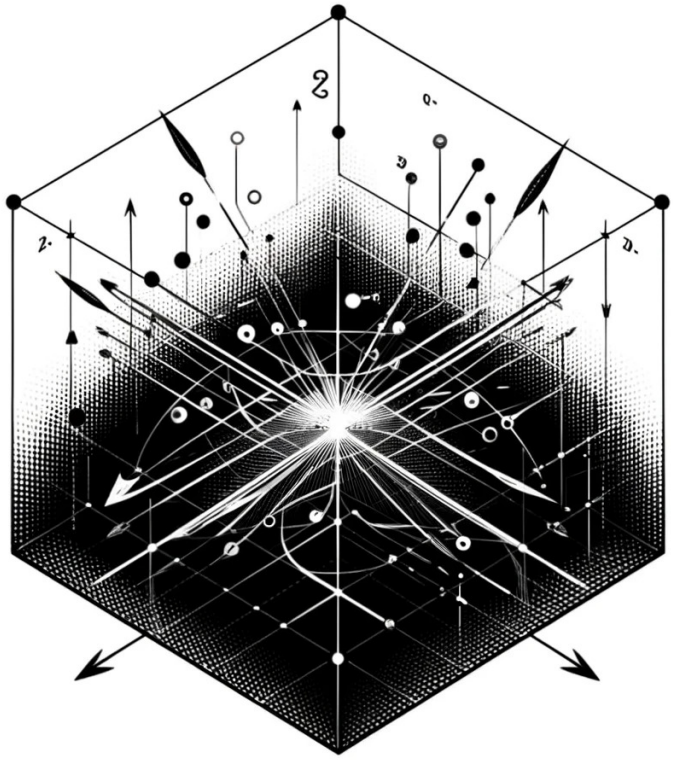
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Leveraging Large Language Models For System Log Analysis

Fault Troubleshooting Radio Units Using Log Data

Master's Thesis in Data Science & AI

JACOB NIR, WILLIAM SNÄLL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Leveraging Large Language Models For System Log Analysis

Fault Troubleshooting Radio Units Using Log Data

JACOB NIR, WILLIAM SNÄLL

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Leveraging Large Language Models For System Log Analysis
Fault Troubleshooting Radio Units Using Log Data
JACOB NIR
WILLIAM SNÄLL

Supervisor: Morteza Haghir Chehreghani, Department of Computer Science and Engineering, Chalmers University of Technology
Examiner: Peter Damaschke, Department of Computer Science and Engineering, Chalmers University of Technology

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Visualization of high dimension vector space
Typeset in LaTeX
Gothenburg, Sweden 2024

Leveraging Large Language Models For System Log Analysis
Master's Thesis in Data Science & AI
JACOB NIR, WILLIAM SNÄLL
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis investigates the application of large language models (LLMs) for system log analysis, specifically focusing on fault troubleshooting in radio units using log data. The primary objective is to enhance the efficiency and accuracy of system monitoring tools through state-of-the-art AI techniques. The research explores the utilization of retrieval-augmented generation (RAG) frameworks and parameter-efficient fine-tuning (PEFT) methods to process and summarize log data. By employing pre-trained models such as Llama2, Llama3 and Mistral, the study evaluates different implementations to summarize segments of logs as well as extracting relevant information from them.

The findings demonstrate that LLMs can significantly automate and improve the analysis of system logs, providing insights and facilitating easier troubleshooting. Additionally, the study examines the impact of enriching chatbot input data with contextual information, leading to substantial performance improvements in specialized domains. Despite the promising results, the research acknowledges limitations related to the quality and structure of log data and the need for source-specific refinements in context-enrichment methods.
The contributions of this thesis are twofold: it presents a viable approach to leveraging LLMs for easier system monitoring and highlights the critical role of context in enhancing chatbot functionalities. Future research directions include integrating more advanced models, fine tuning existing models and exploring other state-of-the-art methods to optimize retrieval-augmented generation pipelines.

Keywords: AI, Artificial Intelligence, Large Language Models, Generative AI, System Logs.

Acknowledgements

We would like to express our sincere gratitude to Ericsson for providing us with the opportunity to undertake our master's thesis within their organization. The resources and support made available to us have been key to this project.

We extend our heartfelt thanks to Morteza Haghir Chehreghani for his great guidance on the academic and theoretical aspects of our research. His expertise and insights were crucial to the project.

Additionally, we are deeply grateful to Amit Kumar Panda for his exceptional supervision, continuous feedback, and innovative ideas that greatly improved the quality of our thesis.

Jacob Nir, William Snäll, Gothenburg, June, 2024

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AI	Artificial Intelligence
ANN	Artificial Neural Network
BERT	Bidirectional Encoder Representations from Transformers
CPU	Central Processing Unit
DB	Database
DCW	Dynamic context window
FN	False Negative
FP	False Positive
GPT	Generative Pretrained Transformer
GPU	Graphical Processing Unit
GQA	Grouped Query Attention
LLama	Large Language Model Meta AI
LLM	Large Language Model
LoRA	Low Rank Adapters
NER	Named-Entity Recognition
NLP	Natural Language Processing
PEFT	Parameter-Efficient fine tuning
QLoRA	Quantized Low Rank Adapters
RAG	Retrieval Augmented Generation
SME	Subject Matter Expert
TN	True Negative
TP	True Positive
TPU	Tensor Processing Unit

Contents

List of Figures

1
Introduction

In a world with fast developing technology, our modern society has become dependent on communication and connection. In the field of telecommunications and network infrastructure, the efficient and accurate troubleshooting of faults in radio units is crucial for ensuring uninterrupted communication services. Radio units are a critical component of mobile networks, and any malfunction or downtime can lead to substantial service disruptions and economic losses on an individual, as well as on societal level. Currently, enterprises predominantly rely on the expertise of domain specialists for the interpretation of logged data. This often involves the utilization of methods developed in-house, leading to varying levels of reliability and demanding a considerable amount of time and effort. This challenge arises from the complex and non-human-readable nature of log data. To address this issue, this thesis outlines a research project aimed at leveraging the capabilities of large language models to assist domain experts in the fault troubleshooting process by obtaining, summarizing and analyzing data from Ericsson's system log files.

1.1 Problem Statement

This thesis will develop a framework that lets users upload an Ericsson log file and use a chatbot to either retrieve specific data, or summarize parts of the log. The research questions for this project are:

How can pre-trained large language models (LLMs) be used to analyze and summarize system log files?

Specifically, we aim to determine if these models can effectively extract useful information from log data and provide concise summaries of log file sections.

How can we enhance chatbot performance by enriching the input data with more context?

We will explore various methods to add context and pre-process log data, aiming to improve the chatbot's understanding of domain-specific terms and the complexities of niche areas.

1.2 Limitations

This section contains the limitations that are set for the project to make it feasible and relevant to the research question.

1.2.1 Running Time

This report did not focus on the implementation's computational efficiency or running time due to its early stage in the research. Given the project's emphasis on exploratory research, optimizing for speed was not a primary concern. This allowed us to prioritize finding a well working framework and thorough documentation of the new methodologies and performance metrics that would be more relevant at the later stages of development.

1.2.2 Hardware

The project was limited to using specific hardware available at Ericsson. Therefore the models that were chosen between might have had better alternatives that were unavailable to us during the course of the project. The hardware limitations could be boiled down to:

- Model size could not exceed $<\approx$ 14 GB of size
- Only make use of services deployable with Ollama
- Use Nvidia A2 TPU

1.2.3 Assumption of Element Extraction

Furthermore an assumption was made on the format of the input data. Since there was already a method for extracting the different elements of a log file. Even though the importance of this was recognized in the project, this was a key assumption to increase the time spent on work relevant to the research questions.

1.3 Ericsson Log / Proactive Log

The data we were working with in this project was Ericsson's proactive logs. These were log files that are extracted from a node where units were raising alarms. Proactive meant that they were used to prevent the need of actually fetching any units, but rather analyzing it on site. These log files could consist of around the order of tens of thousands lines of shell commands and printouts. Among these lines, there were various information about hardware, software and events. The log files contained data on all the units of the alarming node.

Some information was presented in table and time series format, which was the focus in this project. Due to its richness in information compared to other elements of the log.

2
Background

This chapter traces the development of language models from their birth to modern advancements. It begins with the early rule-based systems, moves through the era of statistical methods, highlights the impact of Graphics Processing Units, and discusses the revolutionary introduction of transformer models and the shift to open-source language models.

2.1 Rule-based Systems

The history of language models dates back to the 1950s when researchers began exploring rule-based systems to process language. In 1966 an MIT computer scientist, Joseph Wiezenbaum, developed ELIZA, an early natural language processing (NLP) computer program. ELIZA was designed to simulate conversation by using pattern matching and substitution, particularly in the context of Rogerian psychotherapy. ELIZA's structure allowed it to respond to users' inputs by reflecting their statements back as questions, imitating the style of a therapist. The program demonstrated how computers could engage in pseudo-conversations with humans, raising questions about the nature of communication and the potential for machines to simulate human-like interactions [2].

