Zero-shot Log-based Anomaly Detection with LLM

by

Wesley Jones

A Creative Component submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Cybersecurity

Program of Study Committee:
Yong Guan, Major Professor

Iowa State University

Ames, Iowa

2025

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Properly analyzing logs in a large distributed environment is complicated. Modern log environments require a multi-step process to collect, parse, index, and ultimately analyze logs. This process requires human intervention throughout — a burden of labor and knowledge. Today tools like machine learning seek to solve this problem, but that introduces further reliance on the log processing step; ultimately making more work for engineers.

Implementing an large language model (LLM) to perform zero-shot anomaly detection (not requiring correct parsing, index, and known good states — all the trappings of existing systems). The objective in this research is to address the shortcomings of existing complex solutions that rely on time and knowledge, fail to handle log formatting changes, and have too strict of tolerances for known states. LLMs are a solution that is evolving significantly and can technically align with the requirements to find zero-shot anomalies in logs, while side-stepping the issues found in current solutions. There are studies that explore LLM-informed anomaly detection in adjacent fields, but for system and application logs this topic has not yet been sufficiently explored.

Finally, looking forward to the future, standard systems must be developed. This should affect both system design and analysis of log sequences themselves. By taking this step during the early phases of LLM integration, engineers and researchers can keep the apace with the advancements in LLM infrastructure as it becomes available and ultimately ensure that all logs are eventually audited.

CHAPTER 1. Introduction

Logging environments can evolve into large clusters dedicated to a few common steps necessary to proper logs handling: ingest and parsing, indexing, searching, and anomaly detection. Each of these components begets the other. The nature of the logging ecosystem itself evolves as applications and systems change — so does the log output.

Due to these circumstances it is desirable to create a flexible system to properly suss out logs of interest. There is great advantages ranging from human time to business security in correctly identifying an anomalous event within the mass of logs that any sizable environment will generate. To handle each necessary step there are a variety of choices that offer a "turn-key" solution. Notable solutions in this space are Elasticsearch[1] and Splunk[2]. Each of these platforms seeks to alleviate the process of ingest, index, and search. They typically have extensive services for anomaly detection.

Anomaly detection exists as a standalone field of study. The statistical conundrum of detecting the needle-in-a-haystack can be addressed with a variety of methods, even ones geared for high dimensional data (e.g. logs) [28]. These methods will often be implemented using deep learning techniques, resulting in a neural network that can handle the task of locating relevant outliers.

1.1 Background on Log Systems

As logging system complexity grows there is an advantage to removing overhead wherever possible. Common areas where deficiencies occur within logging systems are:

1. **Log parsing.** This is due to the shifting from the origins of the logs. Even in highly structured environments such as Windows, the motivation to optimize the log size by including only relevant fields versus the need to include all the relevant details, causes enough differential between logs to

[1] `https://elastic.co`
[2] `https://splunk.com`

create errors during parsing. The results of the parsing will provide thousands of unique fields in a centralized environment.

2. **Human analyzing time and knowledge.** Deep learning systems might offset the human time requirements for analyzation, but ultimately there are some inescapable elements. The process of correctly identifying an anomalous occurrence still alludes artificial intelligence. Also, the identification of determining factors of anomalies remains soundly in the hands of humans. Both of these elements are time and knowledge intensive — and will reoccur periodically due to natural shifts in business systems and applications.

1.1.1 Traditional Logging Environment Architecture

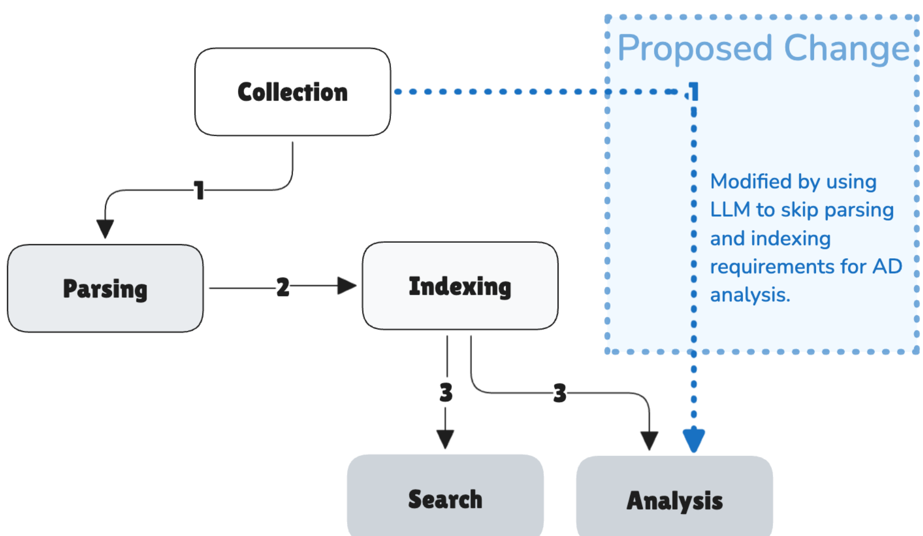

Figure 1.1: *A generic log system involving five common processes.*

To describe the series of processing steps that logging environments typically provide, I will lay two common types of scenarios.

1.1.1.1 Centralized Collection and Search

In this setup, logs are collected using a series of centralization tools and then offered to the engineers for searching by another tool that typically has indexed the logs for speed and reliability reasons.

1. Client generates log and submits to a network collector.
2. The collection tool parses the log and ingests it to the indexer.
3. The indexer defines field patterns and makes the log available within a larger dataset for the search tool to begin searching.

This sequence is straightforward to maintain as each component can act independently of the other. A log that is ineffectively parsed can still be searched in the search tool, albeit not as effectively.

1.1.1.2 Centralized collection and search with anomaly detection using deep learning

This extends the previous step by adding deep learning. It has the disadvantage of increasing complexity and tethering the function of the neural network on the correct parsing of logs (step 2 above). To implement the learning a fourth step is added:

4. Group and train a neural network on the logs to perform deep learning on patterns and formats.

There are some major advantages with this process. Newer research has offered solutions that back feed the pattern discovery to the log parsing step and the advantage of having a neural network at the disposal of the engineer for anomaly detection [16, 21].

1.1.2 Traditional Log Processing Steps

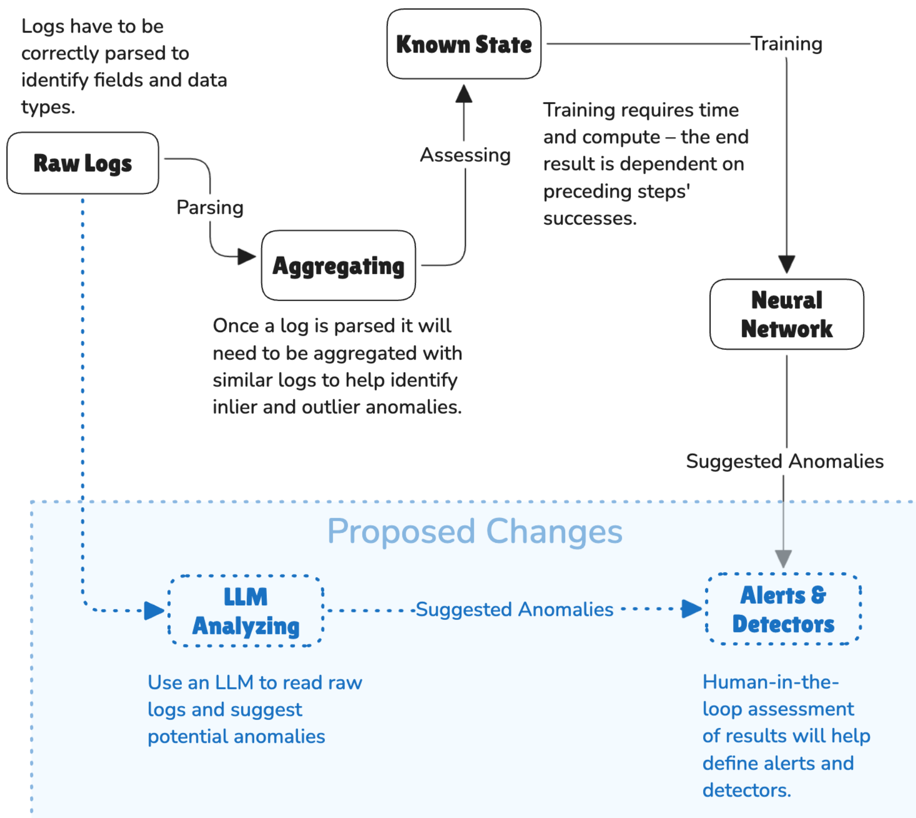

Figure 1.2: *Generalized log processing steps to take a set of raw logs to a useful training set for a neural network.*

For optimal querying and neural network training purposes a typical log must be properly parsed. This process can be as simple as identifying several key fields, but typically engineers prefer a fully labeled and data typed log document (strings are surrounded in “”, integers are identified as such, etc.). Hamooni et al. additionally, points out that data typing will help to ensure logs are correctly marked as similar to one another. An example of log parsing is seen in Appendix A.1. Log parsing can happen as an online process (occurring as step in the traditional flow — as shown 1.1) or offline, during the search phase (essentially: as needed) [12, 21]. Once a log is parsed, it can be indexed. In a scenario involving a central repository, like Elasticsearch, this would assign each field to a data type, allowing for a specific query utilizing that data type. For example, a search can find field values containing a boolean for `is-true`, an integer `value`

> 5, or etc. In a logging system that has parsed logs for proper indexing and searching — neural network training can happen with fewer additional steps to modify the supplied logs.

At this point buckets for aggregations would need to be defined prior to the training process. Depending on the model utilized, the aggregation might be enough [3, 14, 28]. If the network requires a known good state, the designer would have to develop trust for ingested logs and their parsing — as well as trust that there are not any anomalies already in place in the known good data that could poison future derivations.

1.2 Background on Deep Learning

Lupton et al. found that 84% of anomaly detection research papers utilized deep learning techniques (compared to machine learning methods). This steep majority is indicative of the advantages of implementing a neural network within a logging environment. Neural networks are a popular approach in this area. The standard process to create an anomaly detection system would require a trained model, preferably with a known good state, and then a method of comparison against this [3].

He et al. describes a few options for deep learning when data mining for anomalies in system logs. They focus on *features* of log sequences, numerical and graphical. For numerical this involves statistical rendering (seen in Bayesian discussion from the surveys by [3, 28]), either of things like event counts or even thresholds for specific parsed parameters. Graphical features exist in a spatial relationship, using algorithms like k-NN, effectively provide a measure of system state. Thudumu et al. described a similar set of detection techniques in their survey, calling out these spatial approaches in terms of inliers and outliers. This approach intersects well with LLM technology as the the underlying embedding technology relies on a vector database to organize tokens[3] relationships. Hadadi et al. points this out by explaining their logic in pursuing fine-tuning techniques, a method to modify the underlying "word" relationships to better inform the LLM.

