Análise de desempenho de Log Parsers da coleção Logpai em dados brutos de dispositivos Android

João Alfredo Bessa [1], Ricardo Filho[1], Girlana Souza [1], Larissa Pessoa[1], Raimundo Barreto[1], Rosiane de Freitas[1]

[1]Instituto de Computação
Universidade Federal do Amazonas
Manaus, Brasil .

{joao.bessa,ricardo.filho,girlana.santos,lsp} @icomp.ufam.edu.br
{rosiane,rbarreto} @icomp.ufam.edu.br

Abstract. *Structuring log files for better analysis, or "Log Parsing", is crucial for extracting valuable information from records generated by software systems, contributing to the understanding of operations, and the detection of anomalies in the system. comparison of eight parsing tools and models present in the Logpai collection, namely AEL, Brain, Drain, LFA, LogCluster, Logram, NuLog, and SHISO, involving raw logs from Android Devices. The results presented demonstrate a loss of precision in models with instances without pre-processing, where existing tools present great difficulty in handling untreated logs, with consequent low performance in analyzing information from raw Android Logs.*

Resumo. *O processo de estruturar arquivos de logs para melhor análise, ou "Log Parsing", é crucial para extrair informações valiosas de registros gerados por sistemas de software, contribuindo na compreensão das operações e detecção de anomalias no sistema. Neste trabalho é realizada uma análise comparativa de oito ferramentas e modelos de parsing presentes na coleção Logpai, sendo eles, AEL, Brain, Drain, LFA, LogCluster, Logram, NuLog e SHISO, envolvendo dados brutos de Dispositivos Android. Os resultados apresentados demonstram uma perda de precisão nos modelos com instâncias sem pré-processamento, onde as ferramentas existentes apresentam grande dificuldade de manipular logs não tratados, com consequente baixa performance na análise de informações dos Logs Android brutos.*

1. Introdução

Com o avanço contínuo da tecnologia e a expansão do uso de dispositivos Android, a geração em massa de registros de sistema, ou Logs, emergiu como um componente crítico na computação móvel. A análise eficaz desses Logs é fundamental para identificar falhas precocemente, otimizar o desempenho e reforçar a segurança dos dispositivos. No entanto, desafios como a variedade nos formatos dos Logs, a amplitude das informações coletadas e o crescente volume de dados apresentam obstáculos significativos para as ferramentas analíticas atuais [Zhu et al. 2023]. Deste modo, neste artigo é reportado um estudo sobre a eficácia e eficiência de parses de Logs, ferramentas que visam organizar dados brutos em formatos estruturados para análise e monitoramentos desses registros.

Centrando-se na coleção Logpai, que proporciona uma plataforma em Python para os principais parsers. Essa coleção é avaliada em bases de dados de diferentes sistemas,

pré-processadas para uniformidade. O estudo visa comparar o desempenho de diversos parsers, focando em sua eficácia com dados brutos de dispositivos Android de fabricantes como Samsung, Motorola e Google. Os dados são coletados através do BugReport, uma ferramenta que armazena logs em dispositivos Android. Devido a limitações da ferramenta Logpai, que processa apenas uma estrutura de log por vez, foram escolhidos dois tipos de logs para análise comparativa: SystemLog/LogCat e Wake Lock. A análise combina avaliações quantitativas e qualitativas de modelos de parsing, usando benchmarks para medir capacidade de processamento e considerando acurácia, robustez e eficiência. O Artigo segue o seguinte roteiro: Referencial teórico, Coleção de Log Parsers Logpai/Logparser, Caracterização da Base de dados, Projeto de Experimentos, Análise dos Resultados e Considerações Finais.

2. Referencial Teórico

Nesta Seção são apresentados os conceitos e estruturas em que este trabalho se baseia, bem como delimita o escopo da análise comparativa de Logs.

2.1. Estrutura de Logs de dispositivos Android

O sistema de registro do Android consiste em um conjunto de buffers circulares mantido por um processo do sistema chamado Logd. Ele auxilia no desenvolvimento de novas aplicações e acompanha informações como eventos do sistema e relatórios de erros aplicativos em execução. Os registros são armazenados em oito buffers de memória: *main*, *radio*, *events*, *crash*, *security*, *stats*, *kernel* e *system*. As aplicações só podem adicionar mensagens de registro no buffer *main*, enquanto os buferrs restantes são reservados para registros do sistema [Cheng et al. 2021]. O conjunto de buffers disponíveis é fixo e definido pelo sistema, sendo os mais relevantes:

- **Buffer "main":** responsável por armazenar a maioria dos registros de aplicativos.
- **Buffer "system":** destinado a armazernar mensagens originadas do próprio Android.
- **Buffer "crash":** designado para armazenar registros de falhas.
- **Buffer "event":** refere-se ao registro de eventos de diagnóstico, ele inclui registra eventos de diagnósticos do sistema.