In the following decades until early 1990s, the NLP systems remained mainly rule-based. Researchers worked on improving how machines understand language, creating more complex rule-based systems, hoping to better handle the nuances of human speech. However, these systems faced challenges. The researchers found it difficult to fully grasp the context and the many ways we use language.

2.2 Statistical Methods

When the disadvantages of rule-based systems became clear, a transformation in favor of statistical methods started to take shape. This change signaled a turning point in the development of language models, as scientists started looking for patterns and structures in data instead of just predetermined rules. Machine learning approaches gained increasing traction in the discipline in the late 1990s with the introduction of strong tools like Support Vector Machines for diverse NLP tasks and Hidden Markov Models for speech recognition [4] [8].

N-Gram models were developed during this time and had a significant impact on statistical language models. These models, despite their simplicity, were effective in predicting the probability of a word by considering the sequence of preceding words. This approach improved the contextual understanding in language models. Additionally during this time period Convolutional Neural Networks (CNNs) were introduced, while mainly used in image processing it also found applications in NLP, specifically in text classification. This period experienced advancements in AI and neural network architectures [3] including the Perceptron [1], Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) units and CNNs, collectively advancing the field of NLP and deep learning.

The development of statistical models and machine learning algorithms in the 1990s and early 2000s marked the beginning of a new era in NLP research. These technologies, including Probabilistic Context-Free Grammars [12] and the introduction of word embeddings such as Word2Vec [6] allowed researchers to perform more complex language evaluations. The transition to statistical models established a foundation for the future growth of neural network-based models in natural language processing [9] .

2.3 The First Graphics Processing Unit (GPU)

The first GPU, NVIDIA's *GeForce 256* was released in 1999, and it was a turning point for the field of NLP and the creation of language models. Before this, the complexity and size of language models that could be efficiently trained and used were limited by the compute power of Central Processing Units (CPUs). The training of more advanced models was made easier by the GPU's exceptional parallel processing capabilities, which created new opportunities for NLP research and development. The great advancements in NLP and language models that followed were made possible by this breakthrough.

The following years, the capabilities of language models rapidly advanced due to increases in computer technology and new algorithmic developments. The enhanced computational capacity allowed researchers to create models that could process language at previously unheard-of levels. For NLP tasks, methods like deep learning and neural networks became more practical, which resulted in advances in speech recognition, machine translation, and text production. These developments highlights the value of computer power in expanding the field of language modeling possibilities.

2.4 Transformer Models

The Transformer model was first presented in 2017 [15] and completely changed the way language models are built. The Transformer model, which stands out for

having a self-attention mechanism, markedly increased language processing effectiveness and efficiency, allowing models to more accurately represent the complexity of language. This breakthrough sparked the creation of a new class of language models that can understand and produce writing that is similar to that of humans [15].

NLP saw further change with the introduction of Generative Pre-trained Transformer (GPT) models, beginning with OpenAI's GPT-1. These models showed ability to produce language that was relevant to the context, respond to queries, and even produce content that looked like it was written by a human. GPT models became larger and more complex with each iteration, reaching models such as GPT-3 that demonstrated the ability of large language models (LLMs) to execute a range of NLP tasks with no task-specific training [16] [18].

2.5 Open-source Language Models

Another important turning point in the development of language models was the publication of LLaMA (Large Language Model Meta AI) [27] and the shift to open-source models. It standardized the field of NLP by making advanced LLMs available to a wider audience, enabling researchers, developers, and businesses to shape and use these models in a variety of applications. The next wave of advances in language understanding and generation has been made possible by this open-source approach, which has encouraged creativity in NLP.

3

Theory

3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) is one of the cornerstones in modern AI and Machine Learning. The concept is inspired by biological neural networks similar to animal brains. They are essentially computational models that mimic the way our brains draw input from neurons to produce an output to other neurons. They are the foundation of deep learning and thereby play a great roll in large language models and modern conversational AI.

The concept of ANNs dates back to the 1940s, with the pioneering work of Warren McCulloch and Walter Pitts, who proposed a simplified model of the neuron, the basic unit of the brain. This model laid the groundwork for further research [7].

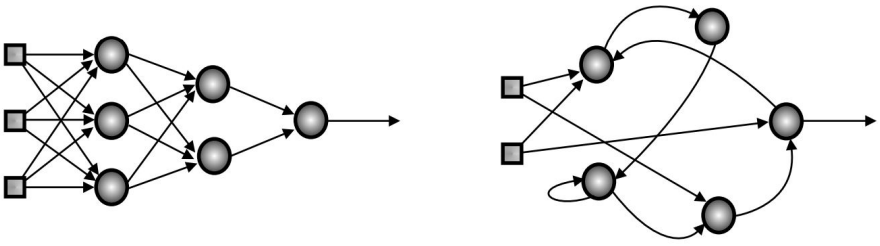


Figure 3.1: A layered neural network (left) and a recurrent neural network (right). The signal flow is from left to right, except for the recurrent connections in the right panel. Image from *Introduction to neural networks* [7]

A neural network consists of layers of neurons, where each neuron in one layer connects to neurons in the next layer through weights. The basic components include:

- **Inputs** (x): Data fed into the network.
- **Weights** (w): Parameters that scale input data within the network's neurons.
- **Threshold** (θ): An additional parameter added to the weighted input before passing it through the neuron's activation function, ensuring that even when all inputs are zero, the neuron can still activate.
- **Activation Function** (σ): A mathematical function applied to a neuron's weighted sum of inputs (plus a threshold), introducing non-linearities into the network, allowing it to learn complex patterns.

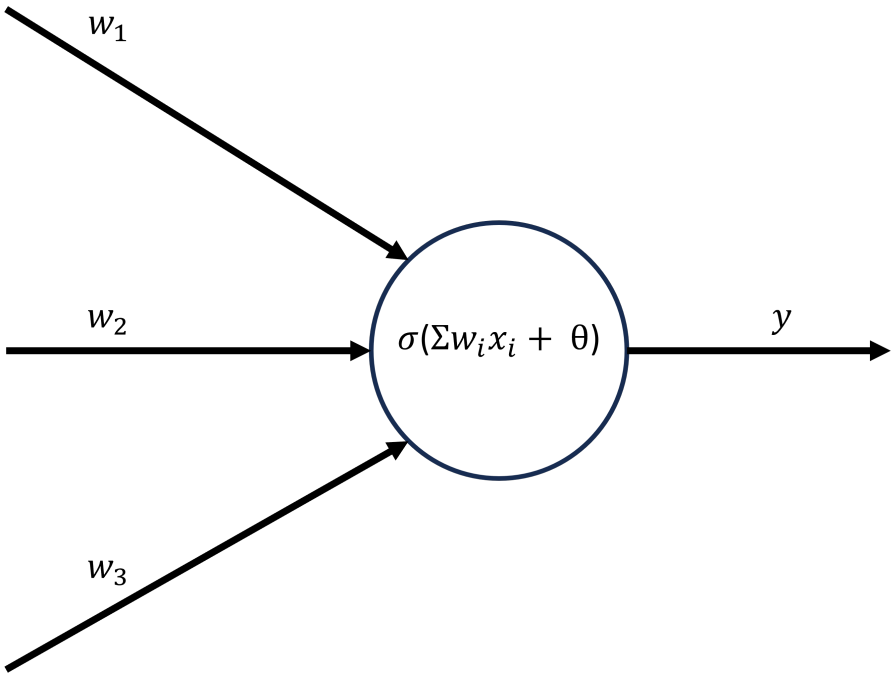


Figure 3.2: McCulloch-Pitts neuron with three input neurons.

In forward propagation, data moves through the network from the input layer to the output layer. For each neuron, the process is as follows:

$$s = \sum_{j=0}^{n} w_j x_j + \theta \tag{3.1}$$

which is a weighted sum, and the output is determined with an activation function

$$y = \sigma(s). \tag{3.2}$$

Backpropagation is the heart of neural network training, allowing the model to adjust its weights and thresholds to minimize the loss function [5]. The process involves two key steps:

- **Computing the Gradient**: Calculate the gradient of the loss function with respect to each weight and threshold in the network. This involves applying the chain rule of calculus to find how the loss function changes as the weights and threshold change.
- **Update the Weights and thresholds**: Adjust the weights and thresholds in the direction that reduces the loss. This is done using gradient descent or variants thereof.

The update rule for a weight is given by:

$$w = w - \eta \frac{\partial E}{\partial w} \tag{3.3}$$

$$\theta = \theta - \eta \frac{\partial E}{\partial \theta} \tag{3.4}$$

where E is the loss retrieved from the loss function, and η is the step size.