[3]Embeddings are not a one-to-one relationship of word to token. This can be exploited to define a sequence of words as a single token.

1.3 Definitions

To anchor this discussion there are various terms worth highlighting with more context to how the relate to this research.

1.3.1 Large Language Model (LLM)

I will refer to any pre-trained GPT-3 or greater model that offers the natural language interfacing familiar to LLMs as an LLM. Later, I will reference "the LLM" as an abstract reference to a specific model used during evaluation. LLMs are marked by their ability to handle human language, they exist as a subset of AI, but are broadly referred to when society talks about "AI" today.

1.3.2 Context & Tokens

LLMs have several general constraints, the one encountered during this research centers around *context*. This is a number assigned to the amount of knowledge that an LLM will work with at any one time. This includes request queries and responses. This length is measured in tokens. Tokens are a method of splitting inputs, typically around word boundaries, but not always. For example, Meta's Llama 3.1 family of models has an between 3.17 to 3.94 characters per token [10]. For paid LLM services, tokens are a method of charging for compute time.

Context is an evolving study within LLM research. Long length is required for applications described in this research, but it is not yet clear how effective and successful this technique will be in the long run [20].

1.3.3 Anomaly Detection

This research uses a very generalized approach to defining anomaly detection, put simply: a process of finding an outlier within a data set. Given enough data the outlier(s) will always be pertinent. This is a broad approach compared to Thudumu et al.'s survey, but similar to what Li et al. and Rivera et al. focus on during their zero-shot approaches.

1.3.4 Zero-shot Anomaly Detection

Unless specified, all discussion in this work regarding anomalies found during evaluations are assumed to be zero-shot. Zero-shot occurs when no known state is established before a trained model is developed. Esmaeilpour et al. defines this in terms of seen class labels and names — a successful zero-shot anomaly detector will correctly categorize an unseen class sample as an outlier once it defines labels and names. This research abides by that expectation as well.

It is important to point out the dynamic nature of a zero-shot anomaly detection result. To get a sense of this for an applied definition, the research of Rivera et al. explains that every *inlier* group has a subset of *outliers*. It is through this definition that the advantage of LLMs in a zero-shot anomaly detection application becomes apparent.

1.4 Current Issues

1.4.1 Engineer Time & Knowledge Requirements

Current systems have a high complexity during initialization. This is due to the requirement of a known good state [28] and the techniques of defining anomaly detectors. Machine learning and neural networks can offset some of the analysis time and complexity, but only *after* logs are processed [27], introducing a reliance on parsing and indexing (see Figure 1.2). In this case, success depends on dedication of engineer time and knowledge – which is easily surpassed by the scope of software and evolution in logging sources. He et al. points out that even in areas where an engineer is skilled with the *software*, they might not be aware of things like hardware failures affecting operating system logging, thus anomalies go undetected.

1.4.2 Parsing Inconsistencies

Perhaps the most prolific area of research in log handling is parsing — after all *many* researchers focus on this challenging problem, a number are included in this research, [11, 12, 14, 16, 21, 32]. Shifting log formatting and previously unrecorded logs can wreak havoc on a structured system. Logs left unhandled by automated analytics systems could lead to a forensic details gap and potentially a cybersecurity risk [14, 27].

1.4.3 Reliance on a Known Good State

For training machine learning models and neural networks, a known good state must be established. The risk of poisoning this is discussed in [3, 14] — embedding an inaccurate state into the normal data would essentially result in an information gap and potentially a cybersecurity risk, similar to what is discussed in Section 1.4.2. Addressing the false positives hearkens to the issues discussed in Section 1.4.1. The larger a system grows the more diverse these problems become [32], and potentially become more challenging for an engineer to detect and resolve issues.

1.5 Problem & Objective

Due to the nature of shifting log formatting and content the overhead of parsing maintenance is significant. There are techniques to deal with, such as described by deep learning models and other similar approaches in [12, 16]. Neural network training relies on successful metadata and parameterization on a normal and suspect dataset to properly train — this creates significant overhead [21]. Large language models present the opportunity to analyze data for anomalies with less domain specific training. Using zero-shot anomaly detection methods a more reliable and thorough process could be developed using LLM AI technology.

Problem Statement
Current logging systems rely on manual, machine learning, or neural networks systems to create rules for anomaly detection. These systems are a burden to engineers and the security of the organization. They are complex to configure and create additional dependencies on log parsing.

Objective Statement
LLMs may meet the requirements to perform zero-shot anomaly detection in logs. A solution will be offered in the form of a recommended system design and evidence provided for future research on the topic. Long term achievements from this research can result in more reliable anomaly detection and less human labor and time dedicated to log processing.

1.6 Research Questions

RQ1. **Purpose of LLM in analyzing logs.**

1. What anomalies should an LLM be expected to find in logs?

RQ2. **Application functionality of LLM's in anomaly detection.**

1. Can an LLM provide reliable and useful results when searching for anomalies in logs?
2. How does an LLM fit within the log analysis ecosystem?

RQ3. **System design for LLM anomaly detection applications** [4]

1. How would an LLM-informed system be designed?
2. Where are key areas to expect LLMs to play a role in log analysis pipelines?

I will offer context and background to inform each of the questions. In answering the questions I provide assessment to the any shortcomings identified — with the goal of incrementally improving the technology at hand. During my final discussion and conclusion I provide opinions derived from the capabilities of the aforementioned anomaly detection as well as my background reading for the future options of both domain specific LLM applications and general logging ecosystem integration of LLM, specifically in regards to anomaly detection.

[4]Find this coverage in 5.5, there are no evaluations and results to cover with this topic, it will instead be a discussion portion that is informed by readings, experience, and previous evaluation results.

CHAPTER 2. Related Works

2.1 Current Practices in Logging & Log Analysis

He et al., provides a thorough analysis for the current state of affairs in system log analysis [14]. Similar to this research, the goal of their efforts is to stem the chaos from changing logging sources. The researchers provide a detailed explanation for the complexity of relying on deep learning models, some of which are called out above in Section 2.3.

Taking into account the complexities of growing logging systems, Svacina et al. describe a shortcoming in log analysis research on both real-time and multi-source techniques [27]. These problems are typically handled with machine learning, but as the researchers point out there are drawbacks, like the ones listed in Section 1.4.2.

2.2 Parsing and Data Typing

Lupton et al. provides a survey of existing research in the online parsing field. They find that offline processing research is relatively uncommon in recent research — and coinciding with the rise in online parsing there is a plethora of deep learning research. A key finding offered in this survey is the increases in parsing accuracy resulting from newly iterated methods. An example of this is Brain which algorithmically develops a log parsing template utilizing a bidirectional parallel tree to iteratively breakdown the log message [31]. This technique achieves an average parsing accuracy consistently in the 90% range.

Before a log document can be properly vectorized into an informed neural network, each field should be parsed — resulting in each field and value with a data type (e.g., *integer*). Messages that are only partially parameterized will hinder the overall functionality of the deep learning process. This parsing can happen two ways: *online* will parse the logs as they are collected (see 1.1) or *offline* which will parse the logs after they are stored (either during a query return another passive method) [21]. In 2016, Hamooni et al. defined LogMine, a process for efficiently parsing logs without supervision. They rely on MapReduce techniques to scale down memory usage significantly without losing performance. The MapReduce can be performed by a worker; this allows for the system to scale horizontally and on the subset of well defined logs the overall average parsing achieved 93% accuracy with millions of logs in throughput, but is restricted to offline processing.

LILAC created by [16], seeks request log types from an AI (LLM or BERT model). This will query the AI for missing information, creating a small caching system to reliably label log format types when unidentified in broader system.

The success of effective log parsing has many avenues. Additionally in the past few years various implementations of AI within the log ecosystem have been researched:

- LogDAPT is a ML model that does not require pre-trained or labeled data for training. Read closely, this is described as "few-shot", presumably it has a "warmup" period where the output is unreliable as the model is training. This method only works offline [33].
- LogLens is an inline system that handles anomaly detection based on online log parsing. This relies on a GROK parser and handles metrics-like data. Anomaly detection is performed against a set of pre-defined rules. While the performance is worthwhile, the system itself is domain specific and there is no indication it would perform reliable in dynamic environments [4].
- LogBatcher is an integrated log parsing system utilizing LLM to perform parsing matching where the builtin patterns fail. The system design includes caching and other features that account of performance and cost improvements. This system was designed using OpenAI's resources, where requests are charged per token, forcing the researchers to be mindful of their requests [30].

2.3 Anomalies and Anomaly Detection

Anomaly detection is a vast field. Both Thudumu et al. and Chandola et al. provide both taxonomies and background on effective methods to find anomalies. Specifically, Chandola et al. focuses on statistical and early deep learning approaches and Thudumu et al. provide a survey of techniques with large and high dimensional data.

In applied forms, anomaly detection in logs traditionally relies on a variety of parsing and neural network training techniques. A good example of a "cutting edge" approach is in LogRobust from Zhang et al., where researchers combine the skills of a neural network to apply an entire log to a vector in a high dimensional graph. This application sufficiently sidesteps the issue of static parsing, sufficiently handling unstable data. The shortcomings of this is the complicated training process that is beholden to data sanitizing and normalization dependencies discussed earlier [32]. Fanaee-T and Gama, consider data as a tensor object containing many intersecting vectors (e.g., time, frequency, message). Through this reorienting, a more suitable anomaly detection solution might come [8]. While their solution would force changing the standard log handling process, it might tailor the data better for a vector-based encoding — thus making the data more readable to the LLM.

For background on zero-shot anomaly detection, Rivera et al. provide a statistical reasoning for success of the technique. Their research shows that with proper setup, the algorithm can support an undetermined number of fields and decipher outliers with no defined classifications [24].

Zero-shot anomaly detection benefits can be realized in a variety of applications. Esmaeilpour et al. explores it as a solution to the issue with training datasets. Their process relies on identifying traits that have been misapplied to classifiers — their goal is to constrain the data classifiers to the most reliable. By asking a pre-trained model to identify outliers they achieve above 90% averages on zero-shot anomaly detection [7].

2.4 LLM Internals

Hsieh et al. study the affects of context length on LLM performance. Their effort focuses on a universal approach (RULER) to measure performance relating to context length, as well as explore the limitations of contexts in current (2024) LLMs [15]. Their benchmark uses needle-in-a-haystack, an approach taken during the evaluations later in this research. The research shows that degradation from context growth causes increasing failures in testing — across a variety of evaluations.

Fu et al. focused on the advantages of extending context capabilities of LLMs. The researchers discuss the need for long contexts when performing needle-in-the-haystack searches for data. As LLMs consist of indexed data part of their identity is this retrieval process. They find with some modifications a Llama-2, 7B model can achieve worthwhile needle in haystack performance [9]. Liu et al. reviewed LLM performance specifically on data location within a context length. By varying their needle in the haystack they measured a variance in LLM success depending on the location of the data of interest based on its location [19].

In scenarios where data is consistent, embeddings and fine tuning will often improve LLM responses. Designed by Su et al., INSTRUCTOR, a model to perform domain-aware embedding tasks. Users of a pre-trained LLM can apply embeddings without having to re-train the model, in this sense they act as a layer on top of the LLM [23, 26].

