# Evaluating Prompt Engineering for Event Log Parsing with Large Language Models: A Comparative Study

Siraaj Akhtar[a], Saad Khan[a,b], Simon Parkinson[a]

[a]*Department of Computer Science, Queensgate, Huddersfield, HD1 3DH, West Yorkshire, UK*
[b]*Corresponding author*

**Abstract**

The application of Large Language Models (LLMs) in event log parsing has gained traction, yet the potential of prompt engineering as a stand-alone solution remains largely underexplored. This study investigates whether prompt engineering can serve as a viable alternative to more resource-intensive methods, such as fine-tuning and in-context learning, for parsing event logs. We designed a prompt based on established principles of prompt engineering, integrating task relevance and domain-specific abbreviations that could appear in the event logs. In addition, as much of the existing literature focuses on closed-source LLMs, their reliance on external servers poses significant privacy risks. Recent advancements in open-source LLMs (particularly those that use reasoning such as Deepseek and Qwen) suggest that they may offer competitive performance. To evaluate this, we conducted a comparative analysis of several open-source and closed-source models. Our findings show that prompt engineering alone leads to a notable decline in parsing performance. However, it is important to note that the use of prompt engineering in combination with other techniques, such as RAG, can produce satisfactory results, which is also evident in the existing literature. In addition, closed-source models consistently outperformed their open-source counterparts, but as their use can not be justified due to privacy concerns, we suggested that Grok 2 1212 and Gemma 3 are suitable alternatives.

*Keywords:* Event logs, Large Language Models, Log Parsing, Prompt Engineering, Cyber Security.

## 1. Introduction

Event log parsing is an important task as it transforms raw event logs, which come from heterogeneous data sources and contain a large amount of noise, into structured logs that can be used as part of many cybersecurity analysis and monitoring processes [1]. The process of parsing event log data sources is achieved in the majority of cases by handling a known data structure to extract key parts. However, this requires the creation of a parser for each unique event log data source and has the limitation of being difficult to achieve if the event data source is unstructured or semi-structured. The absence of consistent or uniform formatting makes automated parsing a complex, error-prone, and resource-intensive task. As a result, scalability and adaptability to diverse systems are significantly hindered. Large language models (LLMs) have proven capable of performing a wide range of tasks in a manner that exceeds human performance and other intelligent approaches [2]. As a result of this, this paper intends to evaluate the use of LLMs as parsers to see if this same trend could be replicated, and therefore, progress research in the field.

This paper aims to achieve two main objectives. The first is to assess whether prompt engineering could be used for event log parsing by adapting proven techniques and criteria from the literature to match the paper's application. The second objective is to produce a comparison of LLMs, in particular, highlighting whether open-source models can compare to the performance achieved by closed-source models, such as those from the GPT and Claude series, which have been the focus of the majority of published literature.

Regarding our first objective, previous research used fine-tuning and in-context learning [3, 4, 5], both of which require datasets. These are difficult to obtain due to the justified reluctance of security teams to provide their organisation's event logs. In addition, fine-tuning requires a large amount of resources [6], and the implementation of both techniques is costly. Coupled with this, another important factor in the relevance of our approach is that the literature review highlights that research has yet to assess the impact of prompt engineering.

In terms of the second objective, early research included LLM comparisons [7]; however, currently open source alternatives are available, competing with popular closed source models such as those of the GPT series, namely DeepSeek. In addition, closed-source LLMs, such as GPT, have security issues due to data sharing with their servers when sending requests [8], making their use as event log parsers unfeasible in real-world environments. Due to the combination of these notable factors, it is important to conduct a contemporary comparison of LLMs as event log parsers with a focus on assessing whether open-source models are a viable alternative.

Our results demonstrate that prompt engineering, when used in isolation, is insufficient, as performance dropped significantly in our study compared to previous approaches. In addition to this, closed-source LLMs were shown to significantly outperform open-source models. However, critical conclusions

were made regarding the use of all models due to the large amount of time taken to complete parsing and how this relates to their deployment in real-world settings.

In this study, we make the following key contributions:

- We present a systematic evaluation of prompt engineering as a standalone approach to event log parsing with LLMs, offering an effective alternative to existing methods, such as fine-tuning and in-context learning.
- We compare and evaluate 10 open- and closed-source LLMs on 16 diverse event log datasets with common log parsing metrics to quantify performance, latency, and feasibility in real-world scenarios.
- We present practical insights into the failures of existing log parsing LLMs, such as hallucinations, structural errors, and high latency, and propose corrections for future hybrid models.

Regarding the structure of the article, we begin by conducting a comprehensive literature review in Section 2 with a focus on the following three main topics: LLM parsers that are not related to event logs, LLM-based event log parsers, and event log parsers that are not LLM-based. Following this, we present the methodology in Section 3 that we used to carry out our experiments, the LLMs that were selected, with justification for our decisions, and provide information regarding the datasets in Section 4 used for our experiments. We then present the template for the prompt in Section 5 that would be used in the experiments; this section includes the prompt itself with an analysis and justification for its structure. After this, we presented the results of our experiment and discussed our findings, their implications and future work that could build on the study in Section 6.

## 2. Literature Review

Prior to the development of large language models (LLMs), a wide range of traditional rule-based event log parsers were developed, which remain relevant due to their interpretability, efficiency, and suitability. Therefore, we begin by reviewing key studies that present such tools and highlighting their strengths and limitations. A review article [9] evaluates four existing parsers. The study identifies three key findings: current parsers demonstrate high accuracy, preprocessing using domain knowledge enhances performance, and parallelisation can scale clustering-based methods effectively. With a similar objective, another study [1] evaluates various automated log parsers in multiple datasets. This study showed the effectiveness of automated parsers, not only in datasets but also in practice at Huawei. Future work based on this article can continue to use the datasets shown in this paper to evaluate event log-related tools. A broader perspective is offered by a recent review [10], which presents findings from the evaluation of multiple parsers using 16 datasets. The review highlights inconsistencies in evaluation metrics, the benefits of pre-processing, and the challenges in understanding domain-specific terminology. It recommends adopting standardised benchmarks and improving semantic modelling of logs as priorities for future research.

Shifting focus to online log parsing, a research article [11] introduces Spell, one of the earliest online log parsers. It employs the Longest Common Subsequence (LCS) algorithm to isolate components that are altered in the log messages. The results indicate superior performance compared to offline parsers. Future work based on this article could investigate how accurately LLMs extract essential information that differs in logs. Similarly, a research article [12] addresses the limitations of offline parsing by proposing Drain, an online parser that uses a tree-based structure to classify logs. Drain achieves accuracy ranging from 0.84 to 0.99 across five datasets. However, this variability in performance highlights the importance of developing new methods that can produce consistent results across diverse log sources.

Within the domain of semantic understanding, a research article [13] presents UniParser, which aims to interpret log messages in heterogeneous systems. Using a Token Encoder and Context Encoder, the model achieves an average accuracy of 0.986. This article demonstrates the potential of NLP-based approaches and motivates further exploration of semantic log parsing. A similar study [14] introduces Log3T, a flexible transformer-based parser that can adapt to changing/different log formats. It achieved an average group accuracy of 0.94. However, further research is required into Log3T's resilience using datasets with log format drifts. Although newer methods show promise in semantic understanding, the field still lacks unified benchmarks and broad validation across diverse environments.

The domain of event log parsing, which is a key task in several applications, such as anomaly detection and system monitoring, has undergone a paradigm shift due to the LLMs. The dynamic and unstructured nature of log data presents many challenges for traditional log parsers like Drain and Spell, which mainly rely on rule-based or statistical techniques. LLMs, on the other hand, present a viable substitute by using their profound contextual knowledge and generalisation skills to parse logs with little oversight.

In a recent study [15], the authors propose a hybrid approach to log anomaly detection by integrating large language models (LLMs), specifically ChatGPT, with BERT-based architectures, called LogBERT. Their methodology uses a few-shot learning. The empirical evaluation, conducted on the HDFS and BGL datasets, shows that the base version of LogBERT is an improvement compared to the non-LLM-based methods, such as Drain, and using GPT-3.5 improved accuracy and semantic understanding of log entries. The study acknowledges the limitations of LogBERT in handling log elements, such as timestamps and identifiers that carry critical anomaly-related information. To build on this study, future research can investigate using other techniques with LLMs, which may improve results detection accuracy. Building on this, a research article [7] assesses which state-of-the-art LLMs (e.g., GPT-3.5, Claude 2.1, CodeLlama, Llama 2) are the most suitable for event log parsing. The results showed that an open-source model, CodeLlama, outperformed others in parsing accuracy, consistency, and

Edit Distance. The results are an important development, as closed-source models come with increased costs and security concerns due to data being stored on the provider's servers. The study also highlights the importance of prompt design, showing that few-shot prompting significantly improves parsing performance across all models. Due to this, future work can investigate using CodeLlama as an alternative and other promising open-source LLMs.