3.1.1 Transformers

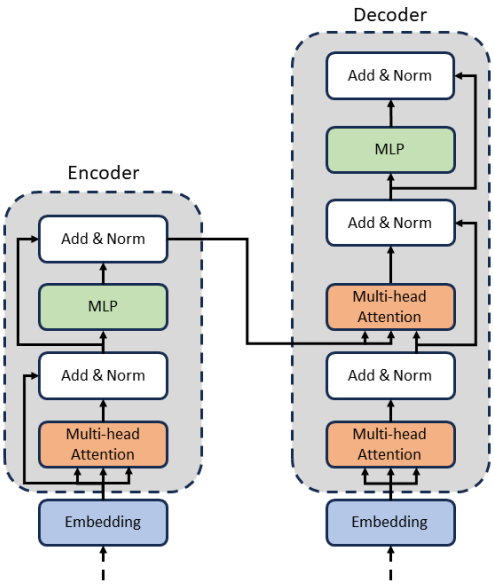


Figure 3.3: Simplified flowchart representation of the transformer architecture proposed in [15].

The Transformer architecture [15] introduced in 2017 is now widely used in Large Language Models. This architecture consists of an encoder, and decoder stack, see figure 3.3.

3.1.1.1 Self Attention Mechanisms

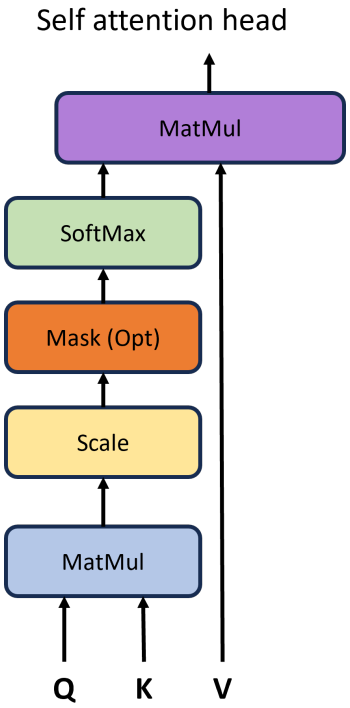


Figure 3.4: Self attention layer

The self-attention mechanism allows the model to weigh the importance of different words in a sentence in relation to each other. It enables the model to capture dependencies between words, regardless of their positional distance within the sentence. This is particularly useful in understanding the context and the semantic relationships within the text.

Essentially, self-attention computes a weighted sum of all words in the input sequence, where the weights are determined by the compatibility (similarity) between the words. For a given word, the self-attention mechanism calculates three vectors:

- **Query vector (Q)**: Represents the word in the context of looking for other words.
- **Key vector (K)**: Represents other words in the context of being looked up.
- **Value vector (V)**: Represents other words in the context of what they have to offer to the query word.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}V\right) \tag{3.5}$$

The attention weights are computed by taking the dot product of the query vector with all key vectors, followed by a softmax operation to normalize the weights. Then, the output is calculated by weighting the value vectors according to these attention weights [15].

3.1.1.2 Multi Head Attention

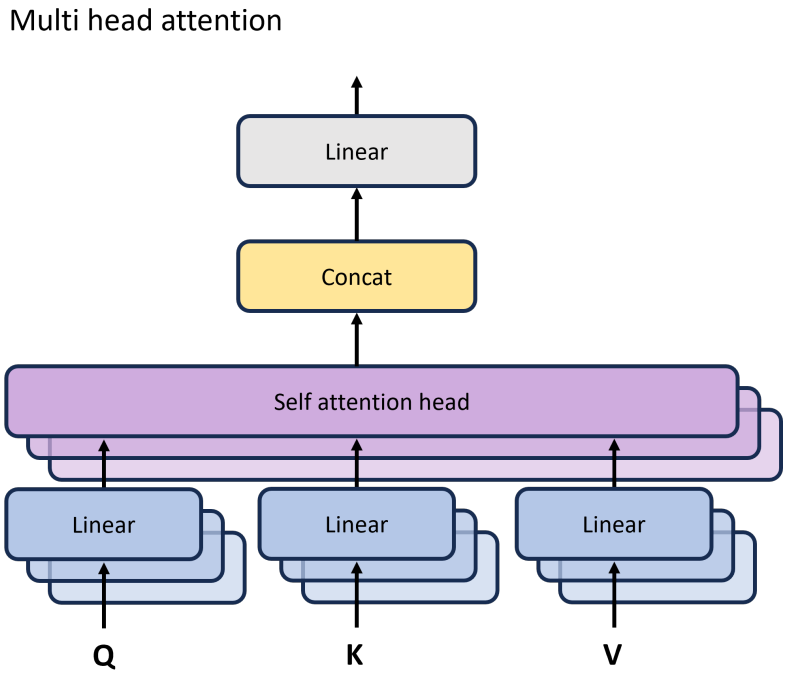


Figure 3.5: Multi head attention layer

While the self-attention mechanism can capture complex dependencies, it might focus on a single aspect of the relationship between words. Multi-head attention addresses this by splitting the query, key, and value vectors into multiple "heads", allowing the model to attend to different aspects of the information contained in the input sequence simultaneously.

Each head in the multi-head attention mechanism performs self-attention independently, with its own set of weights, thereby focusing on different parts of the input. The outputs of all heads are then concatenated and linearly transformed to produce the final output. This design enables the model to capture a wider representation of the input data by considering different perspectives [15].

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, ..., \text{head}_h)W^O \tag{3.6}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3.7}$$

using equation 3.5.

3.1.1.3 Context Window

The context window of an LLM is the number of tokens a language model can take as an input when generating a sequence. GPT-3 for instance, has a context window of 2,000 tokens. For the newer model GPT-4 this increased to 32,000 tokens [30].

Large context windows can improve the ability to make use of in-context learning. Which essentially simplifies the process of asking an LLM to perform a task, whilst also prompting it with key information on how to solve it.

3.2 Key concepts in Language Models

3.2.1 Tokenization

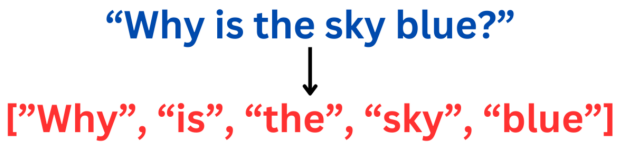


Figure 3.6: Tokenization example of the sentence "Why is the sky blue?".

Tokenization in NLP refers to the process of breaking down text into smaller units, or tokens, which can range from words to sentences. This crucial step transforms raw text into a structured form that computational algorithms can understand and analyze. Although it might seem simple, often starting with splitting text at spaces and punctuation, the complexity of text can make the process increasingly difficult. Tokenization lays the foundation for further text analysis and processing tasks, enabling a variety of applications in machine learning and artificial intelligence by providing a systematic way to interpret language data.

3.2.2 Embeddings

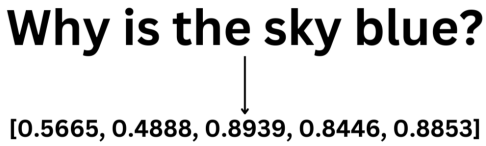


Figure 3.7: Word embedding example for the sentence "Why is the sky blue?"

Word embeddings are a group of techniques in NLP, where words or phrases from the vocabulary are mapped to vectors of real numbers in a low-dimensional space relative to the vocabulary size. This computational method enables machines to understand words in terms of their semantic meanings based on their context—words with similar meanings are positioned closer in the vector space, facilitating a deeper understanding of text. Originating from neural network research, word embeddings are foundational for a wide range of NLP applications, including machine translation, sentiment analysis, and information retrieval, by capturing syntactic and semantic word relationships in a dense, continuous vector space. Tools like Word2Vec

rong2014word2vec, GloVe [13], and FastText [14] are popular implementations that have revolutionized the way computers interpret human language, making it possible for machines to process text in a more nuanced and human-like manner.

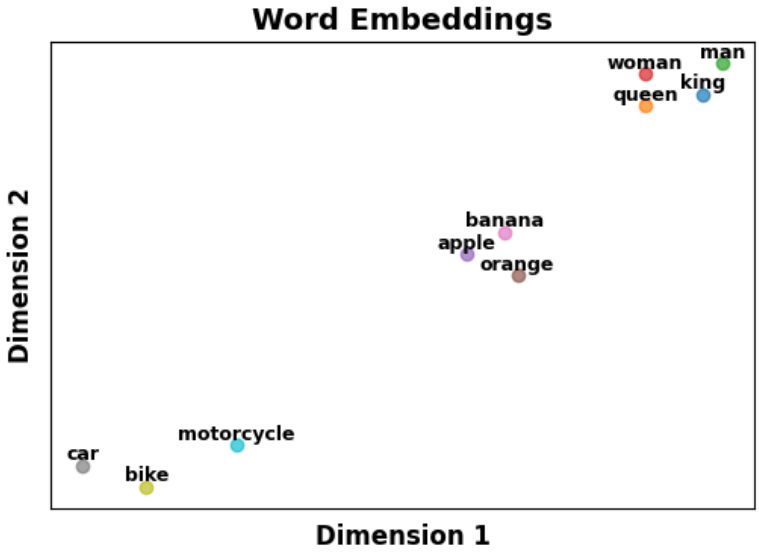


Figure 3.8: Illustration of word embeddings in a vector space. Words like king, man, queen and woman are closely related in context, therefore they lie close to one another in this 2D representation.

