagappan and Vouk [2010] strategies can be seen as Bottom-up. In contrast, methods such as Log execution abstraction Jiang *et al.* [2008] and the accumulation of formats derived from

a generic log Sedki *et al.* [2022] have Top-down characteristics.

## 2.4 Log parsing evaluation metrics

Zhu et al. (2019) evaluate Log parses regarding accuracy, robustness, efficiency, and quantitative metrics. The metrics can be described as follows:

- **Accuracy:** the Log parser's ability to separate the constant and variable parts of the Log.
- **Robustness:** the Log parser's ability to achieve consistent accuracy when working with Logs from different systems or sizes. In this work, robustness is essential since the bug report comprises Logs from different system services, which have different structures and content patterns.
- **Efficiency:** speed of the Log parser to carry out processing. Efficiency was explicitly evaluated for each of the Android bug report log blocks.
- **Parsing time:** This corresponds to the time taken to process the input, extract the templates, and process the instances, resulting in generating the file with the structured data.
- **Quantity of Templates:** The number of templates reflects the complexity of a logging system; in general, the more templates, the more complex the logs. In evaluating Log Parsers, templates can be used to check whether a model can cope with the variability of a given system's logs.
- **F1-Score:** Considering that each field in the Log can contain parts referring to system variables and constants, the F1-score metric is obtained using the formula  $F1 = \frac{2 * Precision * Recall}{Precision + Recall}$  based on the calculation of Precision:  $Precision = \frac{TrueVariable}{TrueVariable + FalseVariable}$  and Recall:  $Recall = \frac{TrueVariable}{TrueVariable + FalseConstant}$ . For the parser, a variable means the changeable part of a log, such as displayed values, custom messages, while a constant refers to immutable system flags, variable names and similars.

## 2.5 Android device log structure

In Android, logs can be generated by the operating system, applications, and system services. The main tool for viewing and collecting these logs is logcat, part of the Android Debug Bridge (ADB). Android organizes the logs into different buffers, such as main, system, radio, and events, to categorize the information according to its source and type. By default, logs are stored in a circular buffer on the device. However, to preserve older or more relevant logs, developers can implement solutions that save them in files or send them to remote servers, allowing for more detailed analysis later.

Android's logging system consists of circular buffers maintained by a system process called Logd. It assists in developing new applications and keeps track of information such as system events and running application error reports. Logs are stored in eight memory buffers: *main*, *radio*, *events*, *crash*, *security*, *stats*, *kernel* and *system*. Applications can

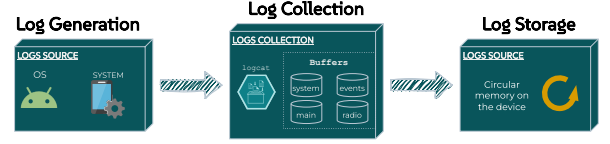


Figure 2. Android Log Collect and Storage Pipeline.

only add log messages to the *main* buffer, while the remaining buffers are reserved for system logs Cheng *et al.* [2021]. The set of available buffers is fixed and defined by the system. Each of the buffers is described in Table 2:

In Android, bug reports include data from *dumpsys*, *dumpstate* and *Logcat*, which are organized into separate sections with system-specific information.

Logcat is a log that contains information about system events and operations. It is divided into two main parts: *system* and *main*. The system is reserved for longer histories than the main part, which covers the rest of the information. Each Logcat log line starts with a timestamp, followed by a user identifier (UID), process identifier (PID), thread identifier (TID), and the Log level. Log levels can be:

- **Verbose (V):** Detailed messages.
- **Debug (D):** Debug messages.
- **Information (I):** Information messages.
- **Warning (W):** Warning messages.
- **Error (E):** Error messages.
- **Fatal (F):** Fatal messages.
- **Silent (S):** Silent messages.

Android's logging system uses several buffers in addition to the standard circular one, allowing it to store different log messages. When using Logcat, you can select only the alternative buffer, allowing you to access logs stored in specific auxiliary buffers. You can view each alternative buffer: Main, System, Crash, Event, and Radio.

```

----- SYSTEM LOG (logcat -v threadtime -v printable -v uid -d *v) -----
----- beginning of main
09-11 19:13:10.073 1000 575 575 W Linker : Warning: failed to find generated linker configuration from "/linkerconfig/ld.config.txt"
09-11 19:13:10.079 1000 576 576 W Linker : Warning: failed to find generated linker configuration from "/linkerconfig/ld.config.txt"
09-11 19:13:10.111 1000 576 576 I hservice: getDeviceManifest: Reading VINTF information.
09-11 19:13:10.129 1000 577 577 I SELinux : SELinux: loaded vndservice context from
09-11 19:13:10.129 1000 577 577 I SELinux : /vendor/etc/selinux/vndservice_context
09-11 19:13:10.137 1000 576 576 I hservice: getDeviceManifest: Reading VINTF information.

```

Figure 3. Logcat System Log Format.

## 3 Android raw log parsing

As presented in Section 2, the Android Operating System's logging routine gathers non-volatile information from various log buffers from each service running on the device. The artifact resulting from this operation is 1 or a group of text files (depending on the manufacturer) containing the different logs separated or not by text headers (Figure 3).

The fact that several logging patterns (which we call blocks) are mixed in a single file makes it difficult to separate each one in an automated way, as it requires prior knowledge of all the blocks present and their formation rules. Another difficulty in dealing with this untreated information is the presence of records with anomalies (incomplete fields, mixed records, illegible characters, instances outside