No Android, os relatórios de bugs abrangem dados provenientes de *dumpsys*, *dumpstate* e *Logcat*, os quais são organizados em seções distintas com informações específicas do sistema.

2.1.1. Dumpsys

O *dumpsys* é uma ferramenta executada em dispositivos Android, destinada a fornecer insights sobre os serviços do sistema em execução. Seu principal objetivo é permitir que desenvolvedores inspecionem os dados de diagnóstico gerados pelos serviços do sistema, como estatísticas de processos, consumo de CPU, uso da rede e comportamento da bateria. O resultado da execução do *dumpsys* fornece uma saída diagnóstica abrangente, revelando todos os serviços que estão em execução no dispositivo conectado. Tipicamente, a saída do *dumpsys* é apresentada no formato: *DUMP OF SERVICE [serviço]* . O *dumpsys* tipicamente abarca uma média de aproximadamente 200 conjuntos de serviços

distintos. Cada serviço contém informações específicas e, em geral, segue um formato padrão para os conjuntos de informações. Dentre os serviços estão:

- ***Batterystats***: Produz dados estatísticos sobre o uso da bateria de um dispositivo, organizados pelo ID de usuário único (UID).
- ***Package***: Oferece detalhes sobre os pacotes de aplicativos instalados no dispositivo.
- ***Procstats***: Apresenta estatísticas sobre os processos em execução no dispositivo, incluindo uso de CPU, memória e outras métricas.
- ***Wi-Fi***: Fornece informações sobre o estado e configurações da conexão Wi-Fi.

2.1.2. SystemLog/Logcat

O Logcat é um registro que contém informações sobre eventos e operações do sistema. Ele é dividido em duas partes principais: ***system*** e ***main***. O system é reservado para históricos mais longos do que a parte do main, que abrange os restantes das informações. Cada linha de registro do Logcat começa com timestamp, seguido de um identificador de usuário (UID), identificador de processo (PID), identificador de thread (TID) e o nível de Log. Os níveis de Log podem ser:

- ***Verbose (V)***: Mensagens detalhadas.
- ***Debug (D)***: Mensagens de depuração.
- ***Information (I)***: Mensagens de informação.
- ***Warning (W)***: Mensagens de avisos.
- ***Error (E)***: Mensagens de erro.
- ***Fatal (F)***: Mensagens fatais.
- ***Silent (S)***: Mensagens silenciosas.

O sistema de logs do Android usa vários buffers além do padrão circular, permitindo armazenar diferentes tipos de mensagens de log. Ao utilizar o Logcat, é possível selecionar apenas o buffer alternativo, permitindo o acesso aos registros armazenados em buffers auxiliares específicos. É possível visualizar cada um dos seguintes buffers alternativos: Main, System, Crash, Event e Radio.

```
------ SYSTEM LOG (logcat -v threadtime -v printable -v uid -d *:v -T 2023-08-23 09:19:56.000) ------
--------- beginning of kernel
08-23 09:19:56.213  logd  1034  1034 I logd    : logdr: UID=0 GID=0 PID=28766 n tail=0 logMask=4 pid=0 start=0ns deadline=0ns
08-23 09:19:56.330  root     0     0 I [    C5] [irq][0x11b328bf3c][09: 19:56.685349] qca6490: [0:I:HIF] hif_wake_interrupt_handler: wake interrupt received on irq 313
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 PID=28768 n tail=0 logMask=20 pid=0 start=0ns deadline=0ns
```

Figura 1. Formato do System Log do Logcat

2.2. Log Parsing

A fase de pré-processamento de Logs é essencial para preparar dados brutos para análises mais refinadas. Este processo inclui a filtragem de dados irrelevantes, correção de erros, estruturação dos dados e imputação de valores faltantes. As técnicas empregadas podem ser categorizadas em dois grupos principais: técnicas de transformação e técnicas de detecção e visualização. As técnicas de transformação visam modificar os Logs para corrigir, completar, filtrar, e/ou formatar os dados antes da análise, melhorando assim a qualidade dos dados. Por outro lado, as técnicas de detecção e visualização focam em identificar problemas nos dados, como estruturas de mensagens problemáticas, facilitando a compreensão dos dados e identificando áreas que necessitam de transformação.

Neste contexto, as técnicas de transformação são de particular interesse, especialmente no que se refere aos Logs do sistema operacional Android, que possuem regras específicas de formatação baseadas em seu código fonte. A importância do processo de Log Parsing é ressaltada por He et al. (2017) , que descrevem o Log Parsing como um passo inicial na mineração de Logs para identificação de padrões anormais e insights operacionais [Xu et al. 2009]. Nessa fase, as principais informações são estruturadas, como representado na Figura 2.