3.2.3 Named-entity Recognition (NER)

Named Entity Recognition (NER) is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

The core of NER involves two steps: detection and classification. First, it identifies that a named entity is present in the text. Second, it classifies the entity into one of the predefined categories.

- **Detection**: This step involves identifying potential named entities in the text. Early approaches used hand-crafted rules or gazetteers (lists of entities) to spot names. Modern systems, especially those based on machine learning, typically use tokenization, and part-of-speech tagging as preliminary steps to identify potential named entities.
- **Classification**: Once potential entities are detected, the system must classify them into specific categories. This step often relies on context, using the surrounding words and their part-of-speech tags to determine the category of each entity.

3.3 Pre-trained Models

3.3.1 Llama 2

Llama 2 (Large Language Model Meta AI) is a series of advanced large language models developed by the team at GenAI, Meta, supplying various models from 7 billion to 70 billion parameters. These models, including the fine-tuned variants specifically optimized for chat applications (Llama 2-Chat), were publicly released by Meta in July 2023 [27].

To create LLama 2, the successor of Llama 1, the team at GenAI followed an improved approach compared to the development of Llama 1. The approach used an optimized auto-regressive transformer with several changes to improve performance. The approach included more robust data cleaning, updated data mixes and training on 40% more total tokens together with doubled context length compared to the training of Llama 1. The data used for the pre-training was a new mix of data from publicly available sources excluding data from Meta's products and services. Specific model architecture details is not provided in the report.

The fine-tuning of the Llama 2 models includes **supervised fine-tuning**, **reinforcement learning with human feedback** and **ghost attention**.

Llama 2 models generally outperforms open-source models across a range of benchmarks [27]. This includes improvements in code, commonsense reasoning, world knowledge, reading comprehension, and math benchmarks. Specifically, Llama 2's performance on popular aggregated benchmarks like MMLU and BBH is highlighted, showing significant improvements over Llama 1 and other open-source competitors.

Llama 2, especially the Llama 2 70B model, shows competitive performance compared to closed-source models like GPT-3.5, PaLM, and even GPT-4 in certain benchmarks [27]. It's worth noting that while Llama 2 demonstrates close performance on MMLU and GSM8K benchmarks compared to GPT-3.5, it still exhibits a gap in coding benchmarks against models like GPT-4 and PaLM-2-L [27].

3.3.2 Mistral 7B

In October 2023 Mistral AI released their large language model called Mistral 7B. Mistral 7B is introduced as a 7-billion parameter designed to deliver high performance alongside with efficiency.

Grouped-Query Attention (GQA) and Sliding Window Attention (SWA) are key to Mistral 7B's efficiency and ability to handle long sequences without increased inference costs.

Mistral 7B is built on the transformer architecture with specific configurations like a dimension of 4096, 32 layers, 32 heads (with 8 key/value heads), and a window

size of 4096 tokens, allowing for a theoretical attention span of approximately 131K tokens after all layers.

The report does not provide explicit pre-training data sources or techniques but emphasizes architectural innovations that enables Mistral 7B to outperform other larger models. Two of the innovations are **rolling buffer cache** that reduces cache memory usage by maintaining a fixed cache size and **Pre-fill and chunking** which enhances processing efficiency by pre-filling the cache with known prompt information and handling long prompts in chunks to optimize memory usage.

The performance of Mistral 7B demonstrates superiority compared to 7B and 13B models in chatbot benchmarks. It surpasses Llama 2 (13B) and approaches the performance of Code-Llama 7B in code-related tasks without compromising on other benchmarks. [24]

3.3.3 Llama3

In April 2024, Meta released LLaMA-3, an advanced large language model with 8 billion parameters, designed to deliver high performance and efficiency [28]. Built on the transformer architecture of its predecessors, LLaMA-3 incorporates newer concepts that enhance its capabilities.

LLaMA-3 leverages advanced techniques such as Grouped-Query Attention (GQA) to efficiently manage long sequences without escalating inference costs. GQA optimizes the query distribution process by grouping similar queries together, improving processing speed and memory usage.

Structurally, LLaMA-3 holds a dimension of 8192 in context window length.
The pre-training phase for LLaMA-3 is marked by its use of diverse datasets, consisting a wide range of text types. This broad dataset provides a broad knowledge base and enhances the model's versatility. The specific data sources and pre-training techniques however, remain undisclosed.

Evaluations of LLaMA-3 shows its superior performance over previous models, including LLaMA-2 (7B) and comparable models like Mistral 7B [28]. LLaMA-3 outperforms these in a variety of benchmarks, including natural language understanding, code generation, and long-form text generation.

3.4 Fine Tuning

LLM fine-tuning is the process of taking pre-trained models and further training them on smaller, specific datasets to refine their capabilities and improve performance in a particular task or domain. Fine-tuning is about turning general-purpose models and turning them into specialized models. It bridges the gap between generic pre-trained models and the unique requirements of specific applications, ensuring that the language model aligns closely with human expectations.

In traditional fine-tuning, we tweak all the parameters of a pre-trained model to better suit a particular task. While effective, this approach can be resource-intensive. Parameter-efficient fine-tuning (PEFT), in contrast, involves updating only a small subset of the model's parameters.
There are many different ways of fine tuning a model. The fine tuning methods that were used in this report was methods with PEFT.

3.4.1 Parameter-efficient Fine Tuning (PEFT)

It was previously mentioned that PEFT [23] [22] involves only updating a small subset of the model's parameters. This subset can be a fraction of the model's total parameters, significantly reducing the computational cost and memory footprint.
While parameter-efficient fine-tuning offers numerous advantages, it's not without challenges. The selection of which parameters to train is crucial and can significantly affect performance. Furthermore, the techniques may not always reach the performance levels of full model fine-tuning, especially for very task-specific adaptations.

3.4.2 Low Rank Adapters (LoRA)

Low Rank Adapters [21] is a technique designed to fine-tune large pre-trained models in a PEFT manner. The idea behind LoRA is to add trainable low-rank matrices to the existing layers of a model without altering the original pre-trained weights. This approach enables significant modifications to the model's behavior while only adjusting a small subset of parameters.
LoRA is particularly popular in scenarios where deploying large-scale models is constrained by computational resources, as it allows for the customization of models without the extensive costs associated with full model training. The technique is a part of a broader trend towards more efficient machine learning methods, particularly in the context of deploying LLMs.

3.4.3 Quantization

QLoRA [29] stands for "Quantized Low Rank Adapters" and is a method for fine-tuning LLMs in a way that is efficient in terms of memory usage. This technique allows for fine-tuning models with billions of parameters on relatively modest hardware, like a single GPU, while maintaining performance close to that of full precision models [17].
Some important components of QLoRA is:

- **4-bit NormalFloat (NF4) Quantization** : This quantizes the model weights to 4-bits using a data type that is optimized for normally distributed data, which reduces memory usage significantly.
- **Double Quantization**: This further reduces memory demands by quantizing the quantization constants used in NF4.
- **Paged Optimizers**: These manage memory spikes during training, preventing out-of-memory errors that are common in GPU-intensive tasks.

3.5 Retrieval Augmented Generation (RAG)

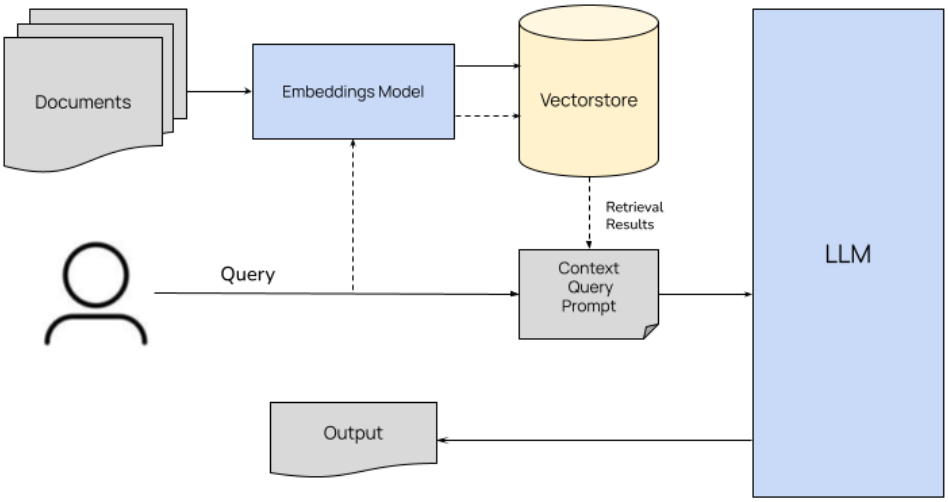


Figure 3.9: The RAG framework, as proposed in [19].