**Table 1.** Selected models for the Benchmark Zhu *et al.* [2019].

Technique	Log Parser	Reference	Description
Frequent pattern mining	<b>SLCT</b>	Vaarandi [2003]	Each cluster corresponds to a certain line pattern that occurs frequently enough.
	<b>LFA</b>	Nagappan and Vouk [2010]	In LFA, token frequencies are compared within each log message, rather than across all log messages.
	<b>LogCluster</b>	Vaarandi and Pihelgas [2015]	LogCluster is a Perl-based tool for clustering log files and mining line patterns from log files.
	<b>Logram</b>	Dai <i>et al.</i> [2020]	Logram is an automated log analysis technique that uses n-gram dictionaries to obtain an efficient log analysis.
Heuristics	<b>AEL</b>	Jiang <i>et al.</i> [2008]	AEL (Abstracting Execution Logs) is one of the most prominent log analysis approaches, comprising four stages: anonymize, tokenize, categorize, and reconcile.
LCS	<b>SHISO</b>	Mizutani [2013]	It is a method for mining registry formats and retrieving retrieval of record types and parameters online.
Parsing Tree	<b>Drain</b>	He <i>et al.</i> [2017b]	Uses an analysis tree with a fixed depth to guide the process of searching for groups of records.
	<b>Brain</b>	Yu <i>et al.</i> [2023]	It creates initial groups according to the longest common pattern. A bidirectional tree is then used to hierarchically complement the constants with the longest common pattern.
Machine Learning	<b>NuLog</b>	Nedelkoski <i>et al.</i> [2021]	It uses a self-supervised learning model and formulates the analysis task as masked language modeling (MLM).
Template-based	<b>ULP</b>	Sedki <i>et al.</i> [2022]	ULP (Universal Log Parsing) is a highly accurate log parsing tool with the ability to extract templates from unstructured log data. ULP learns from sample log data to recognize future log events.

the block's scope) and the immense variability of a log of the same block.

This variability is caused by the large number of services, with different information presentation protocols present on the device sharing the same record. Figure 4 shows how logs from the same block can differ.

**Figure 4.** Example of Variability in Logs of same Block.

This work investigates how the Log Parsing models in the literature behave in a scenario where the only treatment of the logs is the segmentation of the Blocks.

### 3.1 Related works

Table 3 provides a comparative overview of evaluation studies on log parsing techniques, arranged chronologically by publication year. It highlights key aspects of each work—including the bibliographic reference, objectives, adopted approaches, primary methods evaluated, metrics used, and key findings—thereby offering a concise summary of the evolution of log parsing methods. This comparative synthesis underscores the similarities and differences among the studies and draws attention to common challenges, such as scalability and effectiveness in handling modern, heterogeneous log data. As such, the table is a valuable resource for researchers and practitioners in selecting and refining strategies for large-scale log data analysis.

**Table 2.** Description of main Log Buffers in Android.

Buffer ID	ID	Description
LOG_ID_MAIN	0	Main log buffer, accessible by applications. Logs general messages, including errors, warnings, information, and debug messages from apps and the system.
LOG_ID_RADIO	1	Specialized in communication logs, capturing cellular telephony, mobile data, and network interactions.
LOG_ID_EVENTS	2	Stores logs of specific system events, recording user actions, system activities, and important state changes.
LOG_ID_SYSTEM	3	Focuses on operating system-level logs, including system services, running processes, and configuration.
LOG_ID_CRASH	4	Logs detailed information on app and system crashes, including stack traces and error reports.
LOG_ID_STATS	5	Collects performance data and system statistics, such as CPU, memory, and battery usage, aiding in performance analysis.
LOG_ID_SECURITY	6	Dedicated to security logs, documenting access attempts, security violations, and other security-related issues.
LOG_ID_KERNEL	7	Captures Linux kernel logs, including hardware management, memory, and device drivers.

He *et al.* [2016] evaluates four log parsing methods (SLCT, IPLoM, LKE, and LogSig) using five large datasets from various systems, including supercomputers (BGL and HPC), distributed systems (HDFS and Zookeeper), and standalone software (Proxifier), totaling over 10 million log entries. The results show that while these methods generally achieve high accuracy (above 80%), clustering-based techniques like LKE and LogSig do not scale well for large datasets, requiring parallelization. Additionally, the study reveals that parsing errors can significantly impact log mining: a 4% error in critical events during anomaly detection led to a drastic increase in false positives and missed detections. To mitigate these issues, the study suggests adopting preprocessing techniques and best practices, such as incorporating event identifiers into logs at the time of generation.

Zhu *et al.* [2019] evaluates 13 log parsing methods on 16 datasets from various systems, including distributed systems, supercomputers, operating systems, mobile systems, server applications, and standalone software, totaling over 77 GB of logs. The study assesses these parsers' accuracy, robustness, and efficiency, highlighting that while some meth-

ods achieve high accuracy (above 90%), no single parser performs optimally across all datasets. Clustering-based approaches often struggle with efficiency, while heuristics-based methods, such as Drain, offer better accuracy and scalability. The study also presents a real-world deployment at Huawei, where automated log parsing significantly improved log management by reducing manual efforts and enhancing system monitoring. The authors provide open-source tools and benchmark datasets to facilitate further research and industrial adoption, aiming to advance automated log parsing technologies.

Jiang *et al.* [2024] conducted a comprehensive evaluation of 13 log parsing techniques applied to 16 real and diverse log datasets, such as HDFS, OpenStack, and BGL, totaling over 10 million entries. The objective was to compare the effectiveness, robustness, and scalability of these techniques, which were classified into four main groups: rule- and heuristic-based methods, clustering, specialized data structures, and machine learning/NLP. Using metrics like precision, recall, and F1-score, the results showed that Drain and Spell performed well in specific scenarios, but no technique was consistently superior across all cases. The study highlighted challenges such as sensitivity to parameters, difficulty in generalization, and scalability, emphasizing the need for improvements to address the complexity and heterogeneity of logs in real-world environments.

The Logpai/LogParser collection is a project awarded the *First IEEE Open Software Services Award*<sup>1</sup> that seeks to develop tools for automated log analysis supported by artificial intelligence. Among the tools is the collection of processed data **Loghub** Zhu *et al.* [2023], which contains logs from various ecosystems, including Android. The second tool highlighted is the collection of parsers **Logparser**, which provides a machine learning toolkit and benchmarks for automated Log analysis. The tool makes it possible to automatically extract event models from unstructured logs and convert log messages into a sequence of structured events. Logpai/Logparser enables experiments with 13 parser models from the literature; for this work, 10 different models were selected to analyze performance on raw logs from Android devices of different brands. Table 1 briefly describes the strategies used in each model and its reference in the literature.

Two artifacts are generated from the execution of Logparser. The first is the template file, which contains records of patterns identified within the logs. These templates define reusable structures that facilitate the processing of future log entries. There are different types of templates (Figure 5), and a single log entry may match more than one template. Constants are kept literal in the structure, and variable data is represented by the "\*" symbol. The second artifact is the structured data (Figure 6), the Logs processed in table form with the fields separated and identified given the input Log format.