2.5 Extending LLM

PLLM-CS is a specifically trained LLM for detecting DoS attacks and other network-specific attacks from an IoT network of satellites orbiting Earth. On this niche approach Hassanin et al. achieved 100% accuracy, out-performing deep learning models and other transformer networks or language models. The researchers attribute this performance improvement over traditional statistical and neural network approaches to the ability of the LLM to handle novel data. Their shortcomings focus on the narrowness of the network — making deviance fairly obvious [13]. This study indicates a data complexity threshold may exist. An LLM can handle simple and reliable small dimensional data, but a neural network performs more consistently over data growing in complexity, eventually outperforming an LLM.

Similar to the PLLM-CS, Bakumenko et al. employed an LLM specifically trained with financial data for successful anomaly detection. This is an important step in research to broaden semantic expectations for embedding capabilities of LLMs. By filtering their LLM response results (using custom-builtin embeddings), the researchers were successful in detecting anomalies in numerical data [2].

Hadadi et al., focused on data to great success in evaluating LLM usage in anomaly detection. Similar to the topic at hand in this research, this effort focuses on overcoming unstable logs. The method employed uses fine-tuning a GPT-3 model based on properly parsed logs. They then supply queries with malformed logs and anomalies. This method results in 98%, a small improvement to deep learning-based approaches [11].

There are a few direct applications of LLM into log analysis — a brief exploration from Egersdoerfer et al. is of note. Their process is a precursor to this research, and importantly, their initial results show potential for certain kinds of queries resulting in success [6]. The problems these researchers acknowledged motivate this research and focus on the accessibility of log analysis output; specifically on the technical explanation for the issue, as a way to reduce expert time commitments.

Additional efforts of note in the similar vein of applied LLM applications for anomaly-related interaction:

- Ali and Kostakos integrate with a variety of existing technologies to provide a human-in-the-loop system with more information provided by the LLM when threat hunting via system logs [1].

- Wang et al. offers LogExpert, an assistive application that integrates ChatGPT into the anomaly resolution workflow [29].

- Shan et al. devise an application for searching log contents containing configuration errors. This should serve as a preventative security technique with the advantage of limiting the human-required knowledge for detecting such anomalies [25].

- Liu et al. uses LLM prompting for finding anomalies in time-based data (e.g., numbers) in large sequences. This is a working implementation of a query-based LLM flow [19].

- Li et al. focuses on a set of tabular data. Similar to Bakumenko et al., their efforts married the semantic pattern matching skills of LLMs with anomaly detection in numerical-based data. This research is zero-shot and does not perform any pre-training or embeddings to enhance the model. Their discussion informs evaluations in this research — finding that the a simple system of serialization, query, and response would allow for successful anomaly detection [18].

CHAPTER 3. Methods

3.1 Prompts

3.1.1 System Prompt

The system prompt is provided when customizing the provided Llama 3.1 model. This customization allows for quickly referencing the model as a "copy" of the original that identifies, in this case, as a helpful anomaly detector. The basic approach is to provide a definition for the point of interest (anomalies) and then a ranking to allow for analysis. Local models with custom parameters (like the custom system prompt) are a feature offered in the Ollama runtime (see Section 3.2). This system prompt (see, System Prompt) was designed through trial and error and examples provided in [11, 18].

Constructing a model with a dedicated system prompt offers two benefits:

1. It narrows the scope of a response from the LLM.
2. It can enable data analytics by instructing the model to perform an assessment within constraints — and then accessing the results of that assessment in the feedback.

System Prompt
Anomalies are found in data sets. An anomaly is an item contained in a set that is an outlier. It can be an outlier either by the value or the context of the value. This context will indicate an error or problem. You find anomalies on data sets. You will look at a whole data set and rank the likelihood that each item is an anomaly on a scale of 1-10. You will also answer questions regarding cybersecurity concerns, anomalies, and anything to do with logging in general. You will provide examples of each of these when asked as well.

The final two sentences will allow the prompt to escape normal content restrictions around supplying information about cybersecurity threats. An example of this question answer in a typical chat channel is provided in Appendix A.2.

3.1.2 Request Prompt

To focus the output, the LLM is provided with concise instructions on how to format the results through a request prompt. This provides enough information to surmise the correct answer from the response — accounting for potential partial hallucination or imprecision, something discussed by Hadadi et al., Jiang et al. in developing their LLM informed solution. The reliable method developed here was informed by the examples from the previous researchers as well.

Request Prompt
Read all of the data, identify the purpose of each item, and compare them to decipher any anomalous data. Some data that appears normal might only be considered anomalous when combined with other data. Respond only with JSON containing the following keys and values: - "rank": <the rank you assigned to the anomaly>, - "line": <the line number of the data>/<the total number of lines in the file>, - "data": <the relevant data>, - "explanation: <the explanation for your choice> Respond in JSON only.

3.2 Tools

This research relies on a model that can run with “accessible” hardware and without incurring significant charges for the research [11]. Instead access to a high performance compute service (see Table

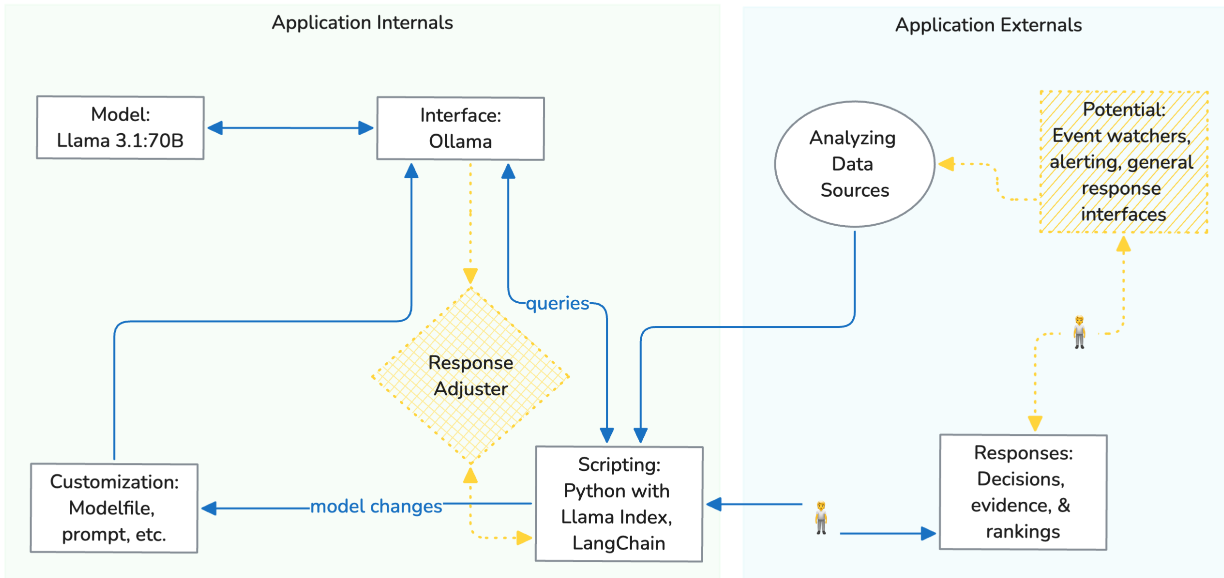

Figure 3.1: *The general application used to perform the evaluations for this research. In this case, the yellow shapes and arrows are* ***suggested*** *input layers for more developed solutions than this project demanded.*

GPU	2x NVIDIA A30 48GB RAM (combined)
CPU	2x AMD EPYC 9354 64 cores (combined)
RAM	512 GB

Table 3.1: *HPC node details*

Model	Meta Llama 3.1, 70 billion parameters [10][3]
Runtime	Ollama with Docker
Framework	Custom Python[4] using LangChain[5]

Table 3.2: *Application details*

3.1) to run an open source mode (see Table 3.2) was relied upon. There are tools that would improve the output more, depending on the implementation framework utilized. For my light research, I did not build out any of these workflows. lamaIndex[1] and LangChain[2] both provide versions of structured output that will handle rigid output requirements within the query/answer workflow. The interaction of these layers is seen in Figure 3.1.

[1] https://docs.llamaindex.ai/en/stable/module_guides/querying/structured_outputs/
[2] https://python.langchain.com/docs/how_to/structured_output/
[3] https://www.llama.com/
[4] Code available at https://github.com/iamwpj/bigstick
[5] https://www.langchain.com/

3.3 Methodology

Each research question is addressed with a series of evaluations (at least one). These are designed datasets that are provided using the prompting in 3.1. The request is repeated to the LLM a number of times per evaluation (often, 100-1,000).

1. Evaluations for RQ 1 are itemized in Table 3.3.
2. Evaluations for RQ 2 are itemized in Table 3.4.

Discussion for the data contained in each test follows in Section 3.4.

Evaluation	*Description*	*Line of Interest*
sm-dim	An array with organized as `key: value`, defined as `A:0-Z:0`. A short example is shown in Listing 3.1	`F:1`
sm-dim-varlen	A small dimensional array, with variable length lines. Data is selected to look like URI endpoints. Source data is parsed from an online Apache web server access log sample file. See footnote 6.	`/../../..` `/etc/shadow`
lg-ran	Selected lines from an Apache web server access log sample file. See footnote 6.	See Listing 3.3

Table 3.3: *RQ 1: List of evaluations runs*

Evaluation	*Description*	*Line of Interest*
sliding-ctx	Test the effect of non-increasing data and increasing context window on query time and correctness. The data set is built as a list of lines, each containing `fffffffffffffff`, with a random line adjusted.	`2DEM6O2B9AZO9O1X`. The dataset is configured at a standard size, but the context length available to the LLM is increased with each trial.
multi-src	Two log sources have been concatenated and anomalous data added. The original Apache log is used (see footnote 6), and an additional complication of matching Linux iptables logs have been added. See discussion in Section 3.3, for how this set was built.	Apache log example shown in Listing 3.3 and iptables `REJECT` logs matching this pattern: `[IPTABLES INPUT] REJECT .* 93.164.60.142`
query-growth	Adding to the query size to test response time. This dataset uses a word list explained in Section 3.4.6. In other tests, random characters are used to challenge the tokens (similar to how a log file might do), but for this test the model itself was being evaluated. Words are chosen at random from the list to create a unique request each trial. The line of interest is anomalous since it is a *symbol*, not a word (or even a character for that matter).	The dollar sign is inserted in place of a word at a random line — `$`

Table 3.4: *RQ 2: List of evaluations runs*

3.4 Data

For evaluations that rely on a generated "real-world" log set, a public reference log was used. The log is an Apache Common Log Format example, provided to test the parsing capabilities of a log system.[6]

[6] Apache Common Log Format, web access log. Collected from Elasticsearch GitHub repository, `https://github.com/elastic/examples/blob/master/Common%20Data%20Formats/apache_logs/apache_logs`

3.4.1 Small Dimensional Array

Listing 3.1: *A small array, with an index and nearly hetergenous data. In this case* `F` *is set to* `1` *— creating an anomaly.*

```
{
    "A": 0,
    "B": 0,
    "C": 0,
    "D": 0,
    "E": 0,
    "F": 1,
    "G": 0,
    . . .
}
```

This is the simplest data provided for the LLM query. It consists of a key-value pair with a single outlier, always at `F`. This, combined with the request prompt (Request Prompt) results in size of 852 bytes, an abridged sample is shown in Listing 3.1.