Retrieval-Augmented Generation (RAG) is a hybrid model that combines the generative power of modern language models with the information retrieval system strengths in the theoretical exploration of developments in natural language processing. This model represents an important step towards developing AI systems that can dynamically obtain information from large data sources to provide text outputs that are accurate and contextually improved.

By equipping pre-trained, parametric-memory generation models with a non parametric memory through a defined pipeline, RAG can be achieved. A framework that operates on a two-step process to function. First, it searches a large database for documents that are relevant to the input query and retrieves them. The information retrieved then provides the generative model with a contextual foundation, to base a factual answer on. This approach overcomes the fundamental limitations of standalone language models, especially when producing material that requires in-depth, specialized, or current information beyond what is provided by the model's training set. [19]

3.5.1 Retrieval

As mentioned the retriever component in the RAG framework plays a critical role in identifying and fetching relevant information from the data source, which is then used to inform the generation process of the model. Both the query and the documents are transformed and represented as vectors in a high-dimensional space. This is achieved with embedding models [10] (section 3.2.2).

The query q is transformed into a vector $\vec{q}$ using an embedding function E_q.

$$\vec{q} = E_q(q) \tag{3.8}$$

Documents $D = \{d_1, d_2, \ldots, d_n\}$ in the data source are transformed into vectors $\{\vec{d_1}, \vec{d_2}, \ldots, \vec{d_n}\}$ using the same embedding function E_q:

$$\vec{d_i} = E_q(d_i), \quad \forall d_i \in D \tag{3.9}$$

The foundation of the retrieval mechanism is to calculate the similarity between the query vector and each document vector. This similarity score is generally computed using cosine similarity or the dot product between the two.
The cosine similarity measures the cosine of the angle between the vectors.

$$S_{\cos}(q, d_i) = \frac{\vec{q} \cdot \vec{d_i}}{\|\vec{q}\| \|\vec{d_i}\|} \tag{3.10}$$

Based on the calculated similarity scores, the retriever selects the top k documents, which have the highest similarity to the query. These documents are then passed to the generation component of the RAG model as information.

3.5.2 Generation

The generation component in RAG frameworks is responsible for producing the final text output, utilizing both the original query and the context provided by the retrieved documents from the retriever component. This process is usually handled by a generative language model, based on architectures such as transformers or its variants like GPT or Bidirectional Encoder Representations from Transformers (BERT) for sequence-to-sequence tasks [20].

3.5.3 Chunking

Chunking, as used in RAG, is the process of dividing large datasets or text documents into smaller, easier-to-manage "chunks". This is an important step in the RAG framework, especially for the retriever component, since it affects how quickly and effectively important information can be retrieved from a large body of data. Chunking is useful when dividing a collection of documents into distinct sections or when dividing a single, large document into multiple parts.

For single documents, especially those that are lengthy, chunking involves dividing the text into smaller sections that can still be meaningfully understood and processed independently. This is important because large documents might contain multiple themes or topics, and chunking allows the model to focus on the parts most relevant to the query. Chunking, when used to a collection of documents, usually involves dividing the dataset into subsets based on certain criteria (e.g., topic, length, source). Because the retriever can focus on a more manageable and useful

portion of the full data set, this raises the likelihood of locating relevant information and makes the retrieval process more effective.

There are several approaches to chunking, each with its own advantages. The **fixed-length chunking** method divides text into chunks of a predetermined size. It's straightforward but can split sentences or paragraphs in ways that might lose context or meaning. With **semantic chunking**, NLP techniques are used to identify meaningful sections of text, such as paragraphs or sections based on headings. This method is more complex but preserves the semantic integrity of the text. Lastly, **hybrid chunking** is a combination of fixed-length and semantic methods, where text is primarily chunked based on semantic boundaries, but long sections are further divided into fixed-length pieces to ensure uniformity [26].

3.6 Evaluation

This section will present some key concepts used in the evaluation of different solutions.

3.6.1 Metrics

To compare different approaches it's important to have measurable values. Following, is a brief walk-through of the commonly used metrics: recall, precision and F_1 score.

3.6.1.1 Binary classes

When working with binary classification problems the predictions can be divided into 4 groups, namely:

- **True Positive (TP)** - Instances of class 'positive' that were correctly predicted as positive.
- **False Positive (FP)** - Instances of class 'negative' that were mistakenly predicted as positive.
- **True Negative (TN)** - Instances of class 'negative' that were correctly predicted as negative.
- **False Negative (FN)** - Instances of class 'positive' that were mistakenly predicted as negative.

3.6.1.2 Recall

The recall score, also known as sensitivity or true positive rate, is defined as:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.11}$$

where the score should ideally be 1 when testing a good classifier. In words, recall can be seen as the ratio of correctly classified positives out of all positive samples present.

3.6.1.3 Precision

Precision, also known as positive predictive value, is the rate of correctly predicted positives with respect to all the predicted positives.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.12}$$

3.6.1.4 F1 score

The F_1 score is a metric combining both the recall and precision.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.13}$$

3.6.2 Self-Evaluation

Self-Evaluation refers to the model's ability to assess the quality and relevance of its own generated outputs or responses. The framework Ragas is designed to help evaluate and quantify the performance of RAG pipelines. In this section we will further explain the metrics that were used for self-evaluation.

3.6.2.1 Answer Semantic Similarity

Answer Semantic Similarity [35] involves the evaluation of the resemblance between the generated answer and the ground truth. The value of the result falls within the range of 0 to 1, where a higher score indicates a better alignment between the generated answer and the truth.

The ragas framework performs this calculation in three steps:

1. Vectorizing the ground truth using the specified embedding model.
2. Vectorizing the generated answer using the same embedding model.
3. Computing the cosine similarity between the two vectors.

3.6.2.2 Answer Correctness

Answer correctness [31] involves calculating the accuracy of the generated answer compared to the ground truth. This evaluation includes two critical aspects: The semantic similarity as explained above, together with the factual similarity.
The factual similarity is calculated by calculating the F1 Score. (eq. 3.13)
The semantic similarity is calculated as described above. Once the semantic similarity is calculated, a weighted average of the semantic similarity and factual similarity is calculated to arrive at the final score. The default weightage is 0.75 and 0.25.
The final score is calculated with:

$$\text{Answer Correctness} = 0.75 \cdot \text{F1 Score} + 0.25 \cdot \text{Semantic Similarity} \tag{3.14}$$

3.6.2.3 Answer Relevancy

Answer Relevancy [32] focuses on assessing how relevant the generated answer is to the given prompt. A lower score is given to answers that are incomplete or contain redundant information and higher scores indicate higher relevancy.

$$\text{Answer Relevancy} = \frac{1}{N}\sum_{i=1}^{N}\frac{E_{g_i}\cdot E_o}{\|E_{g_i}\|\|E_o\|} \tag{3.15}$$

Where E_{g_i} is the embedding of the generated question i, E_o is the embedding of the original question and N is the number of generated questions, generally three.

The Ragas framework uses two main steps to calculate the relevance of the answer to the given question. The first step is to reverse-engineer three questions from the generated answer using a large language model. The second and last step is to calculate the mean cosine similarity between the generated questions and the actual question. If the generated answer accurately addresses the initial question, the LLM should be able to generate similar question from the answer only.

3.6.2.4 Context Precision

The metric Context Precision [33] evaluates whether all of the ground truth relevant items present in the context are ranked higher or not. Optimally all the relevant chunks must appear at the top when the retriever fetches the chunks based on a question. The metric is calculated in two steps.

$$\text{Precision@k} = \frac{\text{TP@k}}{(\text{TP@k} + \text{FP@k})} \tag{3.16}$$

$$\text{Context Precision@K} = \frac{\sum_{k=1}^{K}(\text{Precision@k} \times v_k)}{\text{Total number of relevant items in the top } K \text{ results}} \tag{3.17}$$

where K is the total number of chunks, v_k is the relevance indicator at rank k ranging from 0 to 1.

3.6.2.5 Faithfulness

The metric Faithfulness [34] measures the factual consistency between the generated answer against the retrieved context. The generated answer is regarded faithful if the claims in the answer can be found in the given context.

$$\text{Faithfulness} = \frac{|\text{Claims in the generated answer that can be inferred from given context}|}{|\text{Total number of claims in the generated answer}|} \tag{3.18}$$

4

Methods

The methods chapter covers how the theories discussed in this paper was leveraged in practice and adapted to the benefit of the project. We explain preprocessing for contextual amplification, deviations and customization from "standard" retrieval-augmented generation, fine-tuning, implementations, and finally how we evaluated all the different implementations and the methodology behind that.

4.1 Data Ingestion

The data ingestion process was managed by a pipeline specifically developed for this thesis. This pipeline was designed under the assumption that Ericsson has pre-existing log parsers in place, therefore, parsing a complete log file was not necessary. Initially, the log file comprised tens of thousands of rows. However, after processing it with Ericsson's established parsers, the data was reduced to more manageable and relevant segments of the log file.