The Loghub collection's set of Android Logs was extracted in a format similar to the Logcat block and a normalized format. Table 4 compares the performance between the

<sup>1</sup>Described in: <https://www.cse.cuhk.edu.hk/news/achievements/lyu-team-first-ieee-open-software-services-award/>



**Table 3.** Comparison of log parsing evaluation studies.

Work	An Evaluation Study on Log Parsing and Its Use in Log Mining	Towards Automated Log Parsing for Large-Scale Log Data Analysis	Tools and Benchmarks for Automated Log Parsing	Log Parsing Evaluation in the Era of Modern Software Systems	A Large-Scale Evaluation for Log Parsing Techniques: How Far Are We?
Reference	He <i>et al.</i> [2016]	He <i>et al.</i> [2017a]	Zhu <i>et al.</i> [2019]	Petrescu <i>et al.</i> [2023]	Jiang <i>et al.</i> [2024]
Objective	Compare different parsers and their impact on log mining.	Develop a parallel parser for large-scale data processing.	Provide a comprehensive evaluation of log parsers and release tools/benchmarks for automated log parsing.	Evaluate the effectiveness of log parsing techniques in modern systems.	Create a rigorous benchmark for evaluating log parsers.
Approach	Study 4 popular methods applied to 5 large datasets.	Analyze existing methods and propose the POP (Parallel Log Parser).	Evaluate 13 log parsers on 16 datasets and release an open-source toolkit.	Evaluate 14 methods using 9 public datasets and logs from a bank.	Introduce Loghub-2.0, a larger and more representative dataset.
Main Methods Evaluated	SLCT, IPLoM, LKE, LogSig.	SLCT, IPLoM, LKE, LogSig, POP.	SLCT, AEL, IPLoM, LKE, LFA, LogSig, SHISO, LogCluster, LenMa, LogMine, Spell, Drain, MoLFI.	Drain, Spell, IPLoM, LogCluster, MoLFI, NuLog, among others.	15 parsers, including statistical and machine learning-based methods.
Metrics Used	F-measure, impact on log mining (e.g., anomaly detection).	Computational efficiency, scalability, impact on log mining.	Accuracy, robustness, efficiency.	Parsing accuracy, log template accuracy, edit-distance.	Group Accuracy (GA), F1-score of Group Accuracy (FGA).
Key Findings	Accurate parsing improves log mining efficiency; clustering-based methods do not scale well.	POP improves scalability and efficiency, but parameter tuning remains a challenge.	Drain demonstrated the best overall performance; some parsers do not scale well with large logs; Loghub was released as a public benchmark.	Current parsers struggle with heterogeneous logs; LOGCHIMERA was proposed to generate synthetic data.	Parsers show performance degradation on Loghub-2.0; many fail to process logs in a reasonable time.

```

EventId,EventTemplate,Occurrences
295c14e1,<*> (partial),1203
892993e6,<*> , 3006

```

**Figure 5.** Example of Templates from Android Logs.

Time	Pid	Tid	Level	Component	Content	EventId	EventTemplate	ParameterList
07:00:02.417	root		0	0   trusty	boot args 0x*** (295c14e1	boot args 0x*** (295c14e1		
07:00:02.417	root		0	0   trusty	gicc 0x***, gicd c	gicc 0x***, gicd c	8,93E+11	gicc 0x***, gicd c

**Figure 6.** Example of an Android Logs structured data table.

selected models and the base extracted from Loghub. NuLog had the best accuracy among the models, and SHISO had the worst.

**Table 4.** Results for the Loghub reference log base Zhu *et al.* [2023].

Model	AEL	Brain	Drain	LFA	ULP
F1	0,9404	0,9968	0,9959	0,9220	0.9714
Accuracy	0,6815	0,9605	0,9110	0,6160	0.8380
Model	LogCluster	Logram	NuLog	SHISO	SLCT
F1	0,9840	0,9750	0,9999	0,8437	0.9836
Accuracy	0,7975	0,7945	0,9945	0,5850	0.8815

## 4 Design of experiments

This section presents how the comparative analysis of the Block Parsers extracted from the log compilation files was structured. The section presents the steps for generating the dataset, separating the blocks, and implementing the benchmarking procedures. Figure 7 shows how these parts form the entire structure of this research.

### 4.1 Dataset description

The dataset was built from logs extracted from different models of Android devices. The data was collected manually on each Android device selected for the study. To do this, the device's interface was used in its developer options, which makes it possible to generate bug reports containing detailed information about the device's performance and operation. Each device was identified with a specific designation, and statistical information was recorded. The main aspects include:

- **Device Model:** Indicates the specific model of the Android device used to collect the data.
- **Collection:** Indicates the total number of collections for each device model.
- **Lines:** Represents the total number of lines in the files collected in each collection.
- **File Size:** Refers to the total size of the data files generated in all collections for each device model. This size is expressed in megabytes (MB) and represents the storage space occupied by the captured data.

**Table 5.** Device Collection Results.

Device Model	Collections	Line	Data Size(MB)
Samsung Galaxy A22	1	1188315	108.50
Samsung Galaxy A70	1	1361706	113.65
Motorola Moto G200	4	3685713	244.36
Samsung Galaxy S23	4	7757897	712.15
Google Pixel Pro 6	5	2131908	161.26
Motorola Moto G71	8	6206584	494.90
Motorola Moto G30	11	7638586	589.43
<b>Total</b>	<b>34</b>	<b>29970709</b>	<b>2424.25</b>