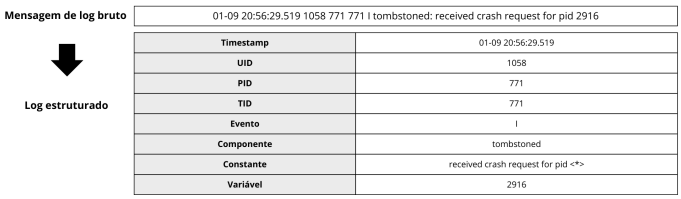


Figura 2. Ilustração de exemplo de pré-processamento realizando o parser de um Log Android.

Zhu et al. (2019) compilaram diversas técnicas de Log Parsing, realizando benchmarks para avaliar seu desempenho em detectar constantes e variáveis em Logs de diferentes sistemas [Zhu et al. 2019]. Outros trabalhos se destacaram nesses benchmarks, realizados por Loghub, por sua alta acurácia em processar Logs do buffer Main do Android [He et al. 2017a, Du and Li 2016]. He et al. (2017) utilizam uma árvore de profundidade fixa para acelerar a análise de Logs, empregando uma estrutura de árvore para categorizar mensagens de Log com base em regras pré-definidas. Du et al. (2016), por sua vez, aplicam o algoritmo de subsequência comum mais longa (LCS) para analisar Logs não estruturados e extrair tipos de mensagens e parâmetros de maneira estruturada.

Xu et al. (2009) ressaltam a complexidade da natureza não estruturada dos Logs e a dificuldade de filtrar dados relevantes para a análise. Além disso, a exigência de processamento em tempo real, a fim de identificar falhas e responder a incidentes de segurança, acrescenta uma camada adicional de complexidade ao desafio. Entre os desafios específicos da análise de Logs de bug report, destacam-se a variedade de dispositivos e versões de sistemas operacionais, a presença de informações confidenciais, a complexidade do sistema e a falta de padronização, que dificultam a automatização do processo de parsing.

2.3. Métricas de avaliação de *Parsing* de Logs

Zhu et al. (2019) avalia os Log parses em termos de acurácia, robustez, eficiência a partir de métricas quantitativas. As métricas podem ser descritas da seguinte forma:

- **Acurácia**: capacidade do Log parser em separar as partes constantes e variáveis do Log. É uma das características centrais no estudo de Logs devido ao fato da etapa afetar diretamente os resultados de etapas posteriores no processamento de Logs.
- **Robustez**: capacidade do Log parser em obter uma acurácia consistente ao trabalhar com Logs de diferentes sistemas ou tamanhos. Neste trabalho, a robustez é importante devido ao fato do relatório de bugs ser composto por Logs de diferentes serviços do sistema, que acabam por possuir padrões de estrutura e conteúdo distintos entre si.

- **Eficiência**: velocidade do Log parser para realizar o processamento. Avaliou-se a eficiência de forma específica para cada um dos blocos de Log do relatório de bugs do Android.
- **Tempo de Parsing:** Corresponde ao tempo de processamento da entrada, extração de templates e processamento das instâncias resultando na geração do arquivo com os dados estruturados.
- **Quantidade de Templates:** Refere-se à quantidade de Templates extraídos de uma entrada, geralmente um maior número de templates significa uma maior identificação de variáveis dentro dos campos do Log, porém depende do modelo separar corretamente os campos do Log. **Templates** são estruturas padronizadas que representam a forma e o conteúdo dos logs, onde as partes constantes são fixas e as variáveis são campos que mudam [Zhu et al. 2023].
- **F1-Score:** Considerando que cada campo do Log pode conter partes referentes a variáveis do sistema e contantes, a métrica F1-score é obtida através da fómula $F1 = \frac{2*Precisão*Recall}{Precisão+Recall}$ com base no cálculo da Precisão: $Precisão = \frac{VariávelVerdadeira}{VariávelVerdadeira+FalsaVariável}$ e Recall: $Recall = \frac{VariávelVerdadeira}{VariávelVerdadeira+FalsaConstante}$. Para o parser, uma variável significa a parte mutável de um log, como valores apresentados, mensagens personalizadas, enquanto uma constante de refere à flags imutáveis do sistema, nomes de variáveis e semelhantes.

3. Coleção de Log Parsers Logpai/Logparser

A coleção de Log Parsers Logpai é um projeto premiado com o *First IEEE Open Software Services Award*[1] que busca desenvolver ferramentas para análise de Logs automatizadas suportadas por inteligência artificial. Dentre as ferramentas temos a coleção de dados processados **Loghub** [Zhu et al. 2023] que contém Logs de diversos ecossistemas, dentre eles o Android. A segunda ferramenta destacada é a coleção de parsers **Logparser**, que fornece um kit de ferramentas de aprendizado de máquina e benchmarks para análise automatizada de Logs. A ferramenta possibilita a extração automática de modelos de eventos de Logs não estruturados e converter mensagens de Log em uma sequência de eventos estruturados. O Logpai/Logparser possibilita experimentos com 13 modelos de parsers da literatura, para este trabalho foi selecionado 8 modelos diferentes a fim de analisar o desempenho em Logs brutos de dispositivos Android de diferentes marcas. A Tabela **1** exibe uma breve descrição sobre as estratégias utilizadas em cada modelo e sua referência na literatura.