The first step after the initial data parsing was to apply basic regex parsing to the parsed log file. From the regex parsing, the log data was saved into three custom data structures denoted as **Tables**, **Time Series**, and **Texts**.

4.2 Preprocessing for Contextual Amplification

Since the pre-trained models available are not commonly trained with log-data, different measures had to be taken in order to include as much context as possible for a general LLM and clarify the information that was being fed. This section will discuss more about these methods, which will later act as modules, and then combined into different solutions in section 4.5.

4.2.1 Document Elements

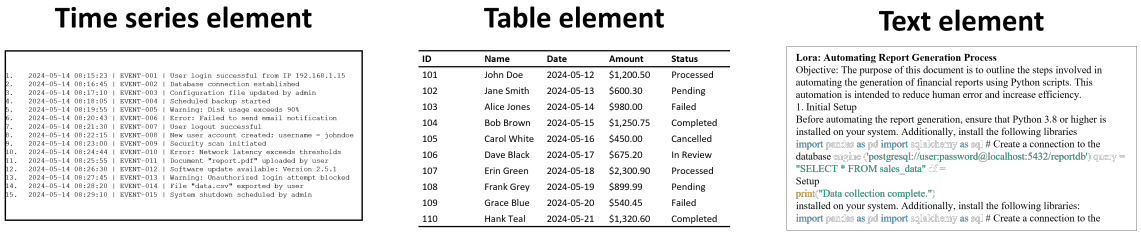


Figure 4.1: Log elements defined

Documents consist of different classes of elements. These were selected to fit the input data of the project. The elements identified was

- Table elements - the elements that were following a strict formatting such as having separators between column and row, and having a header.
- Time series elements - The elements which follows a more system log-like format, including a time stamp and some message.
- Text elements - Elements that did not fall under the previous two.

Having identified these different element classes, the data could then be parsed and converted to a structured format.

```
=================================================================================================================================
FRU       ;LNH      ;BOARD      ;ST ;FAULT ;OPER ;MAINT ;STAT ;PRODUCTNUMBER    ;REV     ;SERIAL        ;DATE     ; TEMP ; UPT
=================================================================================================================================
1         ;000100   ;DUS5201    ; 1 ;  OFF ;  ON ;  OFF ; OFF ;KDUXXXXXXX1    ;R5A     ;CD3XXXXXXX    ;20200508 ;      ;6.40  (Bb5216)
RRU-13    ;BXP_0    ;RRU4426B66 ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX3     ;R1E     ;D82XXXXXXX    ;20200623 ; 52.5 ;0.32  (Radio4426B66)
RRU-14    ;BXP_1    ;RRU4426B66 ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX3     ;R1E     ;D82XXXXXXX    ;20200623 ; 49.5 ;4.24  (Radio4426B66)
RRU-15    ;BXP_2    ;RRU4426B66 ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX3     ;R1E     ;D82XXXXXXX    ;20200623 ; 54.0 ;4.24  (Radio4426B66)
XMU03-2-1 ;BXP_3    ;XMU0301    ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KDUXXXXXXX1     ;R2A     ;CD3XXXXXXX    ;20170414 ; 35.0 ;6.40  (BbR503)
RRU-16    ;BXP_4_8  ;RRU4478B14 ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX3     ;R1B     ;CF8XXXXXXX    ;20200427 ; 45.3 ;0.32  (Radio4478B14)
RRU-17    ;BXP_4_9  ;RRU4478B14 ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX3     ;R1B     ;CF8XXXXXXX    ;20200425 ; 40.7 ;3.38  (Radio4478B14)
RRU-18    ;BXP_4_10 ;RRU4478B14 ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX3     ;R1B     ;CF8XXXXXXX    ;20200504 ; 50.3 ;0.32  (Radio4478B14)
RRU-7     ;BXP_5_3  ;RRUS32B30  ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX1     ;R1D     ;D16XXXXXXX    ;20180607 ; 54.3 ;5.23
RRU-8     ;BXP_5_4  ;RRUS32B30  ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX1     ;R1D     ;D16XXXXXXX    ;20180605 ; 40.0 ;4.41
RRU-9     ;BXP_5_5  ;RRUS32B30  ; 1 ;  OFF ;  ON ;  OFF ; N/A ;KRCXXXXXXX1     ;R1D     ;D16XXXXXXX    ;20180608 ; 65.5 ;5.18
SUP-2     ;Z??_01   ;SUP6601    ; 1 ;  OFF ;  ON ;  N/A ; N/A ;1/BFLXXXXXXX4   ;R2D     ;BW9XXXXXXX    ;20180701 ;      ;
---------------------------------------------------------------------------------------------------------------------------------
```

Figure 4.2: Unstructured table in raw text

	FRU	LNH	BOARD	ST	FAULT	OPER	MAINT	STAT	PRODUCTNUMBER	REV	SERIAL	DATE	TEMP	UPT
0	1	000100	DUS5201	1	OFF	ON	OFF	OFF	KDUXXXXXXX1	R5A	CD3XXXXXXX	20200508	Missing	6.40 (Bb5216)
1	RRU-13	BXP_0	RRU4426B66	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1E	D82XXXXXXX	20200623	52.5	0.32 (Radio4426B66)
2	RRU-14	BXP_1	RRU4426B66	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1E	D82XXXXXXX	20200623	49.5	4.24 (Radio4426B66)
3	RRU-15	BXP_2	RRU4426B66	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1E	D82XXXXXXX	20200623	54.0	4.24 (Radio4426B66)
4	XMU03-2-1	BXP_3	XMU0301	1	OFF	ON	OFF	N/A	KDUXXXXXXX1	R2A	CD3XXXXXXX	20170414	35.0	6.40 (BbR503)
5	RRU-16	BXP_4_8	RRU4478B14	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1B	CF8XXXXXXX	20200427	45.3	0.32 (Radio4478B14)
6	RRU-17	BXP_4_9	RRU4478B14	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1B	CF8XXXXXXX	20200425	40.7	3.38 (Radio4478B14)
7	RRU-18	BXP_4_10	RRU4478B14	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1B	CF8XXXXXXX	20200504	50.3	0.32 (Radio4478B14)
8	RRU-7	BXP_5_3	RRUS32B30	1	OFF	ON	OFF	N/A	KRCXXXXXXX1	R1D	D16XXXXXXX	20180607	54.3	5.23
9	RRU-8	BXP_5_4	RRUS32B30	1	OFF	ON	OFF	N/A	KRCXXXXXXX1	R1D	D16XXXXXXX	20180605	40.0	4.41
10	RRU-9	BXP_5_5	RRUS32B30	1	OFF	ON	OFF	N/A	KRCXXXXXXX1	R1D	D16XXXXXXX	20180608	65.5	5.18
11	SUP-2	Z??_01	SUP6601	1	OFF	ON	N/A	N/A	1/BFLXXXXXXX4	R2D	BW9XXXXXXX	20180701	Missing	Missing

Figure 4.3: Structured table after parsing, before extending acronyms

4.2.2 Domain Specific Acronyms

The Ericsson system logs holds a significant amount of acronyms. Many of these are very field specific or complex, which makes it difficult for LLMs not trained on this exact type of data to interpret or find context. In order to bring more context into the logs a lot of these acronyms were identified using *en_core_web_sm* which is an English language multi-task CNN trained on OntoNotes.

en_core_web_sm utilizes NER to identify entities, and in that way find what words are acronyms. Once having the acronyms identified, a dictionary for mapping each acronym to the corresponding term was formed. Doing this enabled the substitution for the first occurrence of an acronym to it's term, if it exists in the dictionary.

	FIELD REPLACEABLE UNIT (FRU)	LINK HANDLER (LNH)	BOARD	STATE (ST)	FAULT	OPER	MAINTENENCE MODE(MAINT)	STATUS (STAT)	PRODUCTNUMBER	REVISION (REV)	SERIAL	DATE	TEMPERATURE (TEMP)	UPT
0	1	000100	DUS5201	1	OFF	ON	OFF	OFF	KDUXXXXXXX1	R5A	CD3XXXXXXX	20200508	Missing	6.40 (Bb5216)
1	RRU-13	BXP_0	RRU4426B66	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1E	D82XXXXXXX	20200623	52.5	0.32 (Radio4426B66)
2	RRU-14	BXP_1	RRU4426B66	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1E	D82XXXXXXX	20200623	49.5	4.24 (Radio4426B66)
3	RRU-15	BXP_2	RRU4426B66	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1E	D82XXXXXXX	20200623	54.0	4.24 (Radio4426B66)
4	XMU03-2-1	BXP_3	XMU0301	1	OFF	ON	OFF	N/A	KDUXXXXXXX1	R2A	CD3XXXXXXX	20170414	35.0	6.40 (BbR503)
5	RRU-16	BXP_4_8	RRU4478B14	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1B	CF8XXXXXXX	20200427	45.3	0.32 (Radio4478B14)
6	RRU-17	BXP_4_9	RRU4478B14	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1B	CF8XXXXXXX	20200425	40.7	3.38 (Radio4478B14)
7	RRU-18	BXP_4_10	RRU4478B14	1	OFF	ON	OFF	N/A	KRCXXXXXXX3	R1B	CF8XXXXXXX	20200504	50.3	0.32 (Radio4478B14)
8	RRU-7	BXP_5_3	RRUS32B30	1	OFF	ON	OFF	N/A	KRCXXXXXXX1	R1D	D16XXXXXXX	20180607	54.3	5.23
9	RRU-8	BXP_5_4	RRUS32B30	1	OFF	ON	OFF	N/A	KRCXXXXXXX1	R1D	D16XXXXXXX	20180605	40.0	4.41
10	RRU-9	BXP_5_5	RRUS32B30	1	OFF	ON	OFF	N/A	KRCXXXXXXX1	R1D	D16XXXXXXX	20180608	65.5	5.18
11	SUP-2	Z??_01	SUP6601	1	OFF	ON	N/A	N/A	1/BFLXXXXXXX4	R2D	BW9XXXXXXX	20180701	Missing	Missing