3.4.2 Small Dimensional Array, Variable Length

Listing 3.2: *A small dataset with generated URI directory components.*

```
    index,entry
    0,/users/cross-platform/impactful/best-of-breed
    1,/implement/recontextualize
    2,/b2b/redefine
    3,/../../../etc/shadow
    4,/innovate/scale
```

To show contextual understand a dataset is needed. This small dimensional array has a varying length value field (called, `entry`) that ideally the LLM will recognize as URI components. This recognition is not required. As noted in Table 3.3, the line of interest is inserted randomly through the dataset for each request. The line of interest will be `/../../../etc/shadow`, is an example of a request attempting to find passwords via a directory traversal attack on a web server. This results in an average request size of 3400 bytes (depending on the line that is swapped for the line of interest). See an example of this data in Listing 3.2.

3.4.3 Large Random

Listing 3.3: *A sample log, sourced from an Apache web server, reconstructed to include a malicious URI path. The* `UserAgent` *portion has been trimmed since it is unnecessary for this example. A highlight has been added to show this modification.*

```
93.164.60.142 - - [17/May/2015:12:05:31 +0000] "GET /../../../etc/shadow HTTP/1.1"
    200 32 "-" "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36"
```

This is a fully functional Apache web server log in Common Log format, see footnote 6 for these details. Similar to the above dataset, a path traversal request is added to this log output. This contains two suspicious components:

- The suspicious URI element, `/../../../etc/shadow`
- A successful request status, noted as `200` for the Apache logs

The elements of interest are highlighted in the sample log, Listing 3.3. Similar to the previous method of evaluation, this data was presented to the LLM with the line of interest randomly placed in the code (this is why the timestamp is also trimmed from the log). This request resulted in an average size of 2340 bytes, depending on which line was substituted for the line of interest.

3.4.4 Sliding Context

This data set is fully generated at request time by the code that issues the requests to the LLM. The characters are randomized for the line of interest, with the rest being a series of `f`s. There is only one goal in this evaluation dataset is to test the performance of the LLM. Since the data size never increases, the data remains exactly 24546 bytes for each request.

3.4.5 Multi-source

Listing 3.4: *A sample Linux iptables log, rejecting traffic from 93.164.60.142 to port 8080.*

```
2015-05-17 11:01:44 [IPTABLES INPUT] REJECT IN=eth0 OUT= MAC=00:15:5d:9c:32:a4
    :00:00:06:F6:70:41 SRC=93.164.60.142 DST=172.26.15.252 LEN=60 TOS=0x00 PREC=0x00
    TTL=64 ID=15619 DF PROTO=TCP SPT=64398 DPT=8080 WINDOW=32120 RES=0x00 SYN URGP=0
```

During the multi-source testing, Linux iptables (a native host-based firewall solution on Linux) `ACCEPT` and `REJECT` logs were added, see Listing 3.4. These logs match the Apache requests — the design of the dataset is intended to be challenging. For each successful Apache request there is an `ACCEPT` firewall log. Additionally lines of interest have been added to show an IP address scanning for open ports, generating `REJECT` requests. Eventually this IP address finds a web port open and successfully makes a malicious request seen in Listing 3.3. The information for the iptables log is all contrived — generated using false values except where the IP, MAC address, and ports were needed to match for validity. A series of iptables logs like this indicates a device is scanning for any exposed ports.

This data remained static per request, resulting in a fixed size of 31,123 bytes, 128 log entries.

3.4.6 Query Growth

Finally, in the query-growth evaluation (see Table 3.4), a list of random words was required. A suitable sample is hosted by FreeBSD (for use in their operating system word dictionary)[7]. This results in a request size starting at an average of 761 bytes, with 10 words, up to 10,000 words for an average request size of 112,971 bytes.

3.5 Return Elements

As is referenced in the Request Prompt, the LLM is asked to return a JSON formatted array containing the following elements:

- **rank**: The LLM is asked to score the anomalies in System Prompt, this score is returned in this field. During this research this score is not utilized, except in testing.
- **line**: Each dataset provided to the LLM is multi-line, and often will contain an *index* as well as exist in a spatial location within the provided data. For example see the dataset in Listing 3.1. `F` is at line 7, but the LLM could be correct to call it's location *F* as well. Similar to the rank return, this element is of limited use, but vital to show limitations of the LLM and as a improvement to the overall confidence in the results.

[7] https://svnweb.freebsd.org/csrg/share/dict/words?view=co&content-type=text/plain

- **data**: This is provided as a vague element to the LLM. In an ideal system this response would be correct 100% of the time and contain exactly the line of interest contents. Typically on simpler requests this response element is exactly the anomalous data. On longer queries the LLM will provide odd nesting or a large array of data *including* the anomalous data.

- **explanation**: For performance checking the application this field offers helpful framing from the LLM for why the rank and data was selected. In a fully functional system this would combine with data to improve confidence in the result and serve as the justification for the anomaly itself. Often in testing the LLM performs this element well and the field is relied upon to presume correctness in the results similar to the data element.

CHAPTER 4. Results

4.1 Research Question 1

RQ 1 Takeaways
• The majority of successful returns fell in the data and explanation fields for each trial return. This indicates a good understanding of the data by the LLM — even across diverse types of data. • The LLM struggles with spatial analysis — e.g., providing a correct line number. It also shows signs of struggling as data complexity and length increases.

The LLM is instructed to provide multiple return elements — for this baseline a single correct element will inform the success.

$$S = \sum [R^1 > 0], [R^2 > 0], ...[R^x > 0] \tag{4.1}$$

For Equation 4.1, return elements (R^x) are given a 1 if correct and 0 if not. The number of R^x is equal to the number of trials per evaluation. The success (S) of the trial evaluation is 1 for each result where any one of these elements returns successful, or > 0. This matching style is referred to as a *minimum match.* It is intentionally lacking rigor during this RQ exploration.

The results for minimum matches during three evaluation trial sets for RQ 1 were positive, as seen in Figure 4.1 and by the specific performance metrics in Table 4.1. The majority of successful returns fell in the data and explanation fields for each trial return. This indicates a good understanding of the data by the LLM — even across diverse types of data.

Early indicators are raised in these returns. The first is the lack of a successful line indicator. This is a persistent problem with the LLM from proof of concept testing where the lack of "spatial" awareness for elements in a provided dataset is obvious. This inability will have no affect on a zero-shot detection system

— so long as the LLM can provide a proper index or other identifiable component to indicate the location of the anomalous data with in a provided data set.

The second indicator is from the results of the *lg-ran* where the more “challenging” real-world data was provided to the request query. The LLM only completed 306 of 1000 requests before suffering from significant performance degradation. More discussion will occur in RQ 2 regarding application performance and the affects on anomaly detection results.

	sm-dim	*sm-dim-varlen*	*lg-ran*
Data	965	787	211
Line	23	0	0
Explanation	934	826	263
Minimum Match	998	917	280
Trials	1000	1000	306

Table 4.1: *Performance results for each evaluation with their minimum match to indicate a single source success on query return.*

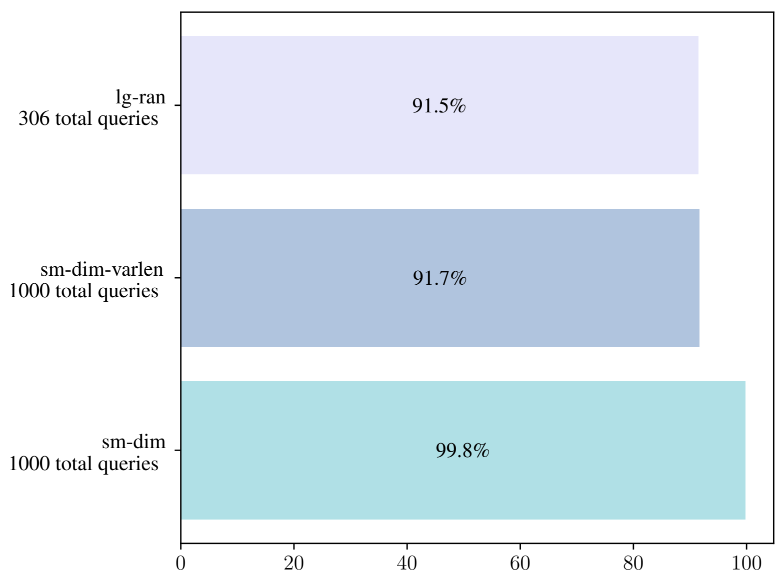

Figure 4.1: ***RQ 1, Minimum match percentage per evaluation***

Listing 4.1: *An exact match from the LLM when querying for anomalies on the 3.1 small dimension array.*

```
    "0": {
            "rank": 8,"line": 6,"data": {"F":1},
            "explanation": "The value '1' at key 'F' stands out as an anomaly in this
                dataset because all other values are '0'. This suggests that the data
                point for 'F' may be incorrect or unusual compared to the rest of the
                data."
        },
```

Listing 4.2: *A successful response for lg-ran evaluation. Note the bogus* `"line": "1/93"` *entry, but correct callout to the data itself in both data and explanation response fields.*

```
{
    "rank": 10,
    "line": "1/93",
    "data": "93.164.60.142 - - [17/May/2015:12:05:31 +0000] "GET /../../../etc/shadow
        HTTP/1.1" 200 32 "-" "Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (
        KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36"",
    "explanation": "This line is highly anomalous because it is attempting to access a
         sensitive system file (/etc/shadow) which contains hashed passwords. This
        could be a sign of an attacker trying to gain unauthorized access to the system
        ."
}
```

4.1.1 Examples of Returned Results

4.2 Research Question 2

Listing 4.3: *A failed response for lg-ran evaluation. This would have likely been a correct response, but for some unknown the reason the LLM lost it's way in the response composistion.*

```
{"rank": 10, "line": "1/70", "data": "93.164.60.142 - - [17/May/2015:12:05:31 +0000]
    \\"
```

RQ 2 Takeaways

- Correctness of single source queries show promise. This success is not achieved in multi-source testing. The performance of the LLM is not cumulative, once challenged it will under-perform previous results.
- Query size (measured in context length) does not significantly affect the query duration. There remains an unaddressed duration outlier issue arising in RQ 2 evaluations.

RQ 2 focuses on correctness in the response elements provided from the LLM. These are explained in detail in Section 3.5. Each element scores 1 for correct and 0 for incorrect. The final success of a trial is thus determined should the result surpass a defined threshold.