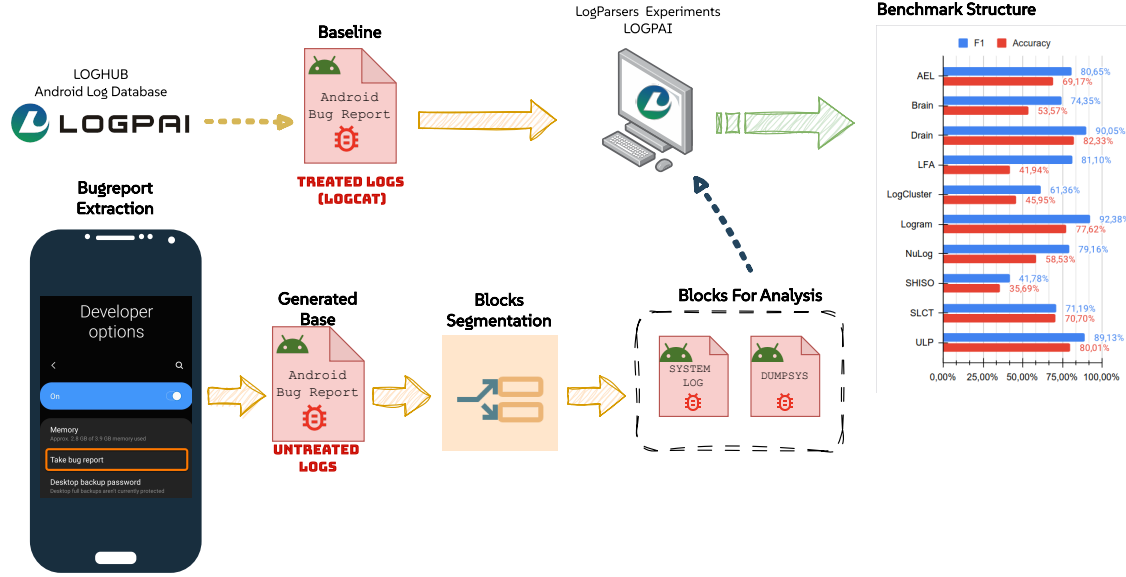


Figure 7. Experiments Pipeline.

## 4.2 Block segmentation

A series of regular expressions and searches in specific regions of the log files were carried out to separate the log blocks. The regular expressions were created to simulate the LOGPAI Collection's template system to deal with the block variations and serve as auxiliary input to the parsers. This cataloging was based on the information found in Android Code Search<sup>2</sup>. Figure 8 shows an example of the regular expression used in the extraction, and Table 6 lists the blocks cataloged and the number of instances.

```
# Abre o arquivo para leitura
file = open("file-pixel.txt", "r", encoding="utf_8")

# Define o padrão da expressão regular para os logs do Dmesg
pattern = re.compile("(^(<\\d+>)?\\[\\s*(\\d+\\.\\.\\d{6})\\]\\ (.*?)$")
```

Figure 8. Block extraction example.

```
01-09 12:17:48.524 - 10182 - REL MotoDisplayWakeLock
01-09 12:17:48.526 - 10182 - ACQ MotoDisplayWakeLock (partial)
01-09 12:17:48.527 - 10182 - REL MotoDisplayWakeLock
```

Figure 9. Dumpsys - WakeLock Example.

```
08-23 09:19:56.331 root 20270 20270 D cnss : Runtime resume start
08-23 09:19:56.331 root 20270 20270 D cnss : Resuming PCI link
08-23 09:19:56.334 logd 1034 1034 I logd : logdr: UID=0 GID=0 P
```

Figure 10. Logcat - Systemlog Example.

In this work, we investigate 5 specific blocks: Logcat - System log and Dumpsys - WakeLock and 3 Minor Wifi Dumps. Dumpsys is a logging tool that inspects hardware diagnostics data and runs device services. typically the data is presented in the format: *DUMP OF SERVICE [service]*. WakeLocks allows an application or service to keep the device active, preventing it from entering power-saving modes

Table 6. Cataloged Blocks and number of instances

Block	In- stances	Match Pattern
System Log	332525	d{2}:\textbackslash d{2}:\textbackslash d{3}" + "(?:\textbackslash s+[0-9A-Za-z ])?\textbackslash s+(\textbackslash d+)\ textbackslash s+(\textbackslash d+)\ textbackslash s+([A-Z])\textbackslash s+ s+" "(+?)\textbackslash s*:(.*)\$
Dumpsy sem_wifi	26159	sem_wifi:.*?(?=DUMP OF SERVICE \$)
Dumpsy wifi	27018	wifi:.*?(?=DUMP OF SERVICE \$)
Kernel Log	109444	\textbackslash d{6})\textbackslash (.*?)\$
Dsy. wifiscanner	107018	wifiscanner:.*?(?=DUMP OF SERVICE \$)
Wake Lock	24474	d{2}:\textbackslash d{2}:\textbackslash d{3} - \textbackslash d+ - (REL ACQ)

<sup>2</sup>Available at: <https://source.android.com>

such as sleep. Logcat is a log that contains information about system events and operations. It is divided into two main parts: System and Main. Typically, the data is presented in the format: *[timestamp UID PID TID log-level] [message]*.

### 4.3 Benchmark structure

As previously presented, the Log Parsers collection Logpai is the object of analysis in this research. As shown in Figure 7, the experiment routine begins by extracting the blocks previously described, then the extracted blocks are grouped into two .Log files per device, one file for the WakeLock block and another for the System Log/LogCat block. The generated files are used as input to the benchmark environment. The benchmark environment is contained in a Docker Container with the following characteristics to maintain isonomy in the measurement of parsing time between tests:

- **Container resources:** 8 Gigabytes of RAM and 8 CPU threads.
- **Operating System:** Nix OS.
- **Hardware Features:** Ryzen 5950X(16c/32t), 32 Gb RAM 4000Mhz.

The benchmark was written in Python, using the *Logparser3* library and files from the Logpai collection repository<sup>3</sup>. The metrics measured are: **Parsing time, Number of templates extracted, F1-Score and Accuracy**. For accuracy and F1-Score, the parsing template for each entry was used. The template was generated by running Parser Drain to check and modify incorrect instances based on the training rules for each block.

The *Logparser3* tool library for the Logpai collection has a unified input processing routine for all models and was unable to fully process all instances of the raw blocks, presenting the *skip line* exception treatment. Table 7 shows the relationship between the number of instances processed and the totals. Since not all instances were processed completely, the missing instances are included in the calculation of accuracy and F1-score, consequently making it impossible to achieve 100% accuracy in certain instances. All instances from WakeLock were Processed. The Log Parser Drain with a set of manual templates was used to generate the True Pattern for each instance, and incorrect instances were processed manually.

**Table 7.** Comparison between input instances and processed instances.

Device	System Log	
	Instances	Processed
Google Pixel Pro 6	30437	30126
Motorola Moto G200	37789	37659
Motorola Moto G30	51973	51893
Motorola Moto G71	36975	36707
Samsung Galaxy A22	161096	161046
Samsung Galaxy S23	142555	142425

Each model has its own set of input hyperparameters for specific settings for each type of Log. To maintain isonomy in the tests, the values in Zhu et al. [2019] for Android Logs

were used. Another critical factor is the optional entry of Regex for pre-processing the input and the format, and both fields were filled in according to the training rules of each block. The WakeLock block does not have Regex because the regex obtained from its training rules did not modify the convergence of the models but only increased the execution time. Finally, the test was run 10 times for each model, and all the results presented in this paper are the average of all the runs. During the 10 runs, only the execution time varied. Considering the temperature of the machine and executions of operating system subroutines, some variation was expected.