Figure 4.4: Structured table after extension of acronyms in table header

4.2.3 Splitting Into Sub-tables

Problems arose because of the limited context window of the models, which restricted the number of tokens that could be processed in a single input. To address this, we split the document elements (figure 4.4) into smaller tables by adding an index, such as the unit's unique name or link id. Each new table then contained only an index and one type of data. This approach allowed for better handling of the context limitations.

This approach also aimed to make it easier to fetch the correct chunks, as the sentiment and context were aggressively narrowed down in the smaller tables, making summarization simpler.

4.2.4 Camel Case

The logs contained a variety of variables and elements of "programming language," resulting in the frequent occurrence of camel case within the text. To enhance clarity for language models or embedding models, these instances of camel case were, in some cases, separated into individual words.

4.2.5 Model Specific Preprocessing

The table elements and time series from the logs were in some cases converted to HTML format. This was helpful for generating more accurate results when working with table-like formats. This has been shown to increase pre-trained LLMs ability

to understand structured data [36].

Missing values were initially parsed as white spaces, but was later changed to *Missing* to indicate that there was no expected values there. This due to many models using the upcoming value in a table when coming across a missing value.

4.2.6 Including Element Metadata

To include further information about the extracted elements as a starting point for the chunk summary, metadata was formed, that would include the name of the element if it was matched to a known format, the columns or headers, a brief description of its contents and an example of a hypothetical question.

Below is an example of the metadata strings passed when creating summaries with metadata inclusion.

```
    {
    "description": "A table that contains information about all the
    units serial numbers and product numbers (FRU's) at the site.
    Also available is their reference name in the log file 'LNH'. LNH
    is the name given to the unit by the custommer.",
    "hypothetical_question": "Give me all the serial number, product
    numbers for corresponding FRU and LNH.",
    "log_file_row": "Line 0",
    "parent_table": "Inventory table"
}
```

4.2.7 Prompt Formatting

Different templates were employed for generating chunk summaries. The 'system prompt' established the contextual parameters and tone, which directed the functionality of the language model used in this study. The 'user prompt' clearly specified the task required from the language model. These components collectively ensured a structured and effective interaction between the user and the system.

```
system_prompt =
You will be assisting in summarizing tables from a radio log file,
you only output information and don't care about having
a conversation.

user_prompt = Give a summary of what this chunk contains, at least
    express everything present in the metadata provided and then move
    on from there. Additionally come up with two hypothetical questions
     in addition to the one already specified in the metdata. Just
    answer with what I am asking for.

Table: {table}
Metadata: {metadata}
```

4.3 RAG - Deviations and Adaptions

This section will focus on how modifications to the standard RAG model (section 3.5), was implemented to better fit the data and project scope.

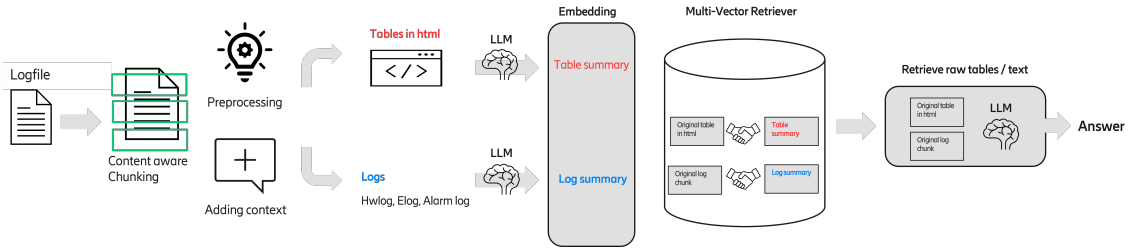


Figure 4.5: Illustration of the proposed RAG pipeline

4.3.1 Content Aware Chunking

To understand and preserve the semantic and contextual structure of the log file during the chunking process, some implementations included content-aware chunking.

Each individual table element and log element became a chunk to include the contextual and semantic structure of the log file.

4.3.2 Semantics summary and hypothetical questions

Each segmented chunk was summarized using an LLM. This was done to capture the semantic content of each segment, and in turn enhancing the ability for retrieving the most relevant chunks with a query.

Additionally, a number of hypothetical questions were produced as a part of the summary, based on the information contained in the chunk. This method was crucial to the framework, due to the large amount of numbered data and lack of sentiment in the data itself.

4.3.3 Indexing and Retrieval

The database system ChromaDB was used as a vector database. Different types of retrievers were used in this project. For the baseline approaches, a basic vectorstore retriever utilizing similarity search was was used. This was the most fundamental method within the ChromaDB system with focus on simplicity. Each chunk was embedded into a vector and then stored in the database. After a query was made and embedded, the system searched for the most similar chunk(s) based on their vector representations.

The second retriever used was the Multi-Vector Retriever which was a more advanced method within the ChromaDB system. It is designed to handle more complex

retrieval tasks and combines the capabilities of a Vectorstore with a Document Store. The combination allowed storing multiple vectors for each chunk. Each embedded summary was indexed together with the original content of that chunk, as was before summarization. Although a match was made between queries and summary embeddings, the actual segment that was being fed to the model, was the original one.

4.3.4 Generation / Retrieval With Agent

```
> Entering new AgentExecutor chain...
 Thought: I need to count the number of rows where the value in the "Severity" column is equal to "Cleared".
Action: python_repl_ast
Action Input: print(len(df[df["Severity"] == "Cleared"]))16
 I now know the final answer.
Final Answer: There have been 16 'Cleared' alarms.

> Finished chain.

{'input': "How many 'Cleared' alarms has there been?",
 'output': "There have been 16 'Cleared' alarms."}
```

Figure 4.6: Conversation between pandas-agents after asking the question *'How many cleared alarms have there been?'*

As a counter approach to 'standard RAG', an effort was made to use an agent based retriever - generator. The concept behind agents is to use a language model to select a sequence of actions. The sequence of actions is predetermined and hardcoded, then the 'prompter' utilizes a language model as a reasoning engine to decide which actions to take and in what order. This chain of reasoning is demonstrated in figure 4.6.

For a open-source model to be able to use the pandas agent, the model had to be fine-tuned on "function_calls" and return "python" as an argument. It appeared that the Llama3 and Llama2 models utilized in this project was not fined tuned on this and only returned a "text" response. In contrast, the Mistral 7B model (section 3.3.2) was and therefore the only model used in generation and retrieval with agents.

4.4 Fine Tuning

This section revolves the fine-tuning process of the LLaMA-3 (section 3.3.3) model with 8 billion parameters, to enhance its ability in comprehending the domain-specific language used in the Ericsson logs. The model was adapted through Parameter-efficient fine tuning (PEFT), combined with quantization to a 4-bit representation, aiming to optimize both performance and efficiency in the summarization part of the RAG pipeline.

4.4.1 Quantization to 4-bit

To further enhance the model's efficiency, particularly for deployment in resource-constrained environments at Ericsson, the model's parameters were quantized to

4-bit integers. Transitioning from 32-bit floating points to 4-bit integers significantly decreased the model's memory usage. Following quantization, additional fine-tuning ensured the model adapted to the lower precision without substantial loss in performance.

4.4.2 Fine Tuning Dataset

Table 4.1: Three examples showcasing the question-answer pairs that were used for the fine tuning step.

	Question	**Answer**
1	What does the STAT indicator represent on a product's faceplate?	STAT indicates the operational state of a yellow LED on a product's faceplate; OFF means no fault, ON indicates a fault, and N/A for not applicable.
2	What does the MO attribute ID represent in a RiLink Table?	The ID in a RiLink Table is an arbitrary number used to identify the CPRI link.
3	What does the BOARD1-BOARD2 attribute indicate in RiLink Table #3?	BOARD1-BOARD2 specifies the BOARD type on each side of the link from the hardware inventory.