São gerados dois artefatos a partir da execução do Logparser. O primeiro é o arquivo de templates (Figura **3**), os templates são registros dos padrões catalogados dentro dos Logs, estruturas que podem ser reaproveitadas para processar os Logs seguintes. As constantes são mantidas de forma literal na estrutura e os dados variáveis são representados pelo símbolo '*'. O segundo artefato são os dados estruturados (Figura **4**), os Logs processados em forma de tabela com os campos separados e identificados dado o formato de Log de entrada. A partir da coleção Loghub foi extraído seu conjunto de Logs de Android, possuindo formato semelhante ao bloco de Logcat descrito na Seção **2.1.2** e formato normalizado. A Tabela **2** compara o desempenho entre os modelos selecionados e a

[1]Descrito em: `https://www.cse.cuhk.edu.hk/news/achievements/lyu-team-first-ieee-open-software-services-award/`

Tabela 1. Modelos Selecionados para a análise de desempenho[Zhu et al. 2019].

Modelo	Descrição	Referência
AEL	O AEL (Abstracting Execution Logs) é uma das abordagens mais destacadas de análise de logs, que compreende quatro etapas: anonimizar, tokenizar, categorizar e reconciliar. Em particular, na etapa de reconciliação, o algoritmo original mescla eventos que têm apenas um token diferente.	[Jiang et al. 2008]
Brain	Cria grupos iniciais de acordo com o padrão comum mais longo. Em seguida, uma árvore bidirecional é usada para complementar hierarquicamente as constantes com o padrão comum mais longo.	[Yu et al. 2023]
Drain	Emprega uma árvore de análise com profundidade fixa para orientar o processo de pesquisa de grupos de registros	[He et al. 2017b]
LFA	O LFA (Log File Abstraction) é uma abordagem de análise de log que se estende ao SLCT . No LFA, as frequências de tokens são comparadas dentro de cada mensagem de log, em vez de em todas as mensagens de log.	[Nagappan and Vouk 2010]
LogCluster	O LogCluster é uma ferramenta baseada em Perl para agrupamento de arquivos de log e padrões de linha de mineração de arquivos de log.	[Vaarandi and Pihelgas 2015]
Logram	O Logram é uma técnica automatizada de análise de logs que utiliza dicionários de n-gramas para obter uma análise eficiente de logs.	[Dai et al. 2020]
NuLog	Utiliza um modelo de aprendizado autossupervisionado e formula a tarefa de análise como modelagem de linguagem mascarada (MLM). No processo de análise, o modelo extrai resumos dos registros na forma de uma incorporação de vetor.	[Nedelkoski et al. 2021]
SHISO	É um método de mineração de formatos de registro e de recuperação de tipos e parâmetros de registro de forma on-line. Ao criar uma árvore estruturada usando os nós gerados a partir de mensagens de registro, o SHISO refina o formato do registro continuamente em tempo real.	[Mizutani 2013]

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

Figura 3. Exemplo de Templates provenientes de Logs Android.

Time	Pid	Tid	Level	Componer	Content	EventId	EventTem	ParameterList
07:00:02.417	root	0	0	I trusty	boot args 0x*** (	295c14e1	boot args 0x*** (	['0x1000 0x0']
07:00:02.417	root	0	0	I trusty	gicc 0x***, gicd (	8,93E+11	gicc 0x***, gicd (	[]

Figura 4. Exemplo de tabela de dados estruturados de Logs Android.

base extraída do Loghub. Dentre os modelos, NuLog teve a melhor acurácia e SHISO a pior.

Tabela 2. Resultados para a base de Logs de referência Loghub[Zhu et al. 2023].

Modelo	AEL	Brain	Drain	LFA	LogCluster	Logram	NuLog	SHISO
F1	0,9404	0,9968	0,9959	0,9220	0,9840	0,9750	**0,9999**	**0,8437**
Acurácia	0,6815	0,9605	0,9110	0,6160	0,7975	0,7945	**0,9945**	**0,5850**

4. Caracterização da Base de dados

A base de dados foi construída a partir de Logs extraídos de diferentes modelos de dispositivos Android. Os dados foram coletados manualmente em cada dispositivo Android selecionado para o estudo. Para isso, foi utilizada a interface do dispositivo em suas opções de desenvolvedor, que possibilita a geração de bug reports contendo informações detalhadas sobre o desempenho e funcionamento do dispositivo. Cada dispositivo foi identificado com uma designação específica, e as informações estatísticas foram registradas. Os principais aspectos incluem:

- **Modelo do Dispositivo:** Indica o modelo específico do dispositivo Android usado para a coleta dos dados.
- **Coleta:** Indica o número de total de coletas realizadas para cada modelo de dispositivo.
- **Linhas:** Representa o total de linhas nos arquivos coletados em cada coleta.
- **Tamanho do Arquivo:** Refere-se ao tamanho total dos arquivos de dados gerados em todas as coletas para cada modelo de dispositivo. Esse tamanho é expresso em

megabytes (MB) e representa o espaço ocupado no armazenamento pelos dados capturados.