Another study [3] aims to analyse the effectiveness of fine-tuning open-source models for log parsing as an alternative to the use of closed-source models. Their methodology was to fine-tune Mistral-7b and compare it to GPT-4-Turbo across multiple configurations, including zero-shot, few-shot, and fine-tuned scenarios. Their results, in terms of message-level accuracy (MLA), edit distance (ED), and F1 score, show that fine-tuned Mistral-7B achieves better performance to GPT-4-Turbo in zero-shot and few-shot settings. In contrast, GPT-4-Turbo performs better in a few-shot setting. However, these results are not conclusive as GPT-4-Turbo was not fine-tuned. Future research should aim at comparing the model in the same (such as both are fine-tuned) and different scenarios (the approach for both could be in-context learning) to establish which model is superior or if one is superior in one scenario and the other in another. Moreover, the study says that expanding the dataset to include more varied and domain-specific logs would enhance the generalisability of the findings. Another recent article [16] presents LILAC that combines adaptive parsing cache and in-context learning to address several challenges of LLM-based log parsing, such as lack of task-specific specialisation, inconsistent outputs for similar inputs, and high computational overhead. The ICL-enhanced parser assists the LLM in learning how to parses LLMs to adjust to system-specific log characteristics without fine-tuning. The adaptive parsing cache reduces inefficiency and inconsistency by storing previously created templates and improving them according to new input samples. Based on empirical evaluation using the Loghub-2.0 dataset, LILAC achieves a 69.5% improvement in F1 score of template accuracy, outperforming state-of-the-art baselines across all metrics. In terms of efficiency, it is similar to the fastest syntax-based parsers like Drain, and it continues to perform well even when combined with smaller LLMs.

Focussing on recent advances which aim to improve on previous LLM-based log parsing methods, the study [5] proposes DivLog that utilises in-context learning to reduce the need for extensive fine-tuning. DivLog selects diverse log samples to construct prompts and enables LLMs to generate log templates in a training-free manner. The results from 16 public datasets showed that DivLog outperformed previous tools, with Parsing Accuracy (PA), Precision Template Accuracy (PTA), and Recall Template Accuracy (RTA) reaching 98.1%, 92.1%, and 92.9%, respectively. Future research could evaluate which LLMs achieve the best accuracy at parsing and how prompt engineering strategies influence performance. Another research article [17] proposes AdaParser, which aims to address two key limitations of current LLM-based parsers: parsing errors and poor adaptability to new/changing log data. It includes an iterative component, called "template corrector", to check if the generated template matches the original log; if it does not, it is corrected. AdaParser also maintains a list of previously parsed log templates, which enables it to learn new log formats without retraining. The results showed that the model performed better than its competitors, achieving a 93.8% Parsing Accuracy and an 87.9% F1 score in template accuracy. However, there was a clear reduction in results when comparing performance in zero-shot scenarios to few-shot scenarios. Future work is needed to ensure consistent performance with both zero- and few-shot learning in the absence of historical data.

A research article [18] proposes LogGenius. In real-world scenarios, it is not feasible to obtain labelled logs due to their sheer volume and heterogeneity. LogGenius addresses these issues by using zero-shot prompts with LLMs to generate diverse log samples and applying unsupervised parsers (like Spell or Drain), enabling scalable and adaptable log parsing without needing labelled data, rules, or manual intervention. Due to this, LogGenius achieved an improvement of up to 100% in parsing accuracy for previously unseen logs. This study demonstrates the potential of refining zero-shot based solution that can match or exceed the performance of few-shot based solutions. Similarly, another research study introduces YALP [19], which unlike other approaches, does not require labelled data for fine-tuning or in-context learning. This makes YALP suitable for online log parsing. YALP combines semantic analysis (using ChatGPT) with syntactic analysis (using the Longest Common Subsequence) to perform log parsing in a zero-shot setting. Based on 16 public datasets, their results show group accuracy F1 score as 0.92 and the parsing accuracy as 0.72. Future work can investigate how to improve the syntactic analysis to further boost parsing precision. Another important technique is RAG (Retrieval Augmented Generation). A study [20] proposes a RAG-based model to address the high computational costs and limited generalisability associated with fine-tuning. They combine RAG with similarity search to find contextually relevant templates from a continuously updated template database based on the log received, and few-shot learning to teach the model how to perform the task. The results demonstrate that increasing size of the template database improves both grouping and parsing accuracy, while reducing the number of LLM calls by 93%. Future work can investigate prompt engineering to further improve results and establish a prompt structure to follow to ensure consistent and scalable results.

Another tool LogBatcher [21] attempts to reduce the high computational costs and extensive data requirements in LLM-based log parsers. Instead of parsing logs individually, LogBatcher clusters and parses them in batches. This technique also enables the LogBatcher to discover hidden similarities and differences among logs, thereby enhancing parsing accuracy without needing labelled log samples. The authors have also included a caching mechanism to avoid redundant queries. The results of 16 public datasets show that the method can significantly reduce costs while reducing token consumption by up to 44%. Both this study and others use well-known event log datasets that are limited and may differ from real-world scenarios, which could be addressed in future work. Another research study [22] presents LUNAR, a tool developed to provide an

alternative to in-context learning. It removes the need for labelled data, which is difficult to acquire in the case of event logs. LUNAR selects logs that have similar structures but differ in parameter values. After that, it constructs prompts to identify logs' dynamic components and extracts templates without any labelled supervision. The results from 14 large-scale datasets showed that the model outperforms unsupervised baselines and provides better scalability. Future work can improve the log selection method to enhance performance and real-time deployment strategies to increase its applicability.

Another research article [23] proposes SelfLog to address efficiency concerns of LLM-based parsers and aims to reduce the number of times the LLM is invoked. SelfLog groups similar logs using 1) LLM, which learns from historical data and 2) an N-Gram-based grouper and log hitter, which clusters logs and caches parsed templates to minimise redundant LLM calls and enhance parsing performance. The results show that SelfLog achieved better accuracy than other LLM and non-LLM parsers, including DivLog. A limitation was that, although a large number of datasets were used, they may not contain logs that are representative of real-world scenarios that occur in the day-to-day event logs of organisations. Thus, future work can address this by evaluating parsers such as SelfLog using datasets that are comprised of real-world scenarios. A similar tool called, Hierarchical Embeddings-based Log Parsing (HELP), is presented in a research article [24], aiming to reduce costs. Their methodology is first to convert logs into vectors, clusters similar logs using Approximate Nearest Neighbor (ANN) and perform few-shot and Chain-of-Thought prompting to LLMs to extract log templates from new clusters. The results from 14 public datasets showed that HELP outperformed all online parsers and achieved performance similar to the best offline parser (LILAC). Future work can evaluate more open-source LLMs to host the tool, as closed-source LLMs, like Claude-3.5-Sonnet used in this paper, pose privacy concerns.

Enhancing LLM-based log parsing through architectural innovations and prompt engineering has also been the focus of recent efforts. LogBabylon [25] is a unified framework that combines RAG and LLMs to create insights that are easily understandable by humans. LogBabylon reduces manual intervention while increasing template extraction accuracy using in-context learning and variable-aware prompting. It is appropriate for large-scale deployments due to its dynamic prefix parse tree and clustering mechanism, which allow scalable parsing with fewer LLM calls. Another article presents LogRules [26], which is a lightweight framework that uses log-specific rules to reduce LLM hallucinations. LogRules improves the reasoning abilities of smaller LLMs (such as 7B–8B models) through a three-stage process: creating a rule database through induction, using contrastive preference optimisation to apply the rules, and rule-guided deduction on new data through prompting. This technique achieved similar performance with significantly lower inference costs than case-based prompting, demonstrating superior generalisation and robustness across a variety of log datasets. Another article presents Large Language Model–Template Detection (LLM-TD) [27], an unsupervised LLM-based template detection technique designed for unstructured security event logs. LLM-TD begins by grouping log messages based on the application. After that, it processes them in batches to extract multiple templates per query using in-context learning with small local LLMs, which are then validated and merged. Unmatched messages and duplicate templates are identified and removed to support further refinement. LLM-TD is computationally efficient and avoids assumptions about log structure (such as fixed token counts). Results from real-world syslog datasets shows strong performance, establishing the feasibility of preserving privacy, making on-premise LLM deployment for log analysis possible.

## 3. Methodology

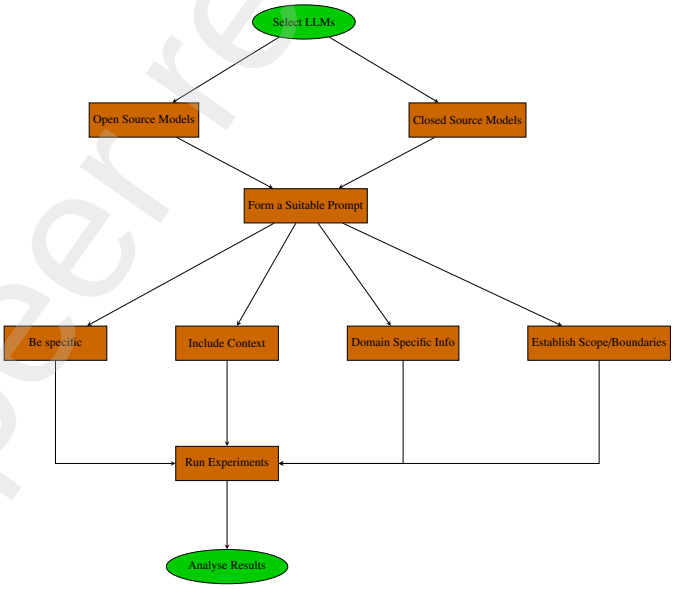


Figure 1: Flowchart for Selecting and Using Large Language Models (LLMs)

In Figure 1, we provide a concise overview of our proposed workflow to parse event logs, and expand upon it in the following sections.