## 5 Benchmark results

The SystemLog block contains the highest number of occurrences 7, and the blocks for devices A22 and S23 have up to four times as many occurrences as the others. The blocks' size directly impacted the number of models generated and the analysis time 8. The SHISO model and the records from device S23 had the worst analysis time, with an average of approximately one hour of execution. It is important to note that the occurrence with the most significant size did not result in the worst analysis time, which suggests that other factors in the structure of a block can affect the analyzer's performance. For the SystemLog/Logcat blocks, the SHISO Model had the longest runs, while LFA and LogCluster had the best average performance among the devices. Analysis times vary considerably between models and even between occurrences of the same model, allowing several different analyzer models to be benchmarked during a single run of the longest model. This time discrepancy negatively affects the efficiency and robustness analysis of the SHISO Model and, to a lesser extent, the Brain, Logram, and ULP models.

WakeLock blocks have fewer occurrences and less complex structures than SystemLog, which is directly reflected in the execution times. The longest time recorded is approximately 1 hour and 50 minutes, attributed to the Logram model. For this block, although the times do not differ as significantly as in the previous block in absolute terms, it is still possible to see that, in proportion, the difference can reach more than 60 times between executions for the same occurrence, as illustrated in Table 9, where the fastest execution occurs on the G200 device (00:00.72 in the Brain model) and the slowest (01:00.11 in the Logram model).

Figure 11 shows the average F1-Score and Accuracy of the SystemLog blocks per parsing model. The main purpose of this article is to analyze the behavior of the main Log Parsers in the literature when receiving untreated Logs as input. When we compare the results of Figure 11 with Table 4, we can see a drop in the performance of all the models, even though they have the same input block, only differing in the treatment of the input. Unlike the Loghub collection, the Logs from the dataset generated do not undergo any normalization or extraction of problematic Logs, in addition to the slight variations in formats between Logs from different devices. These factors increase the challenge of interpreting the models, having a negative impact on their assertiveness. The models with the lowest drop (except for the SHISO

<sup>3</sup>Available at: <https://github.com/Logpai>



Table 8. Benchmark of SystemLog/LogCat Blocks

	Device	Parsing Time	Templates	F1	Accuracy
AEL	Pixel	00:33.08	3016	0.8575	0.6642
	G200	00:38.65	2042	0.8920	0.6842
	G30	00:43.67	1552	0.9985	0.8565
	G71	00:55.78	2179	0.6454	0.6580
	A22	03:40.19	7323	0.6454	0.6580
	S23	06:33.92	4359	0.8004	0.6294
Brain	Pixel	00:17.09	3341	0.6613	0.5066
	G200	00:21.88	2100	0.7427	0.5086
	G30	01:09.94	1480	0.9973	0.2902
	G71	00:19.37	2367	0.6100	0.6163
	A22	06:45.23	7288	0.6155	0.6783
	S23	14:31.00	4322	0.8341	0.6140
Drain	Pixel	00:46.63	3862	0.9789	0.8187
	G200	00:54.56	2464	0.9125	0.7811
	G30	00:58.99	1949	0.8367	0.9108
	G71	00:52.02	2773	0.8562	0.7301
	A22	03:25.69	9793	0.8845	0.8223
	S23	05:54.07	5963	0.9344	0.8768
LFA	Pixel	00:18.42	2574	0.8107	0.4283
	G200	00:22.56	1797	0.8278	0.4908
	G30	00:32.07	1320	0.9977	0.2669
	G71	00:20.40	1809	0.6383	0.4837
	A22	01:23.20	4963	0.8222	0.4967
	S23	01:33.50	3321	0.7693	0.3498
LogCluster	Pixel	00:08.42	2571	0.6101	0.4280
	G200	00:12.56	1800	0.6272	0.4914
	G30	00:14.07	1310	0.6972	0.4123
	G71	00:07.40	1792	0.6781	0.4888
	A22	01:11.32	4261	0.5001	0.4919
	S23	02:03.50	3121	0.5691	0.4443
Logram	Pixel	00:41.61	3867	0.9789	0.9187
	G200	01:00.11	2469	0.9125	0.7809
	G30	00:13.88	1949	0.8367	0.9313
	G71	00:51.18	2722	0.8562	0.6388
	A22	03:45.12	9791	0.8845	0.8130
	S23	07:50.04	5961	0.9344	0.8130
NuLog	Pixel	00:18.41	2512	0.8345	0.6211
	G200	00:21.51	1791	0.6180	0.6014
	G30	00:31.07	1345	0.9212	0.6667
	G71	00:45.07	1811	0.7651	0.8803
	A22	01:09.01	4966	0.8986	0.4123
	S23	00:58.05	3118	0.7123	0.3299
SHISO	Pixel	09:12.55	2339	0.3118	0.4810
	G200	11:26.13	1688	0.3943	0.4813
	G30	13:44.41	1293	0.3588	0.3296
	G71	04:18.12	1803	0.3593	0.2980
	A22	52:12.08	4671	0.5312	0.3298
	S23	59:51.13	3212	0.5512	0.2214
SLCT	Pixel	01:14.08	3016	0.8003	0.6441
	G200	01:08.61	2042	0.8123	0.5825
	G30	00:58.03	1552	0.9014	0.7912
	G71	04:11.15	2179	0.5751	0.6501
	A22	04:05.11	7323	0.4811	0.6642
	S23	08:43.02	4359	0.7012	0.5693
ULP	Pixel	00:45.61	3814	0.9559	0.9100
	G200	01:12.12	2398	0.9433	0.8031
	G30	01:23.48	1999	0.8162	0.9105
	G71	01:13.22	2654	0.8744	0.7091
	A22	03:59.11	9124	0.8441	0.7145
	S23	08:44.67	5671	0.9142	0.7534

parser, which already had low accuracy), and consequently the highest accuracy and F1-Score, were the Drain, Logram and ULP parsers. To a lesser extent, the AEL, Brain, NuLog and SLCT models had accuracy above 50%.

Unlike the SystemLog blocks, the WakeLock block has no treated reference. However, due to its simple construction and few variations, some models were able to get very close to 100% accuracy, and 1 F1-Score, i.e., some models were able to process almost all the input instances correctly, such as the Brain, Drain, Logram, and ULP models. The other parsers performed even worse when compared to the Logcat.