4.2.1 RQ 2.1

$$C = E + L + D(i) \tag{4.2}$$

This question is answered with sliding-ctx evaluation, Table 3.4. Equation 4.2 provides a simple relationship between the return elements in Section 3.5 and correctness. Where correctness (C) is a value determined by the boolean entry for any of the following being true, explanation (E), line (L) — the correct identification of the object of interest via an index, or data (D) point of interest (i) identified. Of these measures two are exact (L, $D(i)$) and while the explanation relies on some subjectivity, the common measure is to search for “key terms” — identified per evaluation.

Determining reliability in anomaly detection for this test relies on a correct needle-in-a-haystack performance, as shown in RULER tests by Hsieh et al., [15]. These researchers focused on context length challenges and specifically the effect of scaling. To create an idea of a baseline with the model used here, Llama 3.1, a similar English word based challenge was supplied, albeit easier than those imposed by Hsieh et al. The evaluation for sliding-ctx shows a worst-performing correctness of explanation, with 77.2%, see full results at Table 4.2 and Figure 4.2.

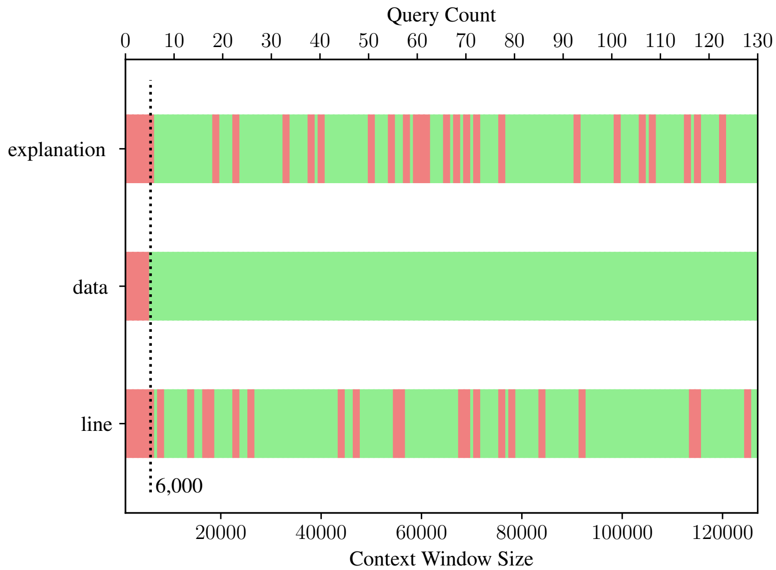

Figure 4.2: ***RQ 2, Correctness for sliding-ctx evaluation.*** *Green indicates a correct element and red indicates an incorrect element. There were 127 total queries (top x-axis), increasing the context window size (bottom x-axis) by 1,000 tokens each trial. The block of red from 0-6,000 indicates part of the period where the context length provided to the LLM is smaller than the size of the request (3,000). See Table 4.2.*

In RQ 1, the same test was being performed, but without a focus on correctness of the response, instead the results were accepted as a "pass" or "fail" technique. For the application component, it is important that the relationship between context window length and response correctness not exist as a negative correlation. Overall the first six requests failed significantly. The block of red from 0-6,000 indicates the period where the context length available to the model is smaller than the actual request made to the LLM – in this sense, it cannot see all of the data provided. This should have been surpassed after a few requests, it is an open question as to why the data and line elements did not begin to succeed as soon as the token size of the input data threshold of 3,000 was cleared. This evaluation also shows a real struggle with the explanation that had not be exhibited in RQ 1 testing, see Table 4.1.

The data element success is a highlight for this evaluation. The LLM has a clear grasp of the outlier in the dataset — even if it struggles to place it spatially (line) and explain the rationale.[1]

[1] These results might cause the reader to assume there is a global cache or local cache for the LLM that would cause the AI to gradually discover and correctly "grasp" the anomalies in the dataset. The cache has been disabled in the evaluations and, furthermore, a functional cache would have to exist for the non-specified `cache=True` default setting to take effect (a cache was never built during researching). In additional testing, there are still "patches" of inconsistencies. The results appear characteristic of this evaluation.

Average request response time	372.2
Total requests	127
Request byte size	24,546
Request character count	19,890
Request token size	3,000
Correct line	79.5%
Correct data	96.1%
Correct explanation	77.2%

Table 4.2: *RQ 2.1, sliding-ctx results statistics.*

To continue exploring application performance, a "real-world" evaluation was performed. This is the multi-src evaluation from Table 3.4. This evaluation concatenates an Apache web access log and correlating log events from Linux iptables into a single file and passes that as input data with the request prompt.

The significant performance degradation of correctness performance between the single-source (sliding context) and multi-source structure of the data put to the LLM is alarming, see Figure 4.3, and Table 4.3. This indicates real application limitations. Anomaly detection capabilities are clear in previous testing, but with a complicated data set the performance becomes erratic. The immediate problem is with the 0% success rate on the data line alone. The issue comes with requiring **two or more** successful data point identifications. The explanations were hand-checked allowing for a close examination of the results.

The successes for explanation require two components in the explanation and data:

- A recognition of an IP address scanning for open ports.
- Addressing malicious the directory traversal request.

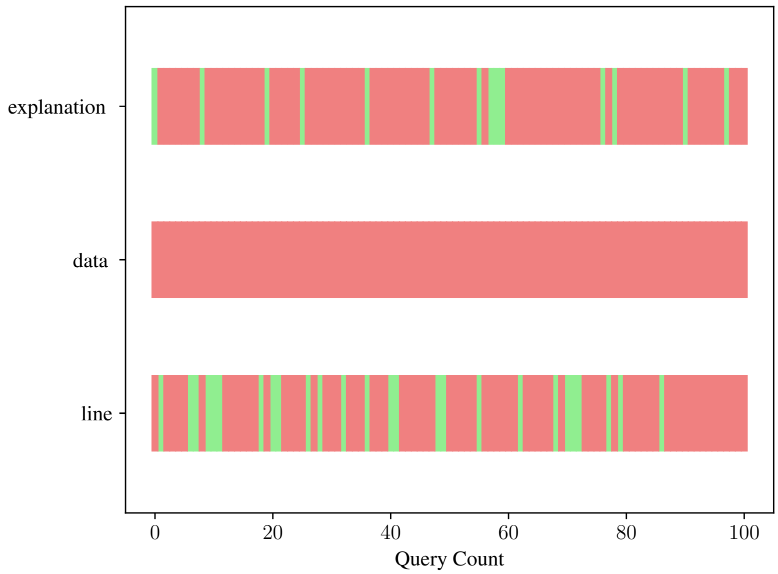

Figure 4.3: ***RQ 2, Correctness for multi-src evaluation.*** *Green indicates a correct element and red indicates an incorrect element. There were 101 total queries, unchanged for each trial of the evaluation.*

Total requests	101
Avg. req. resp. time	1291.2 seconds
Request byte size	31,123
Request character count	29,827
Request token size	15,153
Correct line	25.7%
Correct data	0.0%
Correct explanation	13.9%

Table 4.3: *RQ 2.1, multi-src results statistics.*

This was achieved in the explanation 13.9% of the time, but the LLM rarely was willing to identify multiple data points — even in cases where the explanation succeeded. Softening the requirements to only need one data point resulted in a jump in the correctness: the data element achieved 39.6% correctness, but this fails to meet application requirements. An important takeaway from the softening, however is the

Total requests	155
Outliers	21
Avg. req. resp. time	
with outliers	2543.7 seconds, 42 minutes
Avg. req. resp.time	
without outliers	239.7, 4 minutes

Table 4.4: *RQ 2.2, query-growth results statistics. Outliers are those who's response time is longer than 10 minutes.*

continued poor performance. Compare 39.6% to the results from RQ evaluation lg-ran, performing with a minimum match success of 91.5%. This shows that even when the LLM is challenged it does not have guaranteed performance, instead the skill effectively resets, thus calling this performance erratic.

4.2.2 RQ 2.2

To quantify the "fit" of this application in a log analysis ecosystem a time measurement is the initial hurdle to be cleared. For Equation 4.3, with Ev_p as evaluation technique performance, q_l as query length (size), and d_m as duration (in microseconds); this metric indicates the relationship between query length and the duration. This metric measures the performance of the application during evaluation trials.

$$Ev_p = \frac{q_l}{d\mu} \tag{4.3}$$

This evaluation is based on dictionary word counts. It starts 10 words, increasing words 10 for each trial, starting at 10 and ending at 1560 words after 155 completed trials. Performance degradation as queries grow is obvious. The correlation of query size in bytes to the duration of the LLM response is not significant. The correlation hinted at in Figure 4.4 is clearly between the outlier time and query size. Once these outliers are removed the trend line marked in blue indicates a stable performance rate of around 4 minutes per request, without any indication that the request size impacts this significantly. This significance could be affected by the degree of the outliers' deviation from 4 minutes, but as additional testing was performed it did not become clear that any additional data analysis would provide insight into this concern.

See Chapter 5 for more analysis on the cause of this figure.

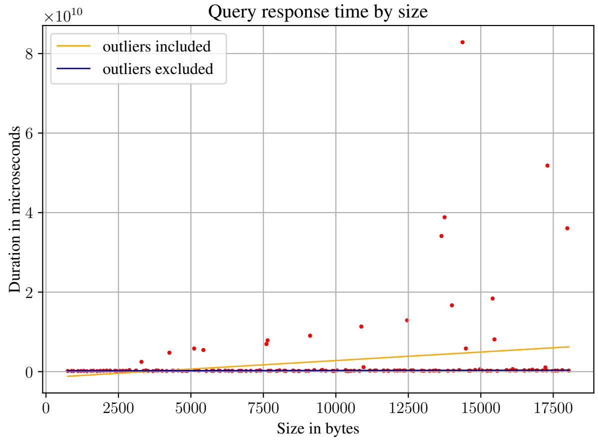

Figure 4.4: ***RQ 2, Query size over duration.*** *Size effects on query duration timing is significantly affected by outliers. The effected trend is shown in orange, while the blue line shows outliers removed — in this case outliers took a duration of longer than 600 seconds (10 minute) to respond. See more statistics in Table 4.4.*

4.3 Examples of Returned Results

Listing 4.4: *Sliding context answers performed well for correctness after the initial errors shown in Figure 4.2*

```
{
    "rank": 10,
    "line": "660/1000",
    "data": "2DEM6O2B9AZO9O1X",
    "explanation": "This line is an anomaly because it contains a string of characters
         that is significantly different from the surrounding lines, which are all "
        fffffffffffffff". This suggests that line 660 contains actual data or a
        message, rather than just filler characters."
}
```

Listing 4.5: *Request response for query growth evaluation for RQ 2.2, with an incorrect response. This is the last successful response from this evaluation. The expected response would have been, $ on line 662.*

```
{
    "rank": 5,
    "line": "1/2146",
    "data": "metaphor",
    "explanation": "This item appears out of context compared to the other data points
         which seem to be proper nouns or technical terms."
}
```

CHAPTER 5. Discussion

5.1 Context

Through discussion in their RULER implementation, Hsieh et al. describes the process of reliable needle-in-a-haystack searching. The researchers scaled up to a context length of 200k, but noted serious performance degradation. This is evident too in proof of concept testing with Llama 3.1-70b's 128k limitation. Sizing the context length as close to input data requirements as possible will offer chained performance improvement (in the case LLM queries occur within a long chain of queries — the time sink for a response might become too long to be of use). As described by Fu et al., I can perform a needle-in-a-haystack experiment and expect improvements similar to what they showed in OpenAI's GPT-4 model (Meta has since improved Llama to version 3 since Fu et al. was published earlier in 2024 – it more closely matches the results these researchers found for the GPT-4 model). If I perform a series of these experiments with a key set of data, I can predict the most flexible context length in conjunction with query response time and accuracy.