### *3.1. Selection of LLMs*

Although using LLMs for event log parsing is a relatively new research field, researchers have published many articles on the subject, as evident in Section 2. Early work included comparisons of various LLMs, but many LLMs have been developed since then. A common limitation in current research on using LLMs for event log parsing or analysis is the reliance on closed-source models, such as those from the GPT series, which raises significant privacy concerns due to the sensitive nature of event log data. At the time of the previous comparative research, most open-source LLMs performed worse compared to closed-source models. However, recently, models such as Deepseek have been developed and have been shown to perform in a way that is equal to, and sometimes better than, closed-source models on other tasks [28, 29, 30, 31]. As a result

of this, we decided that it was important to conduct a comparison to evaluate the current landscape of LLMs to address the limitations identified in previous research.

In this research, we select open-source as well as popular closed-source models to compare against the open-source models that we had selected. This ensured that our experiments provided us with data that could be used to establish the best-performing LLM and to determine how the open-source models compare with the closed-source models. Furthermore, we could see if certain models performed better in certain scenarios, for example, some could perform better when parsing Windows event logs, whereas others could perform better on Linux.

When we selected LLMs for the experiments in this paper, the first consideration was that all models needed to be open source. In our previous paper, [32], we conducted a survey on the use of LLMs to perform event log analysis tasks, one of the most common limitations in the literature was that previous research focused on the use of closed-source LLMs, namely from the ChatGPT series. Closed-source LLMs require interacting with the provider and sending data to their servers, as event logs are sensitive data, which is a security concern. As a result of this, open-source models were a major consideration when choosing LLMs for our experiment.

Secondly, we chose models with more than one billion parameters. This was because the larger the number of parameters in a model, the higher the output quality. In addition, these models usually retain context longer and have better reasoning abilities [33]. These factors are important and motivated previous researchers to choose closed-source models such as GPT and Claude. However, recently open-source models, such as DeepSeek, have been produced, which have a large number of parameters and can produce results that are equal to and sometimes outperform GPT [34]. This is a significant advantage for LLM event log analysis because it provides open-source alternatives without sacrificing quality.

Finally, in relation to what was mentioned at the end of the previous paragraph, we chose models that perform well on benchmarks. Benchmarks are tests that consist of a variety of tasks, and they aim to assess the ability of models to perform these tasks. The higher scores achieved by LLM on the benchmarks are used to determine which models are better than others [35].

#### *3.1.1. Open-source models*

To begin, we chose DeepSeek: R1, which is an open-source model and has 671 billion parameters. This model has recently attracted a lot of attention as it is shown to produce outputs that rival those produced by closed-source models, namely GPT and Claude [34]. Compared to all of the models that we chose, this has a significantly higher number of parameters. This is important as our experiment could highlight whether models that consist of a larger number of parameters perform better or worse than those that contain fewer parameters.

Secondly, we chose Qwen: QwQ, which consists of 32 billion parameters. Although this model has a significantly lower number of parameters, compared to DeepSeek: R1, some research has shown that it can outperform larger models [36]. This makes it essential to evaluate its performance in other domains, such as event log analysis in our case, to ensure we are using the most effective model for our experiments.

We also chose Mistral: Pixtral Large 2411, which has 124 billion parameters. Mistral is one of the leading open-source LLMs series. We chose this variant of the model because it had the most parameters, and we also aimed to assess how well it can retain context and parse the event logs provided to it.

Another leading open-source provider is Meta, known for its Llama series of models. For our experiment, we chose the Llama 3.3 70B Instruct, which has 70 billion parameters. Although this model is not the largest in the series, we chose it due to being the most recent iteration of the model, and it fits within our criteria of being above a billion parameters. Choosing this model for our experiments would allow us to highlight how smaller LLMs can compare to larger LLMs. If they outperform the larger models, it would suggest that, for the use case of event log parsing, they are better suited than larger models.

Similarly, Gemma 3 is a smaller model with 27 billion parameters. In addition to this, it is based on concepts similar to the closed-source Gemini, which is developed by Google. This will also allow us to conduct more comprehensive experiments, compared to if we choose another model from the series selected previously.

Finally, we chose Phi-4 (with 14 billion parameters) by Microsoft, which is the smallest model included in this comparison. Although the model is the smallest, the study [37] shows that it performs close to and at times better than other models such as GPT 4o-mini and Llama-3.3 70b Instruct. This is also a model that, to the best of our knowledge, is yet to be included in event log analysis research,

#### *3.1.2. Closed-source models*

To provide a more comprehensive comparison of the models, we decided to choose four additional models to compare against. These are well-known models, and the vast majority of them are closed-source. As a result of this, the results of our experiments can highlight whether open-source models can perform as well or better than their closed-source counterparts. The models we selected were ChatGPT-4o, Claude 3.7 Sonnet, Cohere: Command R7B and Grok 2 1212.

### *3.2. Formation of a Suitable Prompt*

Secondly, we had to decide the best method to carry out our experiments. Fine-tuning has limitations due to the cost, resources required for it to take place [4], and it is difficult to train for the task related to event log analysis due to the lack of available datasets. The publicly available datasets are commonly used for benchmarking, which raises concerns about overfitting. Since these datasets are widely known and often reused, models may inadvertently learn patterns specific to the test data rather than generalisable features. This can lead to inflated performance metrics that do not accurately reflect how the model would perform on unseen, real-world environments. In-context learning also shares this limitation, and challenges would be present in training for event log parsing, as examples would

need to be given for the various types of logs and scenarios. We have highlighted in the literature review of this paper that research on event log parsing using LLMs has yet to explore the use of prompt engineering as an alternative. Existing literature (Section 2) shows several studies that demonstrate promising results using prompt engineering for parsing tasks in various environments, suggesting its potential relevance and effectiveness for our specific context. As a result of this, we decided to focus on evaluating the use of prompt engineering in our paper and compare our results to the prior research to establish if it is a viable alternative to methods such as fine-tuning and in-context learning.

To ensure that our prompts were of high quality, we established that they needed to possess a set of characteristics. We will mention four of the main qualities with a brief description, and then expand upon them in more detail in the relevant sections of the paper. The first is that it must be specific, which means that one task needs to be mentioned only and does not leave any room for the LLM to deviate from it; this would also reduce the likelihood of hallucinations [38]. The second is establishing context, which would allow the LLM to retrieve information that is relevant to the event logs that it is parsing and should ensure that it can perform the task correctly. The third is to provide domain-specific information, which refers to abbreviated terms that commonly occur within event logs. The final quality of the prompts was establishing boundaries, which meant that we informed the models not to do certain tasks and adhere to a certain format (CSV).

### *3.3. Experiment Details*

Following on from this, we conducted our experiments. We decided to use the LogHub datasets [39], as the event log data that the LLMs had to parse, which were sixteen unique datasets comprising a variety of different types and scenarios of the event log. This will be discussed in more detail in a later section of this paper. In addition, we used a Python script to access the OpenRouter API, which gave us the ability to send queries to the models and receive their responses. The Python script also included a function to save LLM responses to a CSV file that would be used to calculate the scores for each of the responses.

As we were comparing a total of ten different LLMs, we would have one hundred and sixty tests in total. In the preparation stage for our experiments, we decided to run multiple versions (while changing the model to which our queries were sent) of the same Python script to establish if it would harm any of the tests, and found this not to be the case. As a result of this, we decided to duplicate the Python script that we made for testing ten times, and this allowed us to run all the tests for a dataset simultaneously. This meant that we were able to complete the experiments over four days, as on average, we ran four tests in a day, based on the time to receive the complete response of the slowest LLM.

The use of the same Python script was an important part of our experiment as it ensured that all of the models received the same information, in the same manner. Changes were also made consistently and fell into two patterns: those based on the model changing and those based on the dataset changing. When the model changed, the only information that changed was the model that we asked OpenRouter to query. Regarding changes based on the dataset changing, these were the dataset input file itself and the specific context information changing in the prompts (so if it was Linux, we would change it to Linux).

### *3.4. Experiment Cost*

The total cost of our experiments amounted to $150 (USD). The largest model's cost was Claude 3.5 Sonnet at $40.92 for all experiments (10 experiments analysing 200,000 logs). A study [40] highlights that 12 million logs are generated in an organisation daily, and if we apply Claude's total costs, based upon our experiments, it would cost $2,452.80 to analyse the organisation's logs daily. As demonstrated in the literature review, LLM-based parsers significantly outperform their non-LLM counterparts, thereby justifying the cost. In addition, the cost estimation was based on the most expensive model; using open-source alternatives would substantially reduce the cost.

### *3.5. Results and Analysis*

To compare our results, we selected various evaluation metrics and calculated them using a Python script. The script takes the output of the LLMs (in CSV format) and calculates the scores on the metrics. They were chosen as they are specific to event log parsing and have been used throughout the literature; likewise, the details of which can be found in preceding sections.

In addition to this, we found in preliminary stages that the amount of time taken by the LLMs to process the requests differed significantly, with the quickest models performing the task within minutes and the slowest models within eight hours. Event logs are continuously generated on a large scale, and as a result of this, parsing tools that take a long time could be a cause of concern. As a result of this, we decided to include the amount of time taken by the models in our results, to allow us to discuss how it differs between models and to determine which are better suited to event log parsing and which should be avoided.