Given the above results, the parsers in this study can be evaluated on the 3 qualitative metrics described in Section 2.4. In terms of accuracy, the Drain, Logram ULP parsers have the best results, to a lesser degree, Brain obtained good

Table 9. Benchmark of Dumpsys/Wakelock blocks

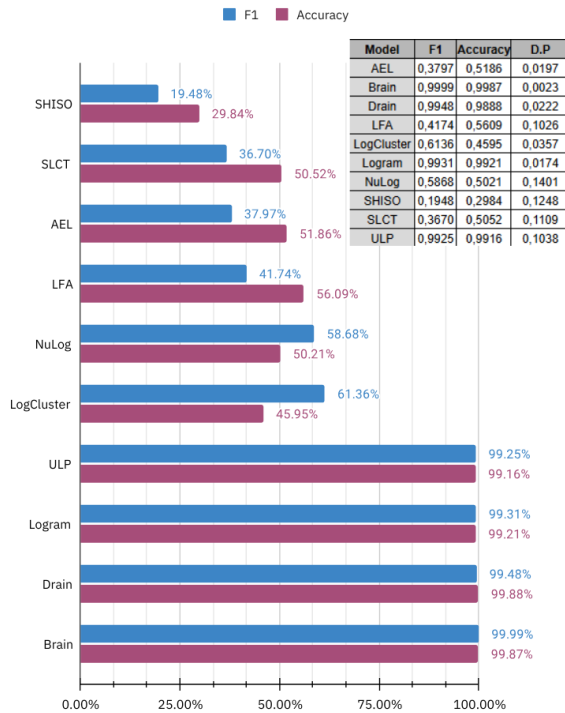
	Device	Parsing Time	Templates	F1	Accuracy
AEL	Pixel	00:01.17	132	0.2945	0.5048
	G200	00:02.30	132	0.2768	0.5135
	G30	00:01.28	94	0.3874	0.5049
	G71	00:01.00	76	0.4567	0.5117
	A22	00:01.20	80	0.6177	0.5193
	S23	00:00.88	115	0.2452	0.5571
Brain	Pixel	00:01.42	251	1	1
	G200	00:00.72	252	1	1
	G30	00:00.78	175	1	1
	G71	00:00.78	135	0.9999	0.9944
	A22	00:00.74	139	1	1
	S23	00:01.07	202	0.9999	0.9978
Drain	Pixel	00:01.66	251	1	1
	G200	00:01.50	252	1	1
	G30	00:01.71	175	0.9982	0.9891
	G71	00:01.48	133	0.9713	0.9444
	A22	00:01.58	139	1	1
	S23	00:01.29	203	0.9992	0.9991
LFA	Pixel	00:01.08	131	0.2945	0.5038
	G200	00:00.77	132	0.2768	0.5135
	G30	00:01.09	94	0.3874	0.5049
	G71	00:01.09	74	0.4567	0.5173
	A22	00:00.81	81	0.8438	0.7660
	S23	00:01.09	114	0.2452	0.5600
LogCluster	Pixel	00:01.42	131	0.6101	0.4280
	G200	00:02.56	132	0.6272	0.4914
	G30	00:01.07	98	0.6972	0.4123
	G71	00:01.40	70	0.6781	0.4888
	A22	00:03.32	85	0.5001	0.4919
	S23	00:01.50	103	0.5691	0.4443
Logram	Pixel	00:41.61	254	0.9981	0.9970
	G200	01:00.11	251	1	1
	G30	01:13.88	168	1	1
	G71	00:51.18	131	1	1
	A22	01:45.12	133	0.9613	0.9567
	S23	01:50.04	198	0.9992	0.9987
NuLog	Pixel	00:18.41	131	0.4266	0.4137
	G200	00:21.51	132	0.8438	0.7634
	G30	00:31.07	94	0.2452	0.5608
	G71	00:45.07	74	0.6105	0.4459
	A22	01:09.01	81	0.6977	0.4157
	S23	00:58.05	114	0.6971	0.4128
SHISO	Pixel	00:31.72	44	0.2325	0.1608
	G200	00:12.99	38	0.2039	0.1990
	G30	00:03.39	53	0.2484	0.0067
	G71	00:03.07	48	0.2920	0.0270
	A22	00:02.88	24	0.5481	0.2868
	S23	00:08.24	50	0.1948	0.2984
SLCT	Pixel	00:01.39	133	0.2948	0.5003
	G200	00:02.13	133	0.2700	0.4900
	G30	00:02.03	98	0.3889	0.5041
	G71	00:02.10	71	0.4661	0.4611
	A22	00:03.08	83	0.5671	0.5191
	S23	00:03.56	118	0.2152	0.5571
ULP	Pixel	00:43.11	250	0.9999	0.9900
	G200	01:23.13	250	1	1
	G30	01:01.82	156	0.9834	0.9678
	G71	01:11.14	122	1	1
	A22	01:48.56	135	1	1
	S23	01:48.04	188	0.9720	0.9921

results in the second block, although it did not have significant results in the first. The AEL, LFA, Logcluster and SLCT parsers did not perform significantly in any of the blocks. The one with the lowest accuracy was the SHISO parser, which does not perform well on Android even when using logs from the LogHub collection.

Regarding robustness, the AEL, LFA, LogCluster, NU-Log, SHISO and SLCT models could not adapt to the WakeLock block and did not perform satisfactorily. Another negative point regarding robustness is the drop in performance of the parsers for the SystemLog blocks compared to the reference results. Efficiency is a highlight of the Brain, Drain, LogCluster, NULog and ULP parsers, although only Brain, Drain manages to associate efficiency with the other metrics positively. SHISO stands out for having the worst times and



**Figure 11.** Average F1-Score and Accuracy of SystemLog/Logcat blocks by Parsing model.



**Figure 12.** Average F1-Score and Accuracy of Wakelock blocks by Parsing model.

also being associated with the worst quantitative metrics. It is worth highlighting the Logram and ULP parsers, which, although they take a long time, have satisfactory assertiveness. In general terms, all the raw logs in the more complex LogCat block fell, while the more straightforward block performed very differently, although it was the block with the

highest accuracy.