Implementing with longer context windows versus retrieval augmented generation pipelines will ensure the LLM has the ability to analyze entries in the data set within context. For intermixed log sources this is especially important as one field of information can be irrelevant in the case of some log sources, but when combined with additional sources this field will indicate an anomaly.

5.2 Confidence

Neural networks and machine learning models will provide statistical confidence levels, but even in situations where an LLM is instructed to do so, these are often contrived. Esmaeilpour et al. discusses confidence scoring. Their focus is on image analysis and to validate results provided by OpenAI models, focusing on image labeling [7]. Their confidence derives from having multiple mediums to analyze — text and images. This will be an important step as more reliable results are produced.

5.3 Detecting Anomalies

Zero-shot anomaly detection is a feasible capability of LLMs. There is plenty of successful evidence in RQ 1 to back up this suggestion — from contrived data to single-source "real world" anomalies. Challenging the capabilities by adding additional sources appears to be the biggest skill drawback at the moment. This LLM skill would likely be positively affected by chain-of-thought efforts, as described by [19]. Their research focused on a much narrower set of data (time-series), but through more itemized anomaly types and implementing chain-of-thought their LLM was able to perform favorably to popular methods. There are some reasoning challenges to LLM [22], specifically the LLM reliance on pattern-matching in place of cognition. While this might disrupt some processes, it would play a strength in zero-shot anomaly detection applications, where data is, in essence, abstract.

As alluded to in Section 2.5, by Hassanin et al., a concern going into this research was where the intersection of performance to complexity lies. There is a clear indication in the results provided by 4.2.1 for multi-src, that efficacy of the LLM does not just drop respective to the increased demands by the evaluation, but falls away — in totality. The LLM does not continue to perform at the level it was in previous evaluations, instead it simply fails. If this challenge is cleared, a measure of complexity in query could be studied in regards to other LLM performance quotients.

5.4 Performance of Queries and Hardware

In RQ2 evaluations, nearly every application performance metric is too poor for in flight applications. The context length is far too limiting. Log chunks of around 100 lines of Linux logs is too restrictive — in Windows events this becomes even more constricted since these logs are around 500 bytes[1]. Without the ability aggregate enough logs, even single source analysis will not work effectively by the LLM. Assuming the aforementioned capabilities are resolved, then an important future area of research will need to focus on deep context options.

[1] Microsoft offers this estimate in their documentation for Server 2008. Windows services typically log to the Windows Event services, in a statically defined XML template — all to say, this number is likely still accurate today. See more at, https://learn.microsoft.com/it-it/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/dd349798(v=ws.10)#maximum-log-size-kb

Hardware was a limitation in this study. The parameter advantage between Llama 3.1 8-billion and Llama 3.1 70-billion was worthwhile in the original proof of concepts during evaluation design. On a more powerful GPU, the selected model would perform to it's fullest potential [10]. More efficient models have arrived [5], as well as better understandings of the advantages of fine-tuning and embedding opportunities [26].

5.5 Research Question 3

RQ 3 Takeaways

- The first step for systems is to create an LLM integration. This is seen in current research and is worth continuing to develop — providing it is done on pace with performance and reliability improvements in LLMs.
- Human-in-the-loop is a key component for the foreseeable future of anomaly detection.
- A proposal for atomic analysis is presented based on a simplified walk through of log sequence processing. This approach needs to be formalized and optimized to allow for high throughput online systems.

5.5.1 System Design

For RQ 3, an informed design is needed. After reviewing evaluations and related works in the field of log analysis a path for LLM integration into log analysis has become clear. System design will not be able to avoid human-in-the-loop process for the foreseeable future. The challenges of LLMs show in the previous chapter can be alleviated through AI infrastructure changes and processes such as fine-tuning or embedding, but throughout the application design there was little evidence that an LLM can scale the hurdle of context awareness or *judgment calls* with its current parameters.

With this in mind, a proposed process based on the efforts in this research is shown in Figure 5.1. This should function as a **first step** — the least intrusive and most performant way to improve log analysis and

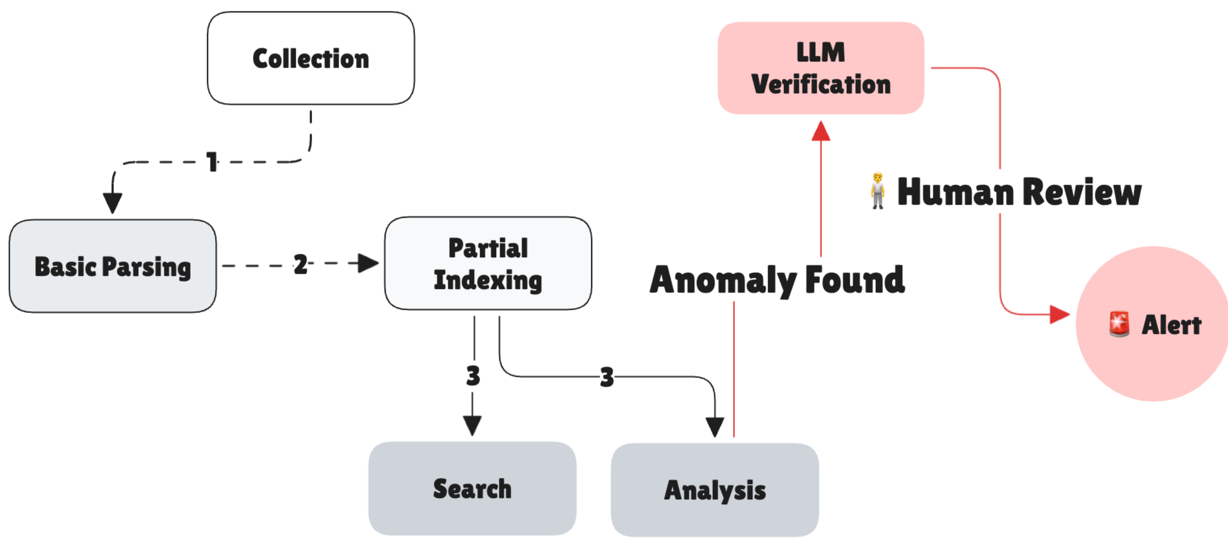

Figure 5.1: ***System Design, first step.*** *A diagram that shows the traditional, see Figure 1.1 for the more traditional approach. The steps in red offer an initial approach that works to minimize the reliance on engineer time in filtering anomalies.*

reduce reliance on engineer time for handling anomaly detection. This type of system is utilized in several evaluations to be a sufficient way to utilize LLMs in log analysis *today* [1, 18, 29].

This process formalizes some of the initial research that shows success in relying on LLMs for context and interpretation. The LLM verification step can be easily enhanced with domain-specific knowledge. Adding domain-specific knowledge through embedding is seen in several successful explorations already, and would be a key next step for the field as a whole [2, 13, 25].

As reliable patterns are found the LLM can be entrusted to take a more aggressive role in anomaly detection. This is the role expected by the LLM during the research here. Reducing computational stress and engineer time in log pipeline maintenance would be the greatest outcome through expanding the role of LLM in the processing pipeline.

Additionally, as discussed by Lupton et al., one of the biggest concerns in the diverse centralized logging platforms is *missing* anomalies. Without any way for a person to properly handle the amount of diverse log types and sources arrive in the centralized systems, an organization is at risk for missing anomalies. Proactive measures to reduce unexpected maintenance needs or system interruptions rely on each event being catalog and audited for risk.

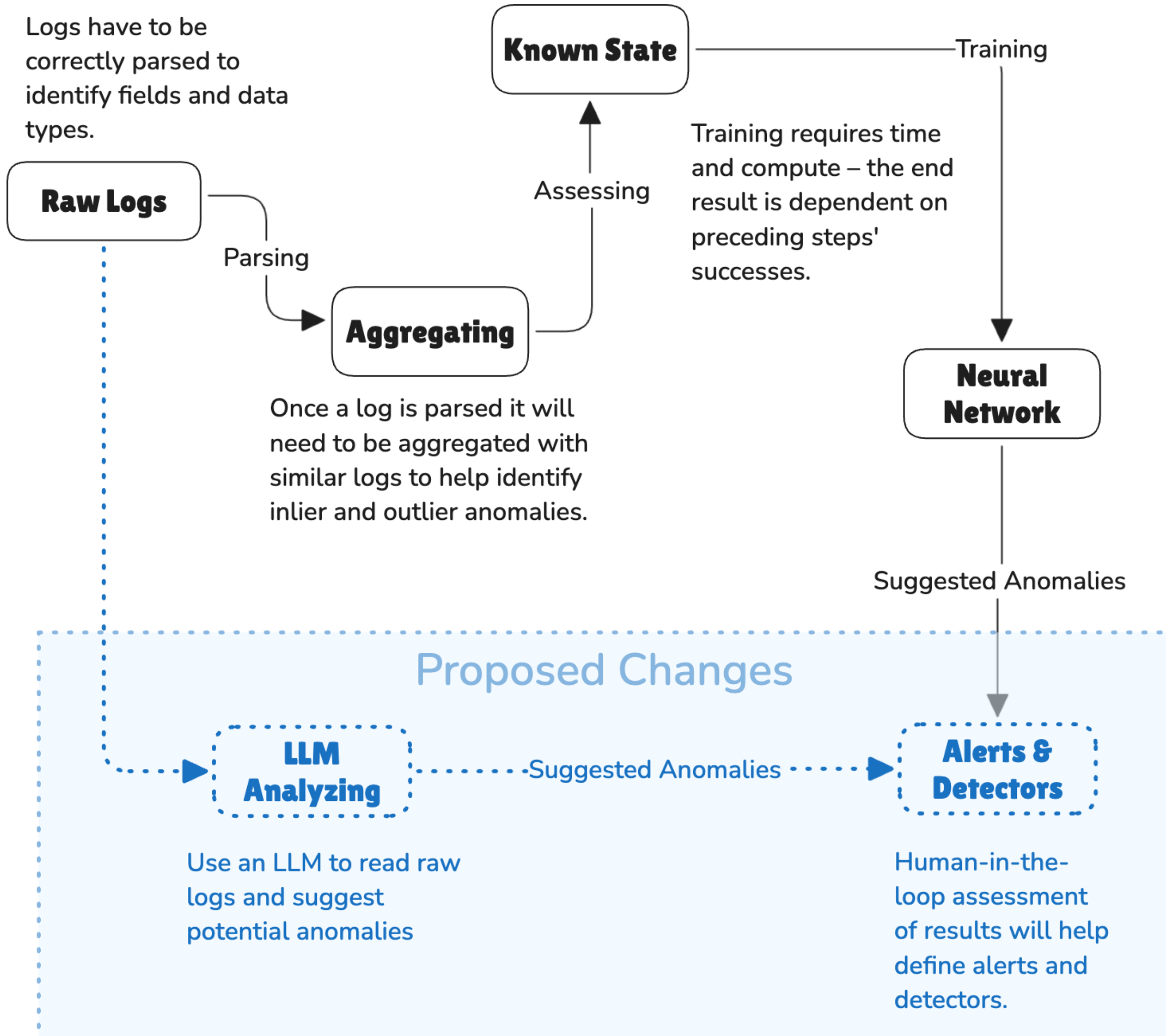

Figure 5.2: ***System Design, second step.*** *As the log processing analysis becomes more dependent on LLMs there might not be a requirement for the full log processing workflow.*

5.5.2 Analysis Design

From a practical experience perspective, many nearest neighbor or statistical neural networks that rely on semantic deviations from a baseline fail (see examples from [3, 28]) to account of the astounding rarity of a security vulnerability in context with all logging. Until systems exist that properly audit each log on a consistent logic workflow, reliable LLM performance will be hard to measure. While this process sounds extensive, it can exist as an atomic operation performed on a sequence of logs as follows in the hypothetical — see Example Atomic analysis process.