## 4. Event Log Dataset

As event logs contain sensitive data, it is difficult to obtain datasets for testing. Organisations are concerned, from a cyber security standpoint, regarding their event log data, because it can expose sensitive information and leave them vulnerable to various cyber security attacks. We resolved this issue by using the LogHub dataset by [39], which the authors mention is "benefiting research in over 450 organisations from both industry and academia". This dataset is comprehensive as it includes event logs from various operating systems, some of which include Windows, Mac, Linux and Android. The data is "not sanitised, anonymized or modified in any way". In addition to this, the data covers a large period, as the lowest time for data collection was 38.7 hours, and the highest was 263.9 days.

This data was collected using two different methods. The first method was using other datasets which researchers had already

developed and shared. This is beneficial as it removes the need to include additional datasets separately. The second method involved creating original datasets in a lab environment. The authors detail that they performed tasks in an organised manner, ensuring that events, such as anomalies, were present within the data. Data must be created in a real-world environment rather than generating the datasets, as the generated datasets could miss out certain events that would occur in the real world.

Finally, we will discuss the ground truth templates accompanying the LogHub datasets. These files consist of various columns that are unique to each dataset and are in a parsed format. They are an important resource and provide researchers with the ability to compare their tools against the structured version to determine how accurate their tool is and its strengths and deficiencies. However, the paper does not detail how they were compiled, although various event log parsing tools were tested on the dataset, so it may be a possibility that the authors used the best-performing tools to compile their ground truth files.

## 5. Prompt Engineering

We investigated prompt engineering with three main objectives. The first is having the ability to understand complex terms within logs, as the literature mentioned previously commonly had limitations when encountering logs that contain complex terminology. The second is in zero-shot scenarios, this is where the LLM has not been trained, nor seen examples of logs, and completes the tasks through descriptions and instructions of how to do so. Finally, we aim to reduce the size of the prompts. Each time an LLM processes a prompt, it incurs a cost. The larger the prompt, which is measured by the number of tokens, the higher the cost. As a result of this, ensuring that prompts are as small as possible, whilst maintaining sufficient information, is essential. The following list discusses the qualities, defined in literature, that make a good prompt specific to event log parsing.

- **Clear, relevant and specific:** Prior research highlights a range of qualities which, if adhered to, result in a significant improvement in performance. The prompt needs to be clear, relevant and set a specific task [41]. When unnecessary information is in the prompt, the likelihood of hallucinations is higher, as the LLM could retrieve information that is relevant to the unnecessary information it received and include this in its output. Regarding our scenario, the prompt would clearly state that the intention is to parse the logs provided, and it will not contain any information related to other tasks such as anomaly detection or parsing.
- **Identity:** As the models will be completing a specific task, it is important to inform them of this so that they can provide relevant responses. Previous research, such as the study [42], highlights that informing LLMs that they are acting as an entity will allow them to respond in a manner that has a high degree of accuracy and is relevant to the scenario. To achieve this, we will state that the LLM is to act as if it were an event log parser.
- **Context:** As event logs are generated by every device and these devices are used in organisations that vary in size, across the world, and in every industry, the context will differ considerably. In addition, logs, provided to an LLM to parse, may not be from commercial infrastructure and may be from a citizen's home network instead. An example scenario for a small store could be "the following logs have been generated from a small store" or for a nationwide health care provider "the following logs have been generated across devices belonging to a health care provider operating in X". Both scenarios differ considerably. The small store would likely have fewer than a hundred devices and applications to complete a finite number of tasks. However, the healthcare provider will have potentially millions of devices that contain tasks that occur in a variety of departments, including technical departments, and contain sensitive data. As a result of this, providing the LLM with context will ensure that it can accurately parse logs, as it has been provided with the relevant context.
- **Allowing the LLM to ask questions if unsure:** LLMs have been shown to hallucinate when they are unsure of information. As the focus of this paper is on event logs, this is a major limitation, as if the LLM incorrectly parses or makes up information, this could have a detrimental effect in terms of cyber security. A method shown by [43] produces outputs that are significantly less likely to contain hallucinated data. This method asks the LLMs to ask questions if certain information that was provided to them is unclear, and ensures that users can provide the LLMs with extra, but necessary, information. The information will allow the LLM to complete the task without introducing additional and potentially incorrect information or results.
- **Defining Complex Terms:** Event logs contain data that includes specific terminology relevant to the field, and LLMs are unlikely to have encountered this during their training. As highlighted in the related work section of this paper, previous research encountered limitations with tools that perform less accurately with event logs that include complex terms. To resolve this issue, we suggest defining a set of frequently used terms that are likely to be present within logs and not in the LLMs training data. For example, Windows event logs abbreviate events as "EVT" or errors as "ERR".
- **Required Output Format:** Without a specification of output format, the models would form their response in a manner which they see to be the most appropriate. Since event log parsing is one step in a broader analytical process, its output must be precise and tailored to meet organisational requirements, ensuring it can be effectively used in subsequent stages of event log analysis. This can be achieved, for example, by stating that we want the parsed logs to be presented in a table format which contains the following columns: date and time, event log ID, username, application used, and a brief description of the activity tak-

```
You will be given a series of Windows 11 event
logs to parse and act as an event log parser.
The logs that you have been provided with have
been generated in a lab and should simulate
various scenarios you could encounter in the
real world.  I will now provide you with some
terms and their abbreviated forms that could
appear in the data provided to you EVT (event),
PID (process ID), SID (security ID), ACL (access
control list), ERR (error), WRN (waning) and CRT
(critical).  Structure your output as a table
with the columns:  Date, Time, Process ID, Event
ID, Application Used and event description.
Until informed, continue to perform the task
according to the prior specification and
context.  If necessary, ask the user questions
to gain clarity when unsure.
```

Figure 2: The prompt used in this research

ing place. This will ensure that the output adheres to expected standards because it is in an organised structure and provides enough detail while eliminating irrelevant information.

- **Infinite Generation Prompt:** The research article [43] also suggests the use of "infinite generation prompts". This is when instructions are included in the original prompt which inform the LLM to adhere to the same format until notified otherwise. This is beneficial in our scenario, as event logs are generated consistently and on a large scale, and it will be time-consuming, costly, repetitive, and unnecessary to specify relevant information if it is the same each time. To use this technique, a phrase such as the following could be used "continue adhering to the prior guidelines until informed otherwise". This will allow the LLM to retain both context and output form, without the need to inform it of these every time, and users can simply inform the model if the context of the event logs changes when necessary.

### *5.1. Proposed Prompt*

Figure 2 presents the prompt used in this research. The prompt begins by defining the LLM's identity: "act as an event log parser". This notifies the LLM that it is performing the specific parsing task and not any other task related to event logs. Then, the context of the event logs is provided: "Will be given a series of Windows 11 event logs". This allows the model to better understand the logs and potentially extract information that is more relevant to this scenario. Secondly, we aim to address the limitation of complex terms "I will now provide you with some terms and their abbreviated forms". They were obtained through experience and the following resource [44] and contain a range of terms that are commonly found in event logs and which we thought the LLM was likely not to have encountered in training. Terms that were self-explanatory or common knowledge, for example, AD for Active Directory, were omitted to ensure that the input token size stays as low as possible, as we did not include unnecessary information. This section of the prompt maintained clarity in its format by defining terms and their definitions in brackets, and the LLM was informed of this to prevent any confusion and thus reduce the likelihood of hallucinations.

Following this, we inform the LLM of how it should structure its output: "Structure your output as a table with the columns". We chose a table as it is easy to read and can contain important information concisely. The columns consist of relevant fields of data and the LLM's ability to generate high-quality text to explain the logs in a manner that is easy to understand. The introduction of a priority system "priority (N/A, low, high and critical)" allows security professionals and other tools (such as anomaly detection) to focus their efforts on logs of higher priority first, compared to other tools and methods that look at logs in their entirety.

Finally, we use infinite prompt generation [43] to give the LLM the ability to adhere to the instructions set, without needing to specify them repeatedly "Until informed, continue to perform the task". In addition, we allow the LLM to retain the context provided by mentioning "according to the prior specification and context". This also comes with the advantage of solely mentioning a change in the input log's context in future prompts, without the need to specify other information, such as the identity and required output format. The final statement "If you have any questions or need clarification, feel free to ask", allows the model to ask any questions if unsure, which reduces the likelihood of hallucinations as the LLM, instead of making up information and hallucinating, will be able to obtain clarity and therefore perform its tasks correctly.

### *5.2. Data Input Format*

As parsers require the LLM to be in a certain format, our tool needs to consider this. This would also benefit LLMs. LLMs can take specific file formats instead of text, which would be easier than manually copying text and then pasting it into the LLM for further event log analysis processes that take place after the initial step of parsing.

Through testing on a variety of LLMs, we found that the original format of LLMs, for example, .evtx for Windows, was not supported by the models, and this was found to be the case universally while testing. However, we found that it was straightforward to convert the output into other forms such as CSV and XML, which is also supported by LLMs and allows effective analysis.

CSV is the format we will use in this paper as it has been shown, through research, to be a format that is commonly used, as highlighted in the study [45], who mention that "Event logs usually come in .csv or .xes format". In addition, as mentioned in the previous paragraph, all LLMs tested were able to analyse CSV files.