Since the Drain, Logram and ULP parsers were the best performers in both blocks, a second battery of tests was carried out with more extracted blocks: Kernel log, coming from the same logging system as the System Log block, and the Sem\_wifi, WifiScannern and Wifi\_dump blocks from the Dumpsys logging system. Figure 13 shows the execution result. It is interesting to note that the Kernel Log block, which has the same training structure as the System Log block, has similar averages to those presented in the detailed benchmark. The Wifi Dumpsys blocks vary more, with Wifiscanner having the best averages and Wifi\_dump the worst.

Furthermore, when we check the match pattern (Table 6) of the Wifi Dumpsys blocks, they appear to be simpler in structure, like the WakeLock block, but their performance is more similar to the Kernel and System Log blocks, which are more complex in structure. This behavior can be attributed to a common feature between the blocks: tables (representing mapping series of wifi networks, connection attempts, and so on) and comment blocks in some log instances. As these structures deviate entirely from the expected segmentation, parsers cannot deal with these instances and mix the information in the wrong fields.

## 6 Concluding remarks

The results show that the tools in the Logpai collection show significant variability in performance, depending on the shape of the logs provided as input. All the models performed less well when dealing with raw log bases than the results obtained with treated input bases. In the case of the WakeLock block, some analyzers obtained almost total accuracy, while others performed less well than the SystemLog block, which has a more complex structure. For the other blocks, the results were similar to the behavior of the system log block. The Kernel log block obtained very similar results to the System block, which is to be expected since they share the same training structures. The Dumpsys Wifi blocks obtained accuracy averages of over 65% for the 3 parsing models used in their benchmarks.

The Logpai collection proved an important log analysis tool, highlighting a feature common to all its analyzers: the performance variability depending on the input and the log structure. Logpai makes it easy to select the best analyzer for a given context by providing a unified user environment for models and benchmark generation tools.

However, specific logging systems like Android often contain several structures in a single log routine. The parsers must extract and process these structures individually since the models can only process one structure at a time. In addition to the need for partitioning, headers, irregular fields, formatting differences, and other anomalies in the records result in a loss of performance for the parsing models. In scenarios where the treatment and segmentation of these logs are complex, investigating new methods that can deal with Android Raw Logs directly becomes relevant.

Taking these issues into context, this research has shown the limitations and capabilities of current log parsing mechanisms, corroborating future work that can parse different

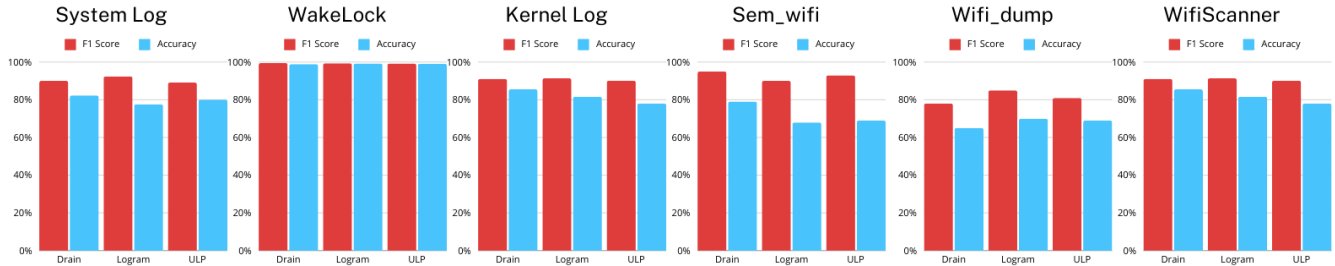


Figure 13. Benchmark results for all extracted blocks.

structures at the same time, deal with untreated data, as well as work that takes advantage of this rich source of information from the subsystems that make up the Android device ecosystem.

This article explored the serverless, cloud-to-thing continuum, addressing the growing demand for low-latency, high-bandwidth computing services in 5G and 6G networks. The proposed model takes advantage of serverless architecture to deploy computing functions on a variety of devices, including satellites and drones, in addition to traditional servers, allowing for a more flexible and distributed computing environment. Furthermore, by adopting an SDN approach and taking advantage of NDN at the data plane, the proposed model facilitates the dynamic deployment and management of computing functions across the network infrastructure. WebAssembly technology is used to implement computing functions, ensuring efficiency and compatibility between devices.

The results obtained in simulation by the serverless-cloud-to-thing proof-of-concept indicate that the deployment of microservices in the cloud-to-thing continuum orchestrated by a Deployment Agent dynamically allocates resources based on service requirements and network conditions. Real-time monitoring and reconfiguration of services ensure adaptability to changing network conditions. Moreover, the reported experience with serverless-cloud-to-thing allowed the identification of other promising research directions. One is to develop models for maintaining Forward Information Base (FIB) entries, which can be done using a name-based routing protocol. Another involves proposing an improvement of SDN reliability in the presence of intermittent connectivity that can degrade the deployment agent. One possible solution may involve developing a distributed system built as chains of microservices, each responsible for a particular control task. Interacting with the community using serverless-cloud-to-thing, the intention is to discuss directions to provide insights into an efficient computing continuum model suitable for the new 5G and 6G networks.

## Declarations

## Acknowledgements

This research was partially supported by the Coordination for the Improvement of Higher Education Personnel - Brazil (CAPES-PROEX) - Funding Code 001, the National Council for Scientific and Technological Development (CNPq), and the Amazonas State Research Support Foundation - FAPEAM - through the POS-GRAD 2024-2025 project. Also, under Brazilian Federal Law No.

8,387/1991, Motorola Mobility partially sponsored this research through the SWPERFI Research, Development, and Technological Innovation Project on intelligent software performance and through agreement No. 004/2021, signed with UFAM. The authors are part of the Algorithms, Optimization, and Computational Complexity (ALGOX) CNPq research group from the Postgraduate Program in Computer Science (PPGI) and Computer Science Bachelor at IComp/UFAM.

## Authors' Contributions

JB conducted this research work, including the theoretical background, the design of the experiments, and the execution and analysis of the benchmark. RM and GS contributed to generating the dataset, extracting blocks, and organizing related works. RB contributed to reviewing this article and supervising part of the research; RdF proposed and supervised all the research and the generation of this article.

## Competing interests

The authors declare that they have no competing interests.

## Availability of data and materials

The data generated during this study are available upon request by contacting the authors via email.