Tabela 3. Resultados das Coletas nos Dispositivos.

Modelo do Dispositivo	Coletas	Linhas	Tamanho do Arquivo(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
Total	**34**	**29970709**	**2424.25**

5. Projeto de Experimentos

Previamente apresentada, a coleção de Log Parsers Logpai é o objeto de análise desta pesquisa. Como mostrado na Figura **5**, a rotina de experimento inicia na extração de dos blocos descritos na Seção **2.1**, em seguida os blocos extraídos são agrupados em dois arquivos *.Log* por dispositivo, um arquivo para o bloco de Wakelock e Outro para o bloco de System Log/LogCat. Os arquivos gerados são utilizados como entrada no ambiente de benchmark. O ambiente de benchmark é contido em um Container Docker com as seguintes características a fim de manter isonomia na aferição de tempo de parsing entre os testes :

- **Recursos do container:** 8 Gigabytes de RAM e 8 Threads de CPU.
- **Sitema Operacional:** Nix OS.
- **Recursos de Hardware:** Ryzen 5950X(16), 32 Gb RAM 4000Mhz.

O benchmark foi escrito em Python, utilizando a biblioteca *Logparser3* e os arquivos do repositório da coleção Logpai[2]. As métricas aferidas são: **Tempo de Parsing, Quantidade de templates extraídos, F1- Score e Acurácia** . Para a acurácia e F1-Score foi utilizada o gabarito de parsing de cada entrada. O gabarito foi gerado através de uma execução do Parser Drain com verificação e modificação das instâncias incorretas com base nas regras de formação de cada bloco.

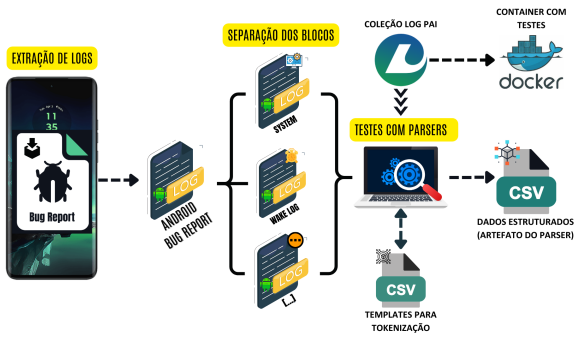


Figura 5. Estrutura geral do Benchmark.

A biblioteca de ferramentas *Logparser3* para a coleção Logpai, possui uma rotina de processamento de entrada unificada para todos os modelos e a mesma não foi capaz de

[2]Disponível em: `https://github.com/Logpai`

processar por completo todas as instâncias dos blocos brutos, apresentando o tratamento de exceção *skip line*. A Tabela 4 mostra a relação entre a quantidade de instâncias processadas e totais.Uma vez que nem todas as instâncias foram processadas por completo, as instâncias faltantes são inclusas no cálculo da acurácia e F1-score, consequentemente impossibilitando 100% de acurácia em determinadas instâncias.

Tabela 4. Comparação entre as instâncias de entrada e instâncias processadas.

Dispositivo	System Log		WakeLock	
	Instâncias Totais	Instâncias Processadas	Instâncias Totais	Instâncias Processadas
Google Pixel Pro 6	30437	30126	4041	4041
Motorola Moto G200	37789	37659	4054	4054
Motorola Moto G30	51973	51893	4153	4153
Motorola Moto G71	36975	36707	4287	4287
Samsung Galaxy A22	161096	161046	4159	4159
Samsung Galaxy S23	142555	142425	3780	3780

Cada modelo possui seu próprio conjunto de hiperparâmetros de entrada para ajustes específicos para cada tipo de Log, a fim de manter isonomia nos testes foram utilizados os valores presentes em [Zhu et al. 2019] para Logs Android. Outro fator importante é a entrada opcional de Regex para o pré-processamento da entrada e o formato, ambos campos foram preenchidos de acordo com as regras de formação de cada bloco, o bloco de WakeLock não possui Regex pois o regex obtido de suas regras de formação não modificaram a convergência dos modelos, apenas aumentaram o tempo de execução. A Tabela **5** descreve com maior detalhe todos os campos de entrada. Por fim, para cada modelo o teste foi executado 10 vezes e todos resultados apresentados neste trabalho são a média entre todas as execuções.

Tabela 5. Parâmetros de entrada por Modelo.