## 6. Experimental Setup and Evaluation

### *6.1. Preprocessing and Evaluation Pipeline Design*

To begin calculating results, we had to consider the limited context length of the LLMs. We found that none of the datasets could be received by any of the LLMs in their original forms. Due to this, we decided to split the dataset into chunks by developing a Python script. We tested various chunk sizes and found that the smallest size the LLMs would accept was fifty chunks, and a single chunk would consist of forty logs.

Following this, we developed a Python script that accesses the OpenRouter API to send different queries to the LLMs. This script comprised of two main functions, the first was a query to OpenRouter to ask for the event logs to be parsed (in chunks that are in a for loop that iterates fifty times, to account for all chunks), and the second was to save the response of the LLM as a CSV file. In the preliminary stages of our experiments, we found that the models tended to include commas within their outputs. As we used the CSV format, this meant that the rest of the table would become unreadable as the columns would be incorrectly formatted. To mitigate this, we include the following statement in the prompts "Only use commas to separate, ensure that you never put commas in the data you put into the table". In addition to this, we found that all LLMs summarised information, including the logs themselves. For example, if a set of logs were similar, the models would summarise them into a single log. As the structured files that came with the datasets were in their complete form, having the outputs structured in this manner would mean that we could not make an accurate comparison between them and the structured form. To resolve this limitation, we also included the following statement in the prompt "parse all of the logs given to you and do not interpret yourself or summarise only parse" which resolved the issue.

Secondly, we develop another Python script to act as a result calculator. The results calculator receives an output and the corresponding structured form, and following on, it calculates the scores for the various evaluation metrics. An example of this in the case of Deepseek R1 on the Android dataset would be to take the CSV output and compare it (using the evaluation metrics scores) with the structured/parsed version of the Android dataset, provided by LogHub.

Similarly to the previous Python script, we encountered various limitations which needed to be addressed to ensure that we achieved accurate results. The first of these was that the columns differed between the various datasets, and as a result, we created a global variable that could be changed to suit any set of columns without modifying the calculations. This meant that results were calculated in the same manner for each dataset, despite their columns differing. Another limitation we found was that the LineID column reset when a new chunk was parsed (meaning that LineID went to 40 and reset); this was the case for all fifty chunks. Due to this, we decided that as this is not directly related to event log parsing, and instead the architecture of the LLMs, to remove this column from the calculations. The final limitation was that the Python Script looked for exact matches in its calculations, which meant that if there was a difference in punctuation or the casing of letters, it would negatively affect the score even if parsing was performed correctly. To address this, we introduced functions to remove punctuation and compare using lowercase strings.

### *6.2. Evaluation Metrics*

Event log parsing is different from other event log analysis tasks, as determining good results would be based on the quality of parsing and the adherence to a certain form or standard, compared to tasks like anomaly detection, which is defined as simply true or false. Consequently, when evaluating event log parsing, evaluation metrics that are specific and relevant should be selected. For this paper, we selected three different metrics that have been used throughout the literature [46, 16, 47, 48, 14], which were the F1-Grouping Accuracy score (F1-G), the F1-Template Accuracy score (F1-T) and the Average Edit Distance (AED). F1-G is used to determine how accurate a response is by comparing the total number of logs parsed against the structured form and takes into account both precision and recall. F1-T also takes precision and recall into account, but is based on how accurately a log is parsed. If a log was parsed identically to the structured file, the F1-T score would be 1. These two metrics were selected as they provide a balanced score that takes into account both true positives and false positives. AED is the average edit distance score, and this score is how many changes need to be made to the received log to be the same as the structured version. If this score is low, we can decipher that the log is close to being identical to its structured LogHub counterpart.

### *6.3. Results*

There were two main points to consider when comparing LLMs. The first is how they perform compared to each other. To achieve this, we used evaluation metrics, as mentioned in the previous section, to calculate scores, and the scores would be used to compare the performance of the LLMs. These scores can be found in Figure 1, the table has columns, which are the LLMs that were selected, subcolumns, which are the various scores achieved on the evaluation metrics, and rows consisting of the sixteen LogHub datasets. The F1-scores, both for group and template accuracy, are presented as decimals that are rounded to two decimal places, whereas average edit distance is presented as percentages and is also rounded to two decimal places. It can be seen that the term "ERROR" is also found within the results; when this appears, it refers to where the output was presented in a form that is unreadable by our results calculator. These unreadable responses were due to three different factors: LLMs not being able to parse the logs, deviating from the set structure of columns, and severe hallucinations, which led to responses that were not related in any form to the task. The latter factor only occurred in some responses from Llama and Qwen.

Additionally, three figures are provided to support the interpretation of our results. Figure 3 displays the average F1 grouping accuracy scores across all models. Figure 4 presents the average F1 template accuracy scores for each model. Lastly, Figure 5 shows the average edit distance scores per model.

Table 1: Performance Metrics Table

| Row | Deepseek R1 | | | Qwen: QwQ | | | Mistral Pixtral Large 2411 | | | Llama 3.3 70B Instruct | | | Gemma 3 | | | GPT-4o | | | Claude 3.7 Sonnet | | | Phi-4 | | | Cohere: Command R7B | | | Grok 2 1212 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED | F1-G | F1-T | AED |
| Android | 0.93 | 0.35 | 48.24% | 0.76 | 0.33 | 50.79% | 0.59 | 0.32 | 43.59% | 0.61 | 0.33 | 53.48% | 0.96 | 0.34 | 50.02% | 0.98 | 0.32 | 48.46% | 0.96 | 0.34 | 48.53% | 0.58 | 0.36 | 49.72% | 0.41 | 0.35 | 52.22% | 0.95 | 0.35 | 49.29% |
| Apache | 0.86 | 0.63 | 2.55% | 0.11 | 0.66 | 1.04% | 0.12 | 0.66 | 1.34% | ERROR | ERROR | ERROR | 0.63 | 0.62 | 1.89% | 0.99 | 0.63 | 2.80% | 0.98 | 0.63 | 2.84% | 0.59 | 0.62 | 2.50% | ERROR | ERROR | ERROR | 0.68 | 0.61 | 3.35% |
| BGL | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | 0.05 | 0.69 | 6.38% | 0.05 | 0.72 | 6.77% | 0.24 | 0.74 | 70.21% | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | 0.12 | 0.68 | 7.74% |
| HDFS | 0.84 | 0.90 | 2.26% | 0.36 | 0.83 | 2.48% | 0.71 | 0.89 | 2.47% | 0.86 | 0.91 | 2.17% | 0.99 | 0.91 | 2.01% | 0.99 | 0.91 | 2.01% | 0.99 | 0.91 | 2.01% | 0.67 | 0.90 | 2.19% | 0.16 | 0.78 | 3.45% | 0.97 | 0.91 | 2.03% |
| HPC | 0.38 | 1.00 | 0.25% | 0.13 | 1.00 | 0.00% | 0.04 | 0.90 | 0.70% | 0.02 | 1.00 | 0.00% | 0.41 | 0.95 | 0.34% | 0.71 | 0.98 | 0.04% | 0.88 | 1.00 | 0.00% | 0.10 | 1.00 | 0.00% | 0.15 | 0.58 | 0.43% | 0.02 | 1.00 | 0.00% |
| Hadoop | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| HealthApp | 0.89 | 0.78 | 2.18% | 0.44 | 0.77 | 4.15% | 0.74 | 0.73 | 3.12% | 0.70 | 0.82 | 2.70% | 0.99 | 0.78 | 2.10% | 0.98 | 0.76 | 3.27% | 0.99 | 0.76 | 3.14% | 0.88 | 0.77 | 2.13% | 0.31 | 0.77 | 5.46% | 0.98 | 0.78 | 2.10% |
| Linux | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| Mac | 0.63 | 0.42 | 31.20% | 0.20 | 0.50 | 50.00% | 0.06 | 0.69 | 21.65% | 0.04 | 0.64 | 38.62% | 0.71 | 0.26 | 19.33% | 0.96 | 0.19 | 15.37% | 0.96 | 0.19 | 15.33% | 0.15 | 0.30 | 25.64% | ERROR | ERROR | ERROR | 0.52 | 0.17 | 14.63% |
| OpenSSH | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| OpenStack | 0.01 | 1.00 | 0.00% | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | 0.00 | 1.00 | 0.00% | 0.03 | 0.60 | 17.05% | 0.00 | 1.00 | 0.00% | 0.01 | 0.86 | 4.36% | 0.00 | 0.83 | 7.83% | ERROR | ERROR | ERROR | 0.02 | 0.95 | 2.47% |
| Proxifier | 0.83 | 0.57 | 16.59% | 0.53 | 0.44 | 23.48% | 0.39 | 0.45 | 20.41% | 0.15 | 0.30 | 32.50% | 0.89 | 0.62 | 14.03% | 0.98 | 0.33 | 31.42% | 0.79 | 0.28 | 34.57% | 0.57 | 0.43 | 17.99% | 0.12 | 0.45 | 13.82% | 0.84 | 0.61 | 16.31% |
| Spark | 0.92 | 0.03 | 21.60% | 0.43 | 0.03 | 21.06% | 0.76 | 0.02 | 20.98% | 0.55 | 0.02 | 21.36% | 0.99 | 0.03 | 21.52% | 0.99 | 0.02 | 20.98% | 0.99 | 0.03 | 20.96% | 0.96 | 0.03 | 21.68% | 0.21 | 0.03 | 22.59% | 0.98 | 0.03 | 21.61% |
| Thunderbird | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| Windows | 0.88 | 0.00 | 43.43% | 0.76 | 0.20 | 45.66% | 0.47 | 0.20 | 46.73% | 0.35 | 0.20 | 42.64% | 0.96 | 0.22 | 42.37% | 0.96 | 0.22 | 47.18% | 0.99 | 0.20 | 46.56% | 0.97 | 0.23 | 41.63% | 0.27 | 0.21 | 42.35% | 0.95 | 0.22 | 41.94% |
| Zookeeper | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR | ERROR |
| Average | 0.72 | 0.57 | 16.83% | 0.41 | 0.53 | 22.07% | 0.43 | 0.54 | 17.89% | 0.37 | 0.58 | 27.64% | 0.69 | 0.55 | 16.11% | 0.78 | 0.55 | 16.21% | 0.80 | 0.54 | 22.59% | 0.55 | 0.55 | 17.13% | 0.23 | 0.45 | 20 | 0.64 | 0.57 | 14.68% |