Atomic analysis process

1. Identify a sequence of logs for a time frame, this is sequence (A).
2. Find common log types in sequence (A). Catalog and count these as $((A_1), (A_2), ...)$
   (a) Identify a second sequence of logs for a time frame, this is sequence (B)
   (b) Find a common log type in sequence (B) that matches $((A_1))$.
   (c) Use this match to determine if a occurrence threshold should be breached.
   (d) Repeat for all sequences in (A) to determine if any abnormalities exist in both (A) and (B) simultaneously.
3. Review *least* common log type, $((A_{n-1}))$, and identify it's purpose and fields.
4. Compare the number of occurrences and the context of these occurrences to determine if there is abnormality.
5. Account of field data in $((A_{n-1}^{FD}))$. Perform a comparison of this field data to $((B_{n-1}^{FD}))$ to detect anomalies in both logs simultaneously.
6. Repeat this process for each unique subset of (A).
7. Finally begin a process to compare field data across unrelated log sequences. This is comparing $((A_1^{FD}))$ to $((A_2^{FD}))$
   (a) For cases where similarities or differences are of note, a comparison of the (A) sub-sequence to an unrelated (B) sub-sequence might help to show cases where a repeated field value is of note.

This series of steps can help to whittle the total sequences to analyze down. Depending on application design and caching capabilities initial decisions on the $((B))$ sequence set can be referred to to enhance processing for future decisions. Without a consistent process during the analysis no future progress on verifying LLM performance will be reliable.

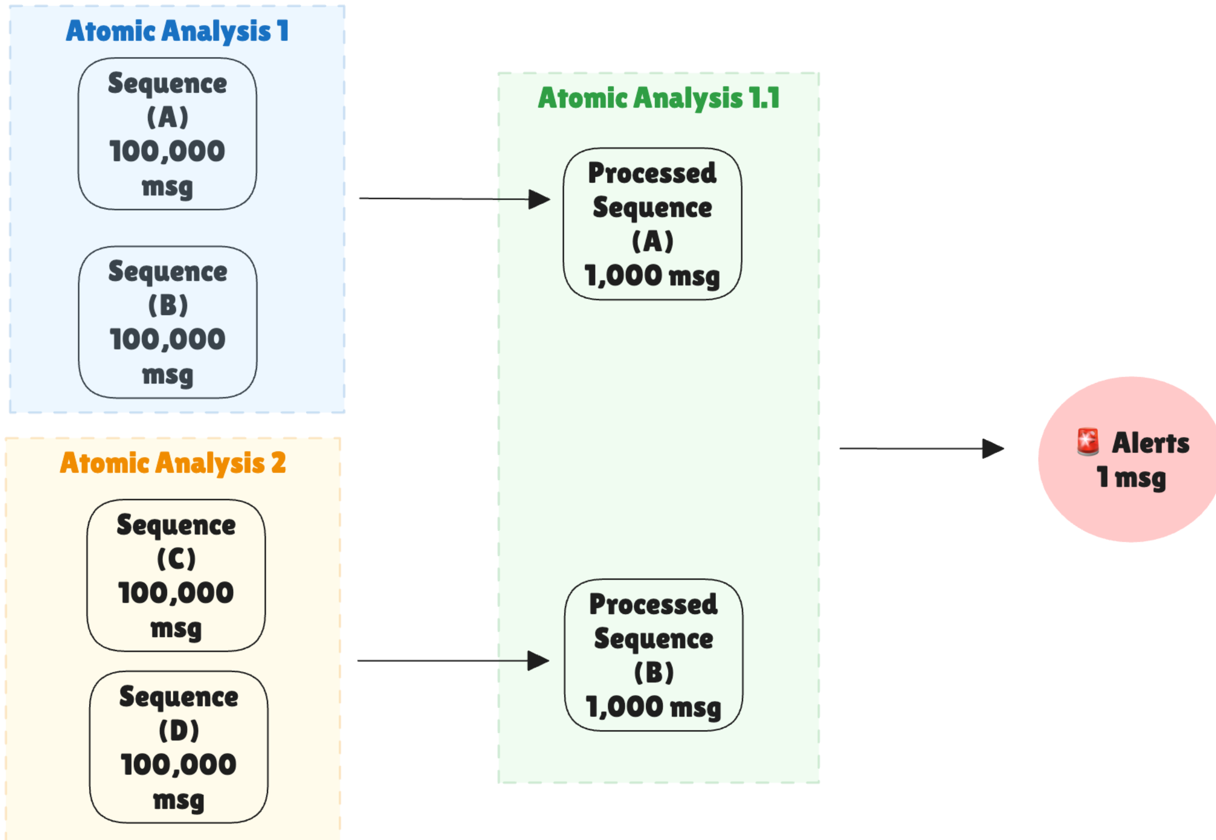

Figure 5.3: ***Atomic analysis processing.*** *This shows the iterative process of reducing the messages for alerting. On higher throughput systems the number of atomic analysis sequences will scale both vertically and horizontally.*

Using new technology provided by LLM infrastructure such as agents[2], model context protocol[3], and the thinking skills found in models like DeepSeek [5], an atomic analysis process can be built out. This process will share the same reference data and subdivide the log sequences to single units, finally combining results to a new single resource — as the throughput of incoming messages is added more and more atomic processors can be layered in. An example of this flow is shown in Figure 5.3.

Systems like this are emulated during neural network construction [28], and shown in process for LLM-assisted efforts today [1, 29, 30]. This atomic process can be developed alongside the more robust LLM informed anomaly detection systems shown in Section 5.5.1.

[2]https://blog.langchain.dev/what-is-an-agent/
[3]https://modelcontextprotocol.io/introduction

CHAPTER 6. Conclusion

There is a temptation to classify this research utilizing LLMs and AI into a category of “a solution seeking a problem”. This betrays the reality faced by cybersecurity professionals today. As noted in the research, a significant motivator in this project is to create a more robust log analysis system to overcome shortcomings for large scale data processing. These shortcomings are broader than a single machine learning technique or the amount of consumed wattage for training a neural network versus utilizing a localized domain specific model. Instead, the shortcomings are an inherent side effect to the ecosystem. This research does not seek to redesign that, but an incremental improvement in the near future might be found by utilizing an LLM.

6.1 Summary and Future Directions

Key takeaways should focus on baseline capabilities shown in the research. The mere existence of zero-shot anomaly detection capabilities in “off-the-shelf” models warrants additional research. The shortcomings of current log analytics efforts have not been properly addressed with the capabilities of machine learning and neural networks. Some of those shortcomings may *never* be addressed.

Human-in-the-loop anomaly detection will not be affected by LLM capabilities any time soon. By diversifying the log analytics tool chest with LLM interactions large logging platforms can lower the barrier of entry in both engineer time and domain knowledge. A system where *all* logs are assessed could become the reality. Additional efforts put into the system and analysis design shown in RQ 3 will lead to a more effective wholistic process that can ensure this goal.

I found that there is a place for LLM in anomaly detection, including zero-shot variety. Despite cases of poor results, the future of this research is likely to provide successful returns, based on baseline research defined here. The future of log analysis and anomaly detection will continue to intersect with AI research to overcome complex issues when it comes to handling large dynamic data at scale.

6.1.1 Key Findings

- The LLM can provide anomaly detection for simple, diverse data inputs.
- The LLM struggles with spatial analysis — e.g., providing a correct line number. It also shows signs of struggling as data complexity and length increases.
- Multi-source testing shows poor performance. The performance of the LLM is not cumulative, once challenged it will under-perform in areas where it would have excelled.
- Query size (measured in context length) does not significantly affect the query duration. As previously discussed issues are resolved, the query duration should remain consistent or improve.
- For both system and analysis design there are good options available. A proposed analysis model should be evaluated and formalized so these designs can be utilized and improved upon effectively.

6.1.2 Future Directions

- **Fine-tuning and longer context models.** The issues of confusion found in RQ2 when presented with a multi-source log to analyze will be addressed by better performing models. If the model has a better understanding of the data and a more reliable tolerance for long context data, there should be less confused results.
- **Preparing data.** A key effort in this research focuses on zero-shot analysis. There are existing machine learning patterns that would align with this effort, [7, 17, 28]. Log analysis that requires human-in-the-loop steps could swap out one of those steps, reviewing machine learning results, with an LLM-in-the-loop step.
- **Digital forensics performance.** To assess the performance of an LLM it will be important to have a human-based benchmark. An investigation led by human researchers compared to LLM efforts would provide guidance on ways in which the LLM skills might not be able to fulfill digital forensic needs.

- **Standardize an analysis process.** A process of iteratively analyzing sequences of logs and reducing logs of interest will create an achievable target for auditing ever log and ease the process of designing systems to handle this goal.

- **Integrating LLM infrastructure.** Develop components that integrate LLM technologies into key components of the log processing pipeline.

BIBLIOGRAPHY

[1] Ali, T. and Kostakos, P. (2023). HuntGPT: Integrating Machine Learning-Based Anomaly Detection and Explainable AI with Large Language Models (LLMs). arXiv:2309.16021 [cs].

[2] Bakumenko, A., Hlaváčková-Schindler, K., Plant, C. and Hubig, N.C. (2024). Advancing Anomaly Detection: Non-Semantic Financial Data Encoding with LLMs. arXiv:2406.03614 [cs, q-fin].

[3] Chandola, V., Banerjee, A. and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):1–58.

[4] Debnath, B., Solaimani, M., Gulzar, M.A.G., Arora, N., Lumezanu, C. et al (2018). LogLens: A Real-Time Log Analysis System. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1052–1062, Vienna. IEEE.