The model was fine-tuned using a dataset created within the project. A dictionary-like document was acquired from an SME. This was a dictionary which consisted of technical terms that were commonly encountered in the log files. An LLM was utilized to assist in easily generate a series of questions and answers, formed from this dictionary.

4.4.3 Training

Due to the fine tuning becoming available very late in the project, it was very experimental and focused on quickly getting a result from training. The idea was to investigate if the results could at least, improve somewhat. The training lacked a validation set for training, this was a result from the insufficient time remaining. Instead the model adapters were saved as checkpoints, with the logic that if the end model was being overfitted, there was a possibility to go back to a previous iteration.

The training was carried out with following training parameters:

- 5 Warmup steps
- 2 Train batches per device
- Gradient checkpointing
- 4 Gradient accumulation steps
- 1000 Update steps

- $2.5 \cdot e^{-5}$ Learning rate
- Paged Adamw 8bit - Optimizer
- 10 Save steps

and LoRA parameters:

- $r = 8$
- $\alpha_{LoRA} = 16$
- LoRA dropout = 0.05
- No thresholds

Training was performed on a Ray-cluster with 4 workers, each holding an Nvidia A2 TPU. The task was completed after 8 hours. However the finally selected checkpoint out of the 100 existing ($\frac{1000 \text{ it}}{10 \text{ Save steps}} = 100$) was checkpoint 20, due to bad tendencies from overfitting.

Below is the prompt template that was used during fine tuning to incorporate all angles of the knowledge.

```
"""Below is an instruction that describes a question, paired with
an input that provides further context. Write a response that
appropriately completes the request.

Question:
{question}

Context:
{context}

Response:
{response}"""
```

4.5 Implementations

This section describes various implementation strategies. The section will refer to, and assemble previously explained concepts from the Theory and Method chapter and use these as modules. Together they form unique solutions and are given unique names and references.

4.5.1 Baseline

This implementation is resembling the most straight forward and established RAG approach. The idea was to use this approach as a benchmark approach and have it as reference for other methods. This implementation was used with all three models present in the report (Llama2, Llama3, Mistral).

Pre-processing:

1. **Chunking strategy:** Fixed chunking
    - Chunk-size: 500 characters
    - Chunk overlap: 50 characters

Storage:

1. **Embedding model:** Nomic-embed-text
    - Embedding dimensions: 768
    - Context length: 8,192
2. **Database:** ChromaDB vector store

Retriever:

- **Retrieval method:** Vector store retriever
    - Documents retrieved: 4
    - Similarity search: Cosine similarity

Generator:

- **Large Language Model:**
    - Llama2 13B Q4
    - Llama3 8B Q4
    - Mistral 7B

4.5.2 Implementation 1

This implementation aimed to enhance the contextual awareness of the model. Log elements were extracted and split into chunks in pandas Dataframe format, see 4.2.1. Each chunk was summarized and hypothetical questions were generated, to enrich the retrieval component of the pipeline. A multi-vector retriever was employed to enable storing multiple vectors for each chunk.

Pre-processing:

1. **Chunking strategy:** Context aware chunking
    - Full size document elements
2. Converted to pandas dataframes
3. Extension of domain specific acronyms
4. Filling missing values

Storage:

1. **Embedding model:** Nomic-embed-text
    - Embedding dimensions: 768
    - Context length: 8,192
2. **Database:** ChromaDB vector store

Retriever:

- **Retrieval method:** Multi-Vector retriever
    - Documents retrieved: 1
    - Similarity search: Cosine similarity

Generator:

- **Large Language Model:**
    - Llama2 13B Q4
    - Llama3 8B Q4
    - Mistral 7B

4.5.3 Implementation 2

Similar to the previous implementation, this approach involved contextual chunking, with the addition of splitting document elements into sub-elements. Each defined column in the previous document element became an independent table, each with a predetermined index. This strategy aimed to reduce the risk of model hallucination due to overfull context window.

Pre-processing:

1. **Chunking strategy:** Context aware chunking
    - Sub-elements
2. Converted to pandas dataframes
3. Extension of domain specific acronyms
4. Filling missing values

Storage:

1. **Embedding model:** Nomic-embed-text
    - Embedding dimensions: 768
    - Context length: 8,192
2. **Database:** ChromaDB vector store

Retriever:

- **Retrieval method:** Multi-Vector retriever
    - Documents retrieved: 1
    - Similarity search: Cosine similarity

Generator:

- **Large Language Model:**
    - Llama2 13B Q4
    - Llama3 8B Q4
    - Mistral 7B

4.5.4 Implementation 3

This implementation adapted the strategies of Implementation 2, with the modification of converting elements into HTML format to potentially improve model accuracy. The idea for this implementation was to see if changing the format of the input to Markup language could improve the understanding of the model.

Pre-processing:

1. **Chunking strategy:** Context aware chunking
    - Full size document elements
2. Converted to html tables
3. Extension of domain specific acronyms
4. Filling missing values

Storage:

1. **Embedding model:** Nomic-embed-text
    - Embedding dimensions: 768
    - Context length: 8,192
2. **Database:** ChromaDB vector store

Retriever:

- **Retrieval method:** Multi-Vector retriever
  - Documents retrieved: 1
  - Similarity search: Cosine similarity

Generator:

- **Large Language Model:**
  - Llama2 13B Q4
  - Llama3 8B Q4
  - Mistral 7B

4.5.5 Implementation 4

Implementation 4 was an adaptation of implementation 3. This was done with the same idea of increasing model accuracy with Markup language. The sole distinction being the conversion of elements into sub-elements in an attempt to further enhance the performance of the model.

Pre-processing:

1. **Chunking strategy:** Context aware chunking
   - Sub-elements
2. Converted to html tables
3. Extension of domain specific acronyms
4. Filling missing values

Storage:

1. **Embedding model:** Nomic-embed-text
   - Embedding dimensions: 768
   - Context length: 8,192
2. **Database:** ChromaDB vector store

Retriever:

- **Retrieval method:** Multi-Vector retriever
  - Documents retrieved: 1
  - Similarity search: Cosine similarity

Generator:

- **Large Language Model:**
  - Llama2 13B Q4
  - Llama3 8B Q4
  - Mistral 7B

4.5.6 Implementation 5

Implementation 5 was an agent based solution. Meaning that the retrieval and generation slightly differ from other solutions. The difference being that the retriever, which previously fetched strings, in this implementation fetched pandas dataframes. The agent ran python code in order to find the appropriate data from said dataframes, and made deductions from it.

Pre-processing:

1. **Chunking strategy:** Context aware chunking

    - Full size document elements
2. Converted to pandas dataframes
3. Extension of domain specific acronyms
4. Filling missing values
5. Adding metadata

Storage:

- **Summarizing chunks:**
  - Summary model: Llama3 8B Q4
  - Temperature 0
- **Embedding model:** Nomic-embed-text
  - Embedding dimensions: 768
  - Context length: 8,192

Retriever:

- **Retrieval method:** Multi-Vector retriever
  - Documents retrieved: 1
  - Similarity search: Cosine similarity
  - Agent chain (as explained in section 4.3.4)

Generator:

- **Agent chain:**
  - Agent model: Mistral 7B Q4

4.5.7 Implementation 6

This implementation is also agent based and close to identical with section 4.5.6. The difference was that the fine tuned version of Llama3 (Llogama3) was used for chunk summaries. The reason to this was to boost context in the chunks when summarizing. And to be able to encode more semantic content in the vectors.

Pre-processing:

1. **Chunking strategy:** Context aware chunking
    - Full size document elements
2. Converted to pandas dataframes
3. Extension of domain specific acronyms
4. Filling missing values
5. Adding metadata

Storage:

1. **Summarizing chunks:** LlogaMa3 was used to create chunk summaries (Fine tuned version of Llama3).
2. **Embedding model:** Nomic-embed-text
    - Embedding dimensions: 768
    - Context length: 8,192

Retriever:

- **Retrieval method:** Multi-Vector retriever
  - Documents retrieved: 1
  - Similarity search: Cosine similarity
  - Agent chain (as explained in 4.3.4)

Generator:

- **Agent chain:**
  - Agent model: Mistral 7B Q4

4.6 Evaluation Methods

The methodology for evaluating the approaches, and the combinations of these are outlined below:

1. **RAG baseline approach:**
   (a) Evaluate the baseline approach with the three models on the generated evaluation set.
   (b) Compare the results for different models to determine which model yields the best results.
2. **Implementations**
   (a) Evaluate the three models from the previous step with all non-agent implementation separately.
   (b) Evaluate the agent implementation with different summary models and Mistral as generation model.
3. **Comparison of results:** Compare the results from the baseline approach with the implementations to analyze how the implementations contribute to the overall results for different models.

4.6.1 Evaluation Models

To evaluate the generated responses, two models were utilized: Vicuna and Llama3.

Vicuna was chosen for tasks that required a longer context windows, such as assessing the faithfulness of the response, the answer relevancy, and the context precision. This means that Vicuna was used to determine how relevant the information