Table 2: Table demonstrating time taken in [hours:minutes] format to process parsing query for each LLM on LogHub Datasets

| Dataset | Deepseek R1 | Qwen: QwQ | Mistral Pixtral | Llama 3.3 70B | Gemma 3 | GPT 4o | Claude 3.7 | Phi-4 | Cohere R7B | Grok 2 |
|---|---|---|---|---|---|---|---|---|---|---|
| Android | 04:17 | 03:23 | 07:53 | 00:23 | 01:20 | 00:15 | 00:09 | 01:10 | 00:24 | 00:25 |
| Apache | 03:27 | 00:23 | 00:24 | 00:24 | 00:25 | 00:10 | 00:21 | 00:23 | 00:10 | 00:21 |
| BGL | 04:33 | 02:25 | 02:45 | 01:15 | 02:28 | 00:15 | 00:40 | 00:47 | 00:20 | 00:37 |
| HDFS | 03:10 | 02:20 | 01:17 | 00:51 | 01:45 | 00:11 | 00:30 | 00:35 | 00:19 | 00:31 |
| HPC | 03:00 | 02:19 | 01:05 | 00:27 | 01:10 | 00:07 | 00:15 | 00:17 | 00:08 | 00:15 |
| Hadoop | 03:13 | 03:14 | 01:43 | 01:07 | 01:47 | 00:12 | 00:32 | 00:50 | 00:14 | 00:33 |
| HealthApp | 02:14 | 02:27 | 01:14 | 00:38 | 01:10 | 00:10 | 00:19 | 00:29 | 00:18 | 00:28 |
| Linux | 02:32 | 03:22 | 01:03 | 00:52 | 01:34 | 00:13 | 00:25 | 00:36 | 00:12 | 00:29 |
| Mac | 03:30 | 03:11 | 00:58 | 00:57 | 01:36 | 00:13 | 00:35 | 00:33 | 00:14 | 00:25 |
| OpenSSH | 03:48 | 03:04 | 01:19 | 00:36 | 01:10 | 00:09 | 00:19 | 00:29 | 00:14 | 00:23 |
| OpenStack | 04:51 | 04:18 | 04:28 | 01:44 | 03:10 | 00:12 | 01:06 | 01:49 | 00:36 | 00:54 |
| Proxifier | 03:34 | 03:53 | 01:20 | 00:49 | 01:15 | 00:12 | 00:31 | 00:35 | 00:14 | 00:29 |
| Spark | 03:00 | 02:51 | 00:48 | 00:34 | 01:00 | 00:10 | 00:17 | 00:23 | 00:09 | 00:20 |
| Thunderbird | 03:58 | 02:36 | 02:46 | 01:18 | 02:08 | 00:18 | 00:31 | 00:49 | 00:15 | 00:40 |
| Windows | 02:54 | 03:44 | 01:28 | 01:01 | 01:05 | 00:12 | 00:21 | 00:35 | 00:14 | 00:28 |
| Zookeeper | 04:02 | 04:19 | 01:10 | 01:03 | 01:37 | 00:14 | 00:26 | 00:42 | 00:14 | 00:30 |
| Average | 03:30 | 2:59 | 01:58 | 00:52 | 01:32 | 00:12 | 00:27 | 00:41 | 00:15 | 00:29 |

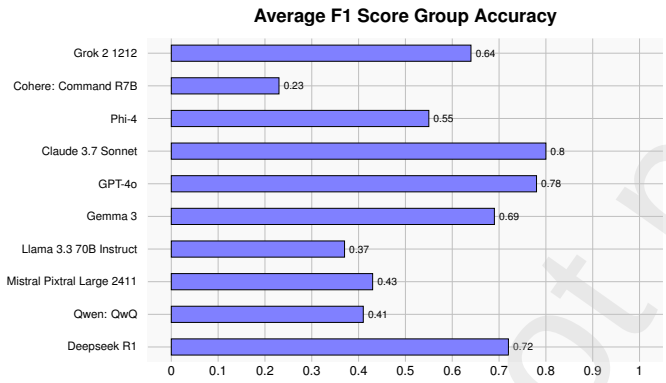


Figure 3: Graph illustrating the average F1 grouping accuracy scores.

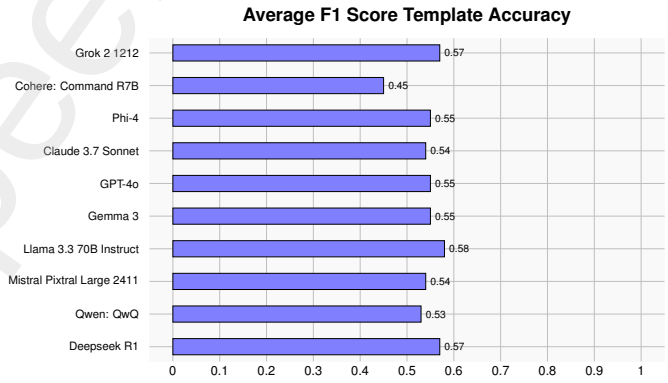


Figure 4: Graph illustrating the average F1 template accuracy scores.

The second factor to consider is the time taken to complete the requests, with results shown in Figure 2. As with the previous table, the columns represent the selected LLMs and the rows correspond to the LogHub datasets. Each value is formatted as hours:minutes, rounded to the nearest minute. Additionally, Figure 6 provides a visual summary of the average completion time for each model.

## 7. Discussion

### *7.1. Reasoning Models*

An important development that served as inspiration for this paper was open-source models, namely Deepseek and Qwen, that used reasoning within their instructions to produce outputs that were of higher quality and precision. As these models were shown to equal and at times outperform GPT and Claude in other tasks, in the literature, we predicted that the same would be the case for event log parsing. Nonetheless, Deepseek was shown to perform slightly better than the other open-source models and less than GPT and Claude, and Qwen achieved results that were among the worst. The reason for Qwen's deficient performance, from a manual analysis of some of the outputs, is due to excessive reasoning, which leads to the model experiencing a lack of clarity and then performing the task incorrectly and at times, hallucinations.

The main disadvantage of both models is processing time; as mentioned previously, the time taken to complete the task is a major factor in choosing a model for parsing. Both models took significantly longer than any of the others to complete the tasks. This can be explained as being due to reasoning. As the results do not show a major advantage, due to reasoning, and were lower than the closed-source models, their use as event log parsers cannot be justified.

### *7.2. Errors, Hallucinations and Decrease in Results*

First, we will address the outlier found in Mistral Pixtral Large 2411's time to process the Android dataset. An analysis

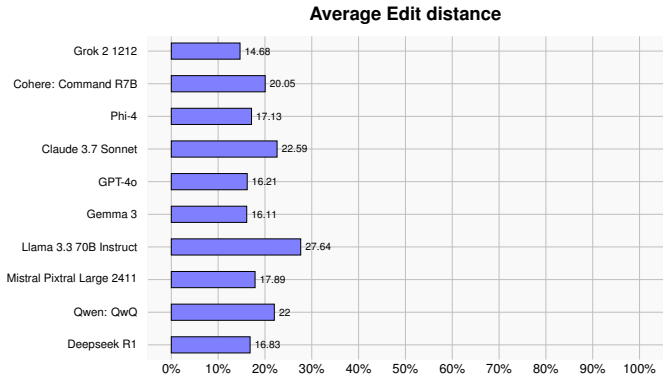


Figure 5: Graph illustrating the average edit distance scores.

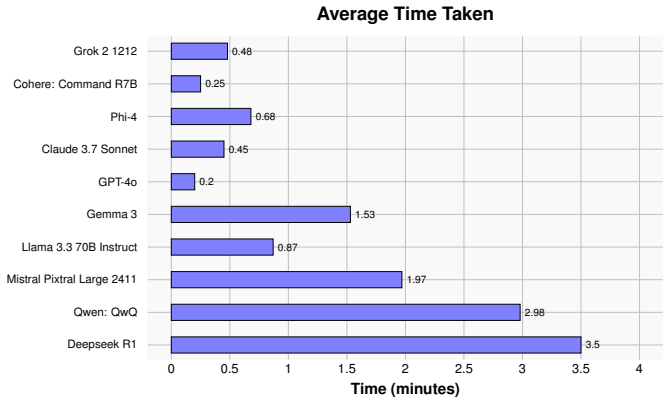


Figure 6: Graph illustrating the average time taken by each of the models to complete their requests.

of the activity logs in OpenRouter showed that normal activity took place for approximately half an hour, then there was a pause for approximately seven hours, and finally, the model processed its final request. As only the final request was singular and took place after a large amount of time, we can infer that this was due to a connection issue between OpenRouter and Mistral (as other tests were running without difficulty during this time), and therefore, it is not an accurate representation of the model's performance.