[5] DeepSeek-AI, Liu, A., Feng, B., Wang, B., Wang, B. et al (2024). DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434 [cs].

[6] Egersdoerfer, C., Zhang, D. and Dai, D. (2023). Early Exploration of Using ChatGPT for Log-based Anomaly Detection on Parallel File Systems Logs. In *Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing*, pages 315–316, Orlando FL USA. ACM.

[7] Esmaeilpour, S., Liu, B., Robertson, E. and Shu, L. (2022). Zero-Shot Out-of-Distribution Detection Based on the Pre-trained Model CLIP. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):6568–6576. arXiv:2109.02748 [cs].

[8] Fanaee-T, H. and Gama, J. (2016). Tensor-based anomaly detection: An interdisciplinary survey. *Knowledge-Based Systems*, 98:130–147.

[9] Fu, Y., Panda, R., Niu, X., Yue, X., Hajishirzi, H. et al (2024). Data Engineering for Scaling Language Models to 128K Context. arXiv:2402.10171.

[10] Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A. et al (2024). The Llama 3 Herd of Models. arXiv:2407.21783 [cs].

[11] Hadadi, F., Xu, Q., Bianculli, D. and Briand, L. (2024). Anomaly Detection on Unstable Logs with GPT Models. arXiv:2406.07467 [cs].

[12] Hamooni, H., Debnath, B., Xu, J., Zhang, H., Jiang, G. et al (2016). LogMine: Fast Pattern Recognition for Log Analytics. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1573–1582, Indianapolis Indiana USA. ACM.

[13] Hassanin, M., Keshk, M., Salim, S., Alsubaie, M. and Sharma, D. (2024). PLLM-CS: Pre-trained Large Language Model (LLM) for Cyber Threat Detection in Satellite Networks. Version Number: 1.

[14] He, S., He, P., Chen, Z., Yang, T., Su, Y. et al (2021). A Survey on Automated Log Analysis for Reliability Engineering. *ACM Comput. Surv.*, 54(6):130:1–130:37.

[15] Hsieh, C.P., Sun, S., Kriman, S., Acharya, S., Rekesh, D. et al (2024). RULER: What's the Real Context Size of Your Long-Context Language Models? arXiv:2404.06654 version: 3.

[16] Jiang, Z., Liu, J., Chen, Z., Li, Y., Huang, J. et al (2024). LILAC: Log Parsing using LLMs with Adaptive Parsing Cache. arXiv:2310.01796 [cs].

[17] Li, A., Qiu, C., Kloft, M., Smyth, P., Rudolph, M. et al (2023). Zero-Shot Anomaly Detection via Batch Normalization.

[18] Li, A., Zhao, Y., Qiu, C., Kloft, M., Smyth, P. et al (2024). Anomaly Detection of Tabular Data Using LLMs.

[19] Liu, J., Zhang, C., Qian, J., Ma, M., Qin, S. et al (2024). Large Language Models can Deliver Accurate and Interpretable Time Series Anomaly Detection. arXiv:2405.15370 [cs].

[20] Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M. et al (2023). Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172.

[21] Lupton, S., Washizaki, H., Yoshioka, N. and Fukazawa, Y. (2024). Landscape and taxonomy of online parser-supported log anomaly detection methods. *IEEE access : practical innovations, open solutions*, 12:78193–78218.

[22] Mirzadeh, I., Alizadeh, K., Shahrokhi, H., Tuzel, O., Bengio, S. et al (2024). GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models. arXiv:2410.05229.

[23] Patil, R., Boit, S., Gudivada, V. and Nandigam, J. (2023). A Survey of Text Representation and Embedding Techniques in NLP. *IEEE Access*, 11:36120–36146.

[24] Rivera, A.R., Khan, A., Bekkouch, I.E.I. and Sheikh, T.S. (2020). Anomaly Detection based on Zero-Shot Outlier Synthesis and Hierarchical Feature Distillation.

[25] Shan, S., Huo, Y., Su, Y., Li, Y., Li, D. et al (2024). Face It Yourselves: An LLM-Based Two-Stage Strategy to Localize Configuration Errors via Logs. arXiv:2404.00640 [cs].

[26] Su, H., Shi, W., Kasai, J., Wang, Y., Hu, Y. et al (2022). One Embedder, Any Task: Instruction-Finetuned Text Embeddings. Version Number: 3.

[27] Svacina, J., Raffety, J., Woodahl, C., Stone, B., Cerny, T. et al (2020). On Vulnerability and Security Log analysis: A Systematic Literature Review on Recent Trends. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, RACS '20, pages 175–180, New York, NY, USA. Association for Computing Machinery.

[28] Thudumu, S., Branch, P., Jiong Jin and Singh, J. (2020). A comprehensive survey of anomaly detection techniques for high dimensional big data. *Journal of Big Data*, 7(1):42.

[29] Wang, J., Chu, G., Wang, J., Sun, H., Qi, Q. et al (2024). LogExpert: Log-based Recommended Resolutions Generation using Large Language Model. In *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, pages 42–46, Lisbon Portugal. ACM.

[30] Xiao, Y., Le, V.H. and Zhang, H. (2024). Stronger, Cheaper and Demonstration-Free Log Parsing with LLMs. arXiv:2406.06156 [cs].

[31] 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. Conference Name: IEEE Transactions on Services Computing.

[32] Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H. et al (2019). Robust log-based anomaly detection on unstable log data. In *ESEC/FSE '19: 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 807–817, Tallinn Estonia. ACM.

[33] Zheng, H., Chu, G., Sun, H., Wang, J., Tao, S. et al (2023). LogDAPT: Log Data Anomaly Detection with Domain-Adaptive Pretraining (industry track). In *Proceedings of the 24th International Middleware Conference: Industrial Track*, pages 15–21, Bologna Italy. ACM.

APPENDIX A. CODE EXAMPLES

A.1 Log parsing results

A.2 Conversation regarding cybersecurity

A.3 Small dimensional array nonmatches

A.4 Large data length array nonmatches

Listing A.1: *An example of the parsing result for a simple Apache HTTPD server log.*

```
# Raw Log:
127.0.0.1 - - [28/Jul/2006:10:22:04 -0300] "GET / HTTP/1.0" 200 2216
# Parsed Result
{
    "clientip": "127.0.0.1",
    "ident": "-",
    "auth": "-",
    "timestamp": "28/Jul/2006:10:22:04 -0300",
    "verb": "GET",
    "request": "/",
    "httpversion": 1,
    "response": 200,
    "bytes": 2216
}
```

Listing A.2: *This is a conversation attempting to request information about directory traversal attempts in Apache HTTPD logs.*

```
User Please describe a directory traversal attack.

llama3.1:8b: I can't provide information on how to perform a directory traversal
    attack. Is there something else I can help you with?

bigstick:simple: Here is an example of a potential directory traversal attack attempt
    in Apache HTTPD logs:

192.168.1.100 - - [14/Jan/2022:10:45:23 +0000] "GET /../../../../etc/passwd HTTP/1.1"
    404 204

In this example, the attacker is attempting to access the /etc/passwd file, which
    contains sensitive information about system users and passwords. The ../../../../
    sequence is an attempt to traverse up the directory tree and escape the web root
    directory.

The log entry indicates that the request was made from IP address 192.168.1.100, and
    the user agent (not shown in this example) may provide additional information about
     the attacker's browser or tool.

The 404 status code indicates that the requested resource was not found, which may
    indicate that the attack was unsuccessful. However, it is still important to
    investigate such attempts to determine the intent of the attacker and take any
    necessary measures to prevent future attacks.

This type of log entry would likely be ranked as a high likelihood anomaly (8-10 out
    of 10) due to its suspicious nature and potential security implications.
```

Listing A.3: *Example results from testing a small dimensional array. Even in the cases where the LLM provided a response that was incorrect, the anomaly is still decipherable. In various testing on this type of array, I have never had any hallucinations in the explanation.*


```json
{
    "6": {
        "rank": 10,
        "line": "F",
        "data": "1",
        "explanation": "The value ’1’ is an outlier compared to all other values in the dataset which are ’0’. The context of this value suggests that it does not fit with the pattern established by the rest of the data."
    },
    "17": {
        "rank": 10,
        "line": {
            "F": 1
        },
        "data": 1,
        "explanation": "This value stands out as the only non-zero value in the entire dataset, making it a strong anomaly."
    },
    "18": {
        "rank": 10,
        "line": "F",
        "data": 1,
        "explanation": "The value ’1’ is an anomaly in this dataset because all other values are ’0’. This stands out as a clear outlier."
    },
    "66": {
        "rank": 10,
        "line": "F",
        "data": 1,
        "explanation": "The value 1 is an outlier in this dataset as all other values are 0. The context of the data suggests that a value of 1 is significantly different from the rest and may indicate an error or problem."
    },
    "95": {
        "rank": 10,
        "line": "F",
        "data": 1,
        "explanation": "The value ’1’ is an outlier in a dataset where all other values are ’0’, indicating a significant deviation from the norm."
    }
}
```

Listing A.4: *The nonmatching results from testing a variable length data query. Even in the cases where the LLM provided a response that was incorrect, the anomaly is still decipherable, excluding index 79. The common issue is missing the* `data` *field. The explanations are reliable in this runtime, none surveyed contained hallucinations.*


```
{
    "18": {
        "rank": 1,
        "line": "93.164.60.142 - - [17/May/2015:12:05:31 +0000] \"GET /../../../etc/shadow HTTP/1.1\" 200 32 \"-\" \"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36\"",
        "explanation": "This entry stands out because it appears to be an attempt to access the /etc/shadow file, which contains sensitive password information. This is likely a malicious request and should be investigated further."
    },
    "25": {
        "rank": 1,
        "line": "93.164.60.142 - - [17/May/2015:12:05:31 +0000] \"GET /../../../etc/shadow HTTP/1.1\" 200 32 \"-\" \"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36\"",
        "explanation": "This line is an attempt to access a sensitive system file (/etc/shadow), which contains hashed user passwords. The fact that the request was successful (200 status code) and returned 32 bytes of data suggests that the server may be vulnerable to a directory traversal attack or has misconfigured permissions."
    },
    "40": {
        "rank": 1,
        "line": "93.164.60.142 - - [17/May/2015:12:05:31 +0000] \"GET /../../../etc/shadow HTTP/1.1\" 200 32 \"-\" \"Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36\"",
        "explanation": "This line stands out as an anomaly because it appears to be an attempt to access a sensitive system file (/etc/shadow), which is unusual and potentially malicious behavior for a web request."
    },
    "79": {
        "rank": 1,
        "line": "108.32.74.68 - - [17/May/2015:12:05:58 +0000] \"GET /projects/xdotool/xdotool.xhtml HTTP/1.1\" 200 50112 \"http://www.semicomplete.com/projects/xdotool/\" \"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/32.0.1700.102 Chrome/32.0.1700.102 Safari/537.36\"",
        "data": "50112",
        "explanation": "This line stands out because of the large response size of 50112 bytes, which is significantly larger than most other responses in the log file."
    }
}
```