## References

- Bessa, J., Filho, R., Souza, G., Pessoa, L., Barreto, R., and Freitas, R. (2024). Análise de desempenho de log parsers da coleção logpai em dados brutos de dispositivos android. In *Anais do XXIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, pages 37–48, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/wperformance.2024.2423.
- Boase, J. and Ling, R. (2013). Measuring mobile phone use: Self-report versus log data. *Journal of Computer-Mediated Communication*, 18(4):508–519. DOI: 10.1111/jcc4.12021.
- Cheng, C. C.-C., Shi, C., Gong, N. Z., and Guan, Y. (2021). Logextractor: Extracting digital evidence from android log messages via string and taint analysis. *Forensic Science International: Digital Investigation*, 37:301193. DOI: 10.1016/j.fsidi.2021.301193.
- Dai, H., Li, H., Chen, C.-S., Shang, W., and Chen, T.-H. (2020). Logram: Efficient log parsing using  $n$  n-gram

- dictionaries. *IEEE Transactions on Software Engineering*, 48(3):879–892. DOI: 10.1109/TSE.2020.3007554.
- Dhanaraj, R. K., Ramakrishnan, V., Poongodi, M., Krishnasamy, L., Hamdi, M., Kotecha, K., and Vijayakumar, V. (2021). Random forest bagging and x-means clustered antipattern detection from sql query log for accessing secure mobile data. *Wireless Communications and Mobile Computing*, 2021:1–9. DOI: 10.1155/2021/2730246.
- He, P., Zhu, J., He, S., Li, J., and Lyu, M. R. (2016). 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)*, pages 654–661. IEEE. DOI: 10.1109/DSN.2016.66.
- He, P., Zhu, J., He, S., Li, J., and Lyu, M. R. (2017a). Towards automated log parsing for large-scale log data analysis. *IEEE Transactions on Dependable and Secure Computing*, 15(6):931–944. DOI: 10.1109/tdsc.2017.2762673.
- He, P., Zhu, J., Zheng, Z., and Lyu, M. R. (2017b). Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services (ICWS)*, pages 33–40. IEEE. DOI: 10.1109/ICWS.2017.13.
- Hwang, K.-H., Chan-Olmsted, S. M., Nam, S.-H., and Chang, B.-H. (2016). Factors affecting mobile application usage: exploring the roles of gender, age, and application types from behaviour log data. *International Journal of Mobile Communications*, 14(3):256–272. DOI: 10.1504/IJMC.2016.076285.
- Jiang, Z., Hassan, A. E., Flora, P., and Hamann, G. (2008). Abstracting execution logs to execution events for enterprise applications (short paper). pages 181–186. DOI: 10.1109/QSIC.2008.50.
- Jiang, Z., Liu, J., Huang, J., Li, Y., Huo, Y., Gu, J., Chen, Z., Zhu, J., and Lyu, M. R. (2024). A large-scale evaluation for log parsing techniques: How far are we? In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 223–234. DOI: 10.1145/3650212.3652123.
- Marin-Castro, H. M. and Tello-Leal, E. (2021). Event log preprocessing for process mining: A review. *Applied Sciences*, 11(22):10556. DOI: 10.3390/app112210556.
- Mizutani, M. (2013). Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing*, pages 595–602. IEEE. DOI: 10.1109/SCC.2013.73.
- Nagappan, M. and Vouk, M. A. (2010). Abstracting log lines to log event types for mining software system logs. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 114–117. IEEE. DOI: 10.1109/MSR.2010.5463281.
- Nedelkoski, S., Bogatinovski, J., Acker, A., Cardoso, J., and Kao, O. (2021). Self-supervised log parsing. In *Machine Learning and Knowledge Discovery in Databases: Applied Data Science Track: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part IV*, pages 122–138. Springer. DOI: 10.1007/978-3-030-67667-4\_8.
- Petrescu, S., Den Hengst, F., Uta, A., and Rellermeier, J. S. (2023). Log parsing evaluation in the era of modern software systems. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, page 379–390. IEEE. DOI: 10.1109/issre59848.2023.00019.
- Romero, C. and Ventura, S. (2007). Educational data mining: A survey from 1995 to 2005. *Expert systems with applications*, 33(1):135–146. DOI: 10.1016/j.eswa.2006.04.005.
- Sedki, I., Hamou-Lhadj, A., Mohamed, O. A., and Shehab, M. A. (2022). An effective approach for parsing large log files. *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 1–12. DOI: 10.1109/ICSME55016.2022.00009.
- Sharma, P., Malik, N., Akhtar, N., and Rohilla, H. (2013). Parsing techniques: A review. *International Journal of Advanced Research in Engineering and Applied Sciences*, 2:65–76. Available at: [https://scholar.googleusercontent.com/scholar?q=cache:B86CWHRDgwQJ:scholar.google.com/+Parsing+techniques:+A+review&hl=pt-BR&as\\_sdt=0,5](https://scholar.googleusercontent.com/scholar?q=cache:B86CWHRDgwQJ:scholar.google.com/+Parsing+techniques:+A+review&hl=pt-BR&as_sdt=0,5).
- Theys, P. P. (1999). *Log data acquisition and quality control*. Editions Technip. Book.
- Vaarandi, R. (2003). A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations Management (IPOM 2003) (IEEE Cat. No.03EX764)*, pages 119–126. DOI: 10.1109/IPOM.2003.1251233.
- Vaarandi, R. and Pihelgas, M. (2015). Logcluster-a data clustering and pattern mining algorithm for event logs. In *2015 11th International conference on network and service management (CNSM)*, pages 1–7. IEEE. DOI: 10.1109/CNSM.2015.7367331.
- Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 117–132. DOI: 10.1145/1629575.1629587.
- Yu, S., He, P., Chen, N., and Wu, Y. (2023). Brain: Log parsing with bidirectional parallel tree. *IEEE Transactions on Services Computing*, 16(5):3224–3237. DOI: 10.1109/TSC.2023.3270566.
- Zhang, Y., Xiao, Y., Chen, M., Zhang, J., and Deng, H. (2012). A survey of security visualization for computer network logs. *Security and Communication Networks*, 5(4):404–421. DOI: 10.1002/sec.324.
- Zhu, J., He, S., He, P., Liu, J., and Lyu, M. R. (2023). Loghub: A large collection of system log datasets for ai-driven log analytics. DOI: 10.1109/ISSRE59848.2023.00071.
- Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., and Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE. DOI: 10.1109/ICSE-SEIP.2019.00021.