Our results show frequent errors for datasets such as OpenSSH, BGL and Linux, which contain less common event logs. In addition to this, even for more common event log types, scores are far from perfect. In the literature, using the same datasets, it was found that prior LLM-based parsers consistently achieved close to perfect scores using the same metrics and had no errors. Based on this, we can deduce that the cause of the errors and decrease in results is that the base models are not trained on event log types that are less common and witness fewer examples of more common event logs. In prior research, the parsers directly encountered relevant examples, either through fine-tuning or through in-context learning. This highlights that relying solely on prompt engineering is not sufficient, as less common event log types can experience a significant decrease in their parsing ability or, at times, not be parsed.

Following this, the errors encountered during result calculation were primarily due to model hallucinations and deviations from the expected task and structure. For the vast majority of LLMs, this was a deviation in structure; however, for Qwen and Llama, this was frequently due to deviating to topics that had no relation to the event logs. This limitation is a cause for concern, as it could potentially be present in other LLM-based parsers. As event log parsing is a sensitive task, the risk of deviations from the required and expected outputs is something that cannot be afforded.

### *7.3. Superior Scores Achieved In Group Accuracy*

The vast majority of models achieve superior scores in F1-G compared to F1-T (where scores in general are lower) and AED (where they have achieved high scores, in this metric, this is a disadvantage). This means that the models parsed the event logs successfully, but the parsed logs differ in structure from their ground-truth counterparts. However, this is not necessarily a disadvantage, as this could happen due to differences in punctuation or the use of different words or phrases, whilst retaining the same and correct meanings. Both would have no value in terms of parsing quality. In addition, it would not result in any difference in performance when the parsed output is used in further event log analysis tasks. Through manual analysis, we found consistent differences across all models and datasets, primarily in punctuation, word choice, and phrasing. As a result, we determined that F1 Group Accuracy is the most reliable metric for evaluating parsing performance among the models.

### *7.4. Time Taken To Process Request*

As mentioned previously, event logs are generated on a large scale. The study [40] found that, while testing within an organisation, "Nearly 12 million events are collected each day". Our results, which are similar to those reviewed from the literature, show that the lowest average time taken was 12 minutes, and this is based on datasets which contain only 2000 logs. This means that if run as a single iteration, it would only run 5 times an hour and 120 times in a day. Meeting the demand for millions of logs would require running separate instances in large numbers, and this is neither practical nor realistic. This is an important point that is essential to address in future research, as it means that LLM-based parsers cannot run in real-world environments. A potential method could be to leverage agents, a group (that is calculated to meet the demand of the average log files) of twin agents can be assigned an equal amount of chunks and parse them, then a separate agent can take all of the outputs and combine them into a single one.

### *7.5. How Do Open-Source Models Compare Against Closed-Source?*

A major objective of our paper was to evaluate how open-source LLMs compare to closed-source models when parsing event logs. The results show that the closed-source LLMs, GPT-4o and Claude 3.7 Sonnet, significantly outperform the open-source models selected. GPT-4o achieved the second highest average F1-G score, and Claude 3.7 Sonnet achieved

the highest average F1-G score, but had a slightly higher average AED score than most models.

These were especially represented in the F1 group accuracy scores, which is arguably the most important metric used. This is because it illustrates how correctly parsing was performed based on comparing the number of logs in the output and the ground-truth. As the scores are high, we can infer that the vast majority of logs were parsed successfully, and thus the closed-source models would be more valuable in real-world scenarios if their security limitations were overlooked. However, it is surprising to see the high scores achieved by Claude in average edit distance, which suggests that the responses provided differ structurally compared to the ground truth. This does not necessarily mean they are incorrect or are deficient in quality, as edit distance scores will be higher if changes (regardless of value) need to be made to the response to make it identical to the respective ground-truth file.

In addition, both models completed their requests much faster than their counterparts. This is a major advantage in terms of parsing, as it needs to take place as quickly as possible because event logs are generated on a large scale.

It is important to note that the security limitations mentioned previously are of the highest severity, and the use of the closed-source models, despite outperforming all of the open-source models, cannot be a reasonable suggestion. All event log analysis tasks are performed to improve security, and as a result of this, using a tool that performs worse but does not have these limitations is the only suggestion that is acceptable.

### *7.6. Lowest Performing Models*

Following this, based on our results, we will first discuss models that are not best suited to event log parsing. Llama 3.3 70B Instruct, in general, was the worst performing LLM, not only due to the severe hallucinations that were discussed previously, but also because it consistently received some of the worst scores on the evaluation metrics. Although Cohere: Command R7B achieved the worst scores in all metrics, except AED (where Llama achieved the lowest AED score), it did not experience errors that were in the frequency or severity, which Llama displayed. Following these is Qwen: QwQ, due to the reasons mentioned previously, which, in brief, were severe hallucinations and a high processing time, which did not make a positive difference in terms of results. The final of the low-performing LLMs is Phi-4 and Mistral Pixtral Large 2411, which achieved results that were not as low as the previously mentioned models; however, they were significantly lower when compared to the other models.

The information presented in the previous paragraph contains models, which vary in their parameter sizes and some, such as Qwen, scored results that competed with GPT and Claude, in previous research, which was related to other tasks. This means that, in general, we can say that performance on event log parsing is not determined by these factors and others, such as training data, are more relevant and would lead to more satisfactory results.

### *7.7. Recommended Model*

In contrast, we will discuss which models performed the best, which, after GPT and Claude, were Gemma 3 and Grok 2 1212, which consistently received high F1-G and F1-T scores and low AED scores. In terms of time taken to process the request, surprisingly, Gemma 3 had the highest average, after Qwen and Deepseek, whereas Grok 2 1212 was the quickest.

As a result of this, we have two recommendations. If the processing time limitation could be addressed, Gemma 3 would be our suggestion, as it had the highest F1-G score, and as mentioned previously, this is the best metric to use to determine model performance. If it cannot be resolved, then Grok 2 1212 would be our recommendation, as it achieved the closest score to Gemma 3 and surpassed it in the other two metrics. In addition, it has the lowest average time taken to process the requests, which is a major advantage.

### *7.8. Threats to Validity*

LLMs suffer from non-determinism, which refers to when their responses differ for the same query [49]. Relating this to our paper, if we re-ran our experiments, the results could be different due to non-determinism. A potential method to address this, in future research, could be to perform multiple iterations of the experiments and use the average of these results as the model's scores. However, this solution has disadvantages as it will require additional time for experiments, which already consume a lot of time and multiply costs.

In addition, the study does not include a direct comparison with other LLM techniques, such as RAG, fine-tuning and in-context learning. In the study, we conclude that prompt engineering, when isolated, is not sufficient to be used as an LLM-based event log parser technique, as our results are significantly lower compared to prior research, which used the aforementioned techniques. As a result of this, future work could include a direct comparison with other methods to further validate this conclusion.

## 8. Conclusion and Future Work

In our study, we assess how prompt engineering and the use of different LLMs, particularly open-source, affect event log parsing using LLMs. We found that prompt engineering was unsuitable for use as a stand-alone technique, as performance decreased significantly in our study compared to those before it. Furthermore, all models took relatively long processing times when parsing just 2,000 event logs. This indicates that, in their current form, they are not suitable for deployment in real-world applications where scalability is critical. While closed-source LLMs showed better performance, they were excluded from consideration due to security and privacy concerns. Among the open-source alternatives, our recommended models are Gemma 3 and Grok 2 1212. The final choice between them depends on whether the time-related limitations of Gemma 3 can be effectively mitigated.

Building upon these findings, future research could aim to address the time taken by LLMs to parse event logs, potentially

with the use of multiple agents. In addition, it could assess if prompt engineering, in combination with other techniques, such as fine-tuning, can improve results, as our study is based on its independent use. Finally, our study showed that LLMs tended to hallucinate when receiving uncommon types of event logs. Future work could assess if this still occurs when fine-tuning and RAG are used to validate the accuracy of parsing tools.

## CRediT authorship contribution statement

**Siraaj Akhtar:** Conceptualisation, Methodology, Software, Writing – original draft. **Saad Khan:** Supervision, Conceptualisation, Methodology, Writing – review & editing. **Simon Parkinson:** Supervision, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no financial or personal relationships with individuals or organisations that could inappropriately influence the content of this work. Furthermore, they have no professional or personal interests in any product, service, or company that could be perceived as affecting the views expressed in or the review of this manuscript.

## References


[1] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, M. R. Lyu, Tools and benchmarks for automated log parsing. in 2019 ieee/acm 41st international conference on software engineering: Software engineering in practice (icse-seip) (2019).

[2] X. Fan, C. Tao, Towards resilient and efficient llms: A comparative study of efficiency, performance, and adversarial robustness, arXiv preprint arXiv:2408.04585 (2024).

[3] M. Mehrabi, A. Hamou-Lhadj, H. Moosavi, The effectiveness of compact fine-tuned llms in log parsing, in: 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2024, pp. 438–448. doi:10.1109/ICSME58944.2024.00047.

[4] V. B. Parthasarathy, A. Zafar, A. Khan, A. Shahid, The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities, arXiv preprint arXiv:2408.13296 (2024).