Modelo	Hiperparâmetros	Blocos	Regex	Formato

AEL	minEventCount: 2,
	merge_percent: 0.5,
		SystemLog	`r'(/[\w-]+)+',`
			`r'([\w-]+\.){2,}[\w-]+',`
			`r'\b(\-?\+?\d+)\b|\b0[Xx][a-fA-F\d]+\b|\b[a-fA-F\d]{4,}\b',`
				<Date><Time><Pid>
				<Tid><Level><Component>

Brain	"delimiter": [r] "theshold": 5			
Drain	"st": 0.2 "depth": 6			
LFA	Nenhum			
LogCluster	"rsupport": 1	WakeLock	Sem Regex	<Date><Time>- <Pid> - <Id><Content>
Logram	doubleThreshold: 15, triThreshold: 10,			
NuLog	"filters": "([ ])" k: 5, nr_epochs: 6, num_samples: 0,			
SHISO	maxChildNum: 4, mergeThreshold: 0.002, formatLookupThreshold: 0.3, superFormatThreshold: 0.85,			

6. Análise dos Resultados

O bloco de SystemLog é o que possui a maior quantidade de instâncias (vide Tabela **4**), ademais, os blocos dos dispositivos A22 e S23 têm até quatro vezes a quantidade de instâncias dos demais. O tamanho dos blocos refletiu-se diretamente na quantidade de templates gerados e no tempo de parsing. OSegundo a Tabela **6**, o modelo SHISO em conjunto com os Logs do dispositivo S23 é o portador do pior tempo de parsing , aproximadamente 1 hora de execução em média. Vale ressaltar que a instância com o maior tamanho não obteve o pior tempo, o que pode indicar que existem outros fatores relevantes na estrutura de um bloco que pode impactar o desempenho do parser. Para os blocos de SystemLog/Logcat, o Modelo SHISO obteve as execuções mais longas enquanto LFA e LogCluster tiveram o melhor desempenho médio entre os dispositivos. O valores de

tempo são bastante díspares entre os modelos e até entre instâncias de um mesmo modelo, de modo que é possível executar a rotina de benchmark de vários modelos de parsers diferentes durante apenas uma execução do modelo mais demorado. Essa disparidade de tempo impacta negativamente a análise de eficiência e robustez do Modelo SHISO e em menor grau dos modelos Brain, e Logram.

Tabela 6. Resultado do benchmark dos blocos de SystemLog em Tempo de Parsing e Templates Gerados (Em verde os melhores resultados, em vermelho os piores).

Dispositivo	Tempo de Parsing							
	AEL	Brain	Drain	LFA	LogCluster	Logram	NuLog	SHISO
Google Pixel Pro 6	00:33,08	00:17,09	00:46,63	00:18,42	00:08,42	00:41,61	00:18,41	09:12,55
Motorola Moto G200	00:38,65	00:21,88	00:54,56	00:22,56	00:12,56	01:00,11	00:21,51	11:26,13
Motorola Moto G30	00:43,67	01:09,94	00:58,99	00:32,07	00:14,07	01:13,88	00:31,07	13:44,41
Motorola Moto G71	00:55,78	00:19,37	00:52,02	00:20,40	**00:07,40**	00:51,18	00:45,07	04:18,12
Samsung Galaxy A22	03:40,19	06:45,23	03:25,69	01:23,20	01:11,32	03:45,12	01:09,01	52:12,08
Samsung Galaxy S23	06:33,92	14:31,00	05:54,07	01:33,50	02:03,50	07:50,04	00:58,05	59:51,13
Dispositivo	Templates Gerados							
	AEL	Brain	Drain	LFA	LogCluster	Logram	NuLog	SHISO
Google Pixel Pro 6	3016	3341	3862	2574	2571	3867	2512	2339
Motorola Moto G200	2042	2100	2464	1797	1800	2469	1791	1688
Motorola Moto G30	1552	1480	1949	1320	1310	1949	1345	1293
Motorola Moto G71	2179	2367	2773	1809	1792	2722	1811	1803
Samsung Galaxy A22	7323	7288	**9793**	4963	4261	9791	4966	4671
Samsung Galaxy S23	4359	4322	5963	3321	3121	5961	3118	3212

Os blocos de WakeLock possuem instâncias e estruturas de menores quando comparadas ao SystemLog, refletindo diretamente nos tempos de execução. O maior tempo registrado é aproximadamente 1:50 minuto e pertence ao modelo Logram. Para este bloco o tempo não é tão dispare quanto o bloco anterior em números absolutos, entretanto ainda pode-se observar que em proporção a diferença pode chegar acima de 60 vezes entre execuções para uma mesma instância, conforme observado na Tabela **7**, a execução mais rápida do dispositivo G200 (00:00,72 no modelo Brain) e a mais lenta (01:00,11 no modelo Logram). A quantidade de templates gerados para os blocos de SistemLog, de modo

Tabela 7. Resultado do benchmark dos blocos de WakeLock em Tempo de Parsing e Templates Gerados (Em verde os melhores resultados, em vermelho os piores).