[5] J. Xu, R. Yang, Y. Huo, C. Zhang, P. He, Divlog: Log parsing with prompt enhanced in-context learning. in 2024 ieee/acm 46th international conference on software engineering (icse), IEEE Computer Society (2024) 983–983.

[6] H. Wu, X. Chen, K. Huang, Resource management for low-latency cooperative fine-tuning of foundation models at the network edge, IEEE Transactions on Wireless Communications (2025).

[7] M. Astekin, M. Hort, L. Moonen, A comparative study on large language models for log parsing, in: Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2024, pp. 234–244.

[8] S. Rangaraju, A comprehensive analysis of gpt applications in third-party vendor security enhancement, Asian Journal of Multidisciplinary Research & Review 4 (6) (2023) 105–115.

[9] P. He, J. Zhu, S. He, J. Li, M. R. Lyu, An evaluation study on log parsing and its use in log mining, in: 2016 46th annual IEEE/IFIP international conference on dependable systems and networks (DSN), IEEE, 2016, pp. 654–661.

[10] Z. Li, Q. Fu, Z. Huang, J. Yu, Y. Li, Y. Lai, Y. Ma, Revisiting log parsing: The present, the future, and the uncertainties, IEEE Transactions on Reliability 73 (3) (2024) 1459–1472.

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

[12] P. He, J. Zhu, Z. Zheng, M. R. Lyu, Drain: An online log parsing approach with fixed depth tree, in: 2017 IEEE international conference on web services (ICWS), IEEE, 2017, pp. 33–40.

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

[14] S. Yu, Y. Wu, Z. Li, P. He, N. Chen, C. Liu, Log parsing with generalization ability under new log types, in: Proceedings of the 31st acm joint european software engineering conference and symposium on the foundations of software engineering, 2023, pp. 425–437.

[15] Y. Zhou, Y. Chen, X. Rao, Y. Zhou, Y. Li, C. Hu, Leveraging large language models and bert for log parsing and anomaly detection, Mathematics (2024).

[16] Z. Jiang, J. Liu, Z. Chen, Y. Li, J. Huang, Y. Huo, P. He, J. Gu, M. R. Lyu, Lilac: Log parsing using llms with adaptive parsing cache, Proceedings of the ACM on Software Engineering 1 (FSE) (2024) 137–160.

[17] Y. Wu, S. Yu, Y. Li, Log parsing with self-generated in-context learning and self-correction, arXiv preprint arXiv:2406.03376 (2024).

[18] X. Yu, S. Nong, D. He, W. Zheng, T. Ma, N. Liu, J. Li, G. Xie, Loggenius: An unsupervised log parsing framework with zero-shot prompt engineering, in: 2024 IEEE International Conference on Web Services (ICWS), 2024, pp. 1321–1328. doi:10.1109/ICWS62655.2024.00159.

[19] C. Zhi, L. Cheng, M. Liu, X. Zhao, Y. Xu, S. Deng, Llm-powered zero-shot online log parsing, in: 2024 IEEE International Conference on Web Services (ICWS), 2024, pp. 877–887. doi:10.1109/ICWS62655.2024.00106.

[20] H. Ju, Reliable online log parsing using large language models with retrieval-augmented generation, in: 2024 IEEE 35th International Symposium on Software Reliability Engineering Workshops (ISSREW), 2024, pp. 99–102. doi:10.1109/ISSREW63542.2024.00055.

[21] Y. Xiao, V.-H. Le, H. Zhang, Stronger, cheaper and demonstration-free log parsing with llms, arXiv preprint arXiv:2406.06156 (2024).

[22] J. Huang, Z. Jiang, Z. Chen, M. R. Lyu, Lunar: Unsupervised llm-based log parsing, arXiv preprint arXiv:2406.07174 (2024).

[23] C. Pei, Z. Liu, J. Li, E. Zhang, L. Zhang, H. Zhang, W. Chen, D. Pei, G. Xie, Self-evolutionary group-wise log parsing based on large language model, in: 2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE), 2024, pp. 49–60. doi:10.1109/ISSRE62328.2024.00016.

[24] A. Xu, A. Gau, Help: Hierarchical embeddings-based log parsing, arXiv preprint arXiv:2408.08300 (2024).

[25] Y. Lu, R. Karanjai, D. Alsagheer, K. Kasichainula, L. Xu, W. Shi, S.-H. S. Huang, Logbabylon: A unified framework for cross-log file integration and analysis, in: Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing, 2025, pp. 1953–1960.

[26] X. Huang, T. Zhang, W. Zhao, Logrules: Enhancing log analysis capability of large language models through rules, in: Findings of the Association for Computational Linguistics: NAACL 2025, 2025, pp. 452–470.

[27] R. Vaarandi, H. Bahşi, Using large language models for template detection from security event logs, International Journal of Information Security 24 (3) (2025) 104.

[28] M. Mahyoob, J. Al-Garaady, Deepseek vs. chatgpt: Comparative efficacy in reasoning for adults' second language acquisition analysis, ChatGPT: Comparative Efficacy in Reasoning for Adults' Second Language Acquisition Analysis (February 02, 2025) (2025).

[29] A. Rahman, S. H. Mahir, M. T. A. Tashrif, A. A. Aishi, M. A. Karim, D. Kundu, T. Debnath, M. A. A. Moududi, M. Eidmum, Comparative analysis based on deepseek, chatgpt, and google gemini: Features, techniques, performance, future prospects, arXiv preprint arXiv:2503.04783 (2025).

[30] S. Singh, S. Bansal, A. E. Saddik, M. Saini, From chatgpt to deepseek ai: A comprehensive analysis of evolution, deviation, and future implications in ai-language models, arXiv preprint arXiv:2504.03219 (2025).

[31] S. Joshi, A comprehensive review of deepseek: Performance, architecture and capabilities (2025).

[32] S. Akhtar, S. Khan, S. Parkinson, Llm-based event log analysis techniques: A survey, arXiv preprint arXiv:2502.00677 (2025).

[33] C. Sun, Y. Li, D. Wu, B. Boulet, Onioneval: An unified evaluation of fact-conflicting hallucination for small-large language models, arXiv preprint arXiv:2501.12975 (2025).

[34] Q. Jiang, Z. Gao, G. E. Karniadakis, Deepseek vs. chatgpt vs. claude: A

comparative study for scientific computing and scientific machine learning tasks, Theoretical and Applied Mechanics Letters (2025) 100583.
[35] M. Ali, P. Rao, Y. Mai, B. Xie, Using benchmarking infrastructure to evaluate llm performance on cs concept inventories: Challenges, opportunities, and critiques, in: Proceedings of the 2024 ACM Conference on International Computing Education Research-Volume 1, 2024, pp. 452–468.
[36] L. Yang, R. Jin, L. Shi, J. Peng, Y. Chen, D. Xiong, Probench: Benchmarking large language models in competitive programming, arXiv preprint arXiv:2502.20868 (2025).
[37] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann, et al., Phi-4 technical report, arXiv preprint arXiv:2412.08905 (2024).
[38] J. He, M. Rungta, D. Koleczek, A. Sekhon, F. X. Wang, S. Hasan, Does prompt formatting have any impact on llm performance?, arXiv preprint arXiv:2411.10541 (2024).
[39] J. Zhu, S. He, P. He, J. Liu, M. R. Lyu, Loghub: A large collection of system log datasets for ai-driven log analytics, in: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 355–366.
[40] R. Vaarandi, M. Kont, M. Pihelgas, Event log analysis with the logcluster tool, in: MILCOM 2016-2016 IEEE Military Communications Conference, IEEE, 2016, pp. 982–987.
[41] L. Wang, X. Chen, X. Deng, H. Wen, M. You, W. Liu, Q. Li, J. Li, Prompt engineering in consistency and reliability with the evidence-based guideline for llms, npj Digital Medicine 7 (1) (2024) 41.
[42] Y. Shao, L. Li, J. Dai, X. Qiu, Character-llm: A trainable agent for role-playing, arXiv preprint arXiv:2310.10158 (2023).
[43] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, D. C. Schmidt, A prompt pattern catalog to enhance prompt engineering with chatgpt, arXiv preprint arXiv:2302.11382 (2023).
[44] [link].
URL `https://learn.microsoft.com/en-us/windows/win32/wes/windows-event-log`
[45] S. Esser, D. Fahland, Using graph data structures for event logs (2019).
[46] A. Zhong, D. Mo, G. Liu, J. Liu, Q. Lu, Q. Zhou, J. Wu, Q. Li, Q. Wen, Logparser-llm: Advancing efficient log parsing with large language models, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 4559–4570. `doi:10.1145/3637528.3671810`.
URL `https://doi.org/10.1145/3637528.3671810`
[47] V. Beck, M. Landauer, M. Wurzenberger, F. Skopik, A. Rauber, Sok: Llm-based log parsing, arXiv preprint arXiv:2504.04877 (2025).
[48] Y. Fu, M. Yan, J. Xu, J. Li, Z. Liu, X. Zhang, D. Yang, Investigating and improving log parsing in practice, in: Proceedings of the 30th ACM joint european software engineering conference and symposium on the foundations of software engineering, 2022, pp. 1566–1577.
[49] Y. Song, G. Wang, S. Li, B. Y. Lin, The good, the bad, and the greedy: Evaluation of llms should not ignore non-determinism, arXiv preprint arXiv:2407.10457 (2024).