Dispositivo	Tempo de Parsing							
	AEL	Brain	Drain	LFA	LogCluster	Logram	NuLog	SHISO
Google Pixel Pro 6	00:01.17	00:01.42	00:01.66	00:01.08	00:01.42	00:41.61	00:18.41	00:31.72
Motorola Moto G200	00:02.30	**00:00.72**	00:01.50	00:00.77	00:02.56	01:00.11	00:21.51	00:12.99
Motorola Moto G30	00:01.28	00:00.78	00:01.71	00:01.09	00:01.07	01:13.88	00:31.07	00:03.39
Motorola Moto G71	00:01.00	00:00.78	00:01.48	00:01.09	00:01.40	00:51.18	00:45.07	00:03.07
Samsung Galaxy A22	00:01.20	00:00.74	00:01.58	00:00.81	00:03.32	01:45.12	01:09.01	00:02.88
Samsung Galaxy S23	00:00.88	00:01.07	00:01.29	00:01.09	00:01.50	**01:50.04**	00:58.05	00:08.24
Dispositivo	Templates Gerados							
	AEL	Brain	Drain	LFA	LogCluster	Logram	NuLog	SHISO
Google Pixel Pro 6	132	251	251	131	131	**254**	131	44
Motorola Moto G200	132	252	252	132	132	251	132	38
Motorola Moto G30	94	175	175	94	98	168	94	53
Motorola Moto G71	76	135	133	74	70	131	74	48
Samsung Galaxy A22	80	139	139	81	85	133	81	**24**
Samsung Galaxy S23	115	202	203	114	103	198	114	50

geral foi proporcional ao tamanho do arquivo quando comparado entre execuções de um mesmo modelo de parsing. Não necessariamente uma maior quantidade de templates está relacionada a uma maior acurácia, entretanto, ao seguir a análise pode-se ver na Figura **6** que os modelos com as maiores quantidades de templates gerados estão entre as maiores acurácias médias. Os Logs de WakeLock possuem comportamento semelhante aos Logs de Logcat, contudo, blocos de dispositivos com menos instâncias possuíram um número maior de templates em relação a blocos maiores. Esse comportamento pode ver explicado pela capacidade de um modelo estabelecer um template que atenda mais instâncias e pela simplicidade do bloco, que possui poucas variações de uma instância para outra.

A Figura **6** apresenta a média de F1-Score e Acurácia dos blocos de SystemLog por modelo de parsing, há também o desvio padrão da acurácia descrita na tabela no canto superior no campo "D.P(ACC)". O principal intuito deste artigo é analisar o comportamento dos principais Log Parsers da literatura ao receber como entrada Logs sem nenhum tratamento. Quando comparamos o resultado da Figura **6** com a Tabela **2** pode-se observar uma queda na performance de todos os modelos, mesmo tendo o mesmo bloco de entrada, apenas diferindo no tratamento da entrada. Diferente da coleção Loghub, os Logs do dataset gerado não passam por nenhuma normalização ou extração de Logs problemáticos, além das pequenas variações nos formatos entre Logs de dispositivos distintos. Tais fatores aumentam o desafio de interpretação dos modelos, causando um impacto negativo na assertividade dos mesmos. Os modelos que tiveram a menor queda (com exceção do parser SHISO que já possuía acurácia baixa), e consequentemente, a maior acurácia e F1-Score, foram os Parsers Drain e Logram. Em menor grau os modelos AEL, Brain e NuLog tiveram precisão acima de 50%.

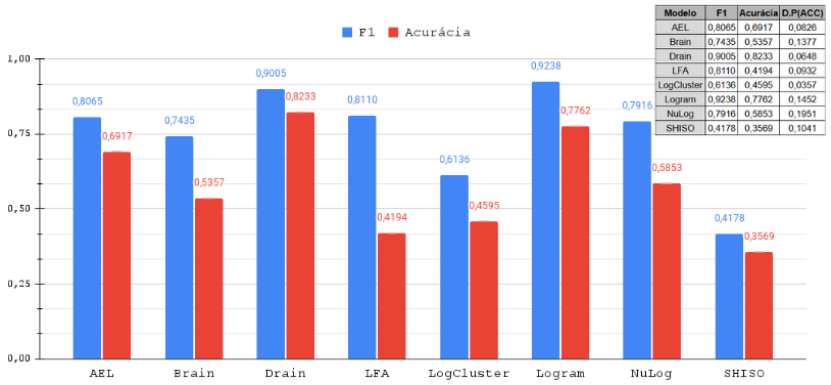
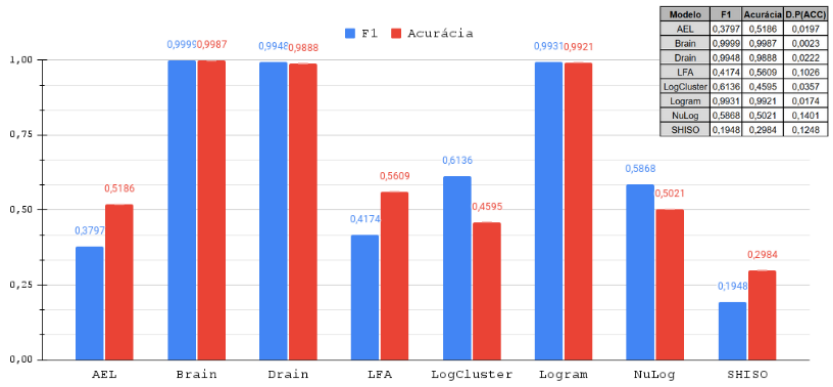


Figura 6. Média de F1-Score e Acurácia dos blocos de SystemLog/Logcat por modelo de Parsing.

Diferentemente dos blocos de SystemLog, o bloco de WakeLock não possui referência tratada, entretanto, devido à sua construção simples e com poucas variações possibilitou que alguns modelos chegassem muito próximo ao 100% de acurácia e 1 de F1-Score, ou seja, alguns modelos foram capazes de processar quase todas as instâncias de entrada de forma correta, são os modelos Brain, Drain e Logram. Os demais parsers tiveram um desempenho ainda pior quando comparados com o bloco de Logcat.

Dado os resultados supracitados, pode-se avaliar os parsers deste estudo nas 3 métricas qualitativas descritas na Seção **2.3**. Quanto à acurácia, os Parsers Drain e Logram possuem os melhores resultados, em grau menor, Brain obteve bons resultados no segundo bloco embora não tenha resultados expressivos no primeiro. O parser AEL,LFA e Logcluster não desempenharam expressivamente em um dos blocos. O que obteve a menor precisão dentre todos foi o parser SHISO, o mesmo não desempenha bem em Android mesmo utilizando os Logs da coleção LogHub. Em relação a robustez, os Modelos AEL, LFA, LogCluster, NULog e SHISO não conseguiram se adaptar ao bloco de WakeLock e não desempenharam de forma satisfatória. Outro ponto negativo à robustez é a queda de performance dos parsers para os blocos de SystemLog quando comparado aos resultados de referência. A eficiência é um destaque dos parsers Brain, Drain, LogCluster e NULog embora somente Brain e Drain consiga associar a eficiência com as outras métricas de



Modelo	F1	Acurácia	D.P(ACC)
AEL	0,3797	0,5186	0,0197
Brain	0,9999	0,9987	0,0023
Drain	0,9948	0,9888	0,0222
LFA	0,4174	0,5609	0,1026
LogCluster	0,6136	0,4595	0,0357
Logram	0,9931	0,9921	0,0174
NuLog	0,5868	0,5021	0,1401
SHISO	0,1948	0,2984	0,1248

Figura 7. Média de F1-Score e Acurácia dos blocos de Wakelock por modelo de Parsing.

forma positiva. Em destaque negativo SHISO o piores tempos e ainda associados com as piores métricas quantitativas. Vale destacar o Logram que embora demande um tempo alto, possui assertividade satisfatória.

7. Considerações Finais

Os resultados mostram que as ferramentas da coleção Logpai variam muito de desempenho conforme o formato do Log de entrada. Todos os modelos tiveram pior desempenho ao analisar logs brutos, em comparação com logs tratados. No bloco WakeLock, alguns Parsers alcançaram alta precisão, enquanto outros tiveram resultados piores em comparação com o bloco SystemLog, que possui estrutura mais complexa. A coleção Logpai se destacou como uma ferramenta importante de análise de Logs, destacando uma característica comum: a performance inconstante dependendo da entrada e da estrutura do Log. Como o Logpai fornece um ambiente unificado para os modelos e ferramentas de geração de benchmarks, isso auxilia na seleção do melhor parser para cada contexto.

Entretanto, alguns sistemas de Log, como o Android, possuem várias estruturas dentro de um mesmo registro, que precisam ser extraídas e processadas individualmente pelos parsers, pois os modelos só processam uma estrutura por vez. Outra limitação é a queda de desempenho com dados brutos, o que pode ser atribuído à dificuldade em lidar com campos faltantes ou diferenças de formatação entre os registros. Portanto, a continuidade desta pesquisa visa desenvolver uma ferramenta de classificação de logs para identificar diferentes estruturas nos registros e um parser focado na análise das estruturas de Logs Android.

8. Agradecimentos

Os autores fazem parte do grupo de pesquisa do ALGOX do CNPq (Algoritmos, Otimização e Complexidade Computacional) e do Programa de Pós-Graduação em Informática (PPGI) do IComp/UFAM. E recebeu apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES-PROEX) - Código de Financiamento 001, Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e pela Fundação de Amparo à Pesquisa do Estado do Amazonas - FAPEAM - por meio do projeto POSGRAD 2024-2025.

Referências

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.

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.

Du, M. and Li, F. (2016). Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE.

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.

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.

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.

Mizutani, M. (2013). Incremental mining of system log format. In *2013 IEEE International Conference on Services Computing*, pages 595–602. IEEE.

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.

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.

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.

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.

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.

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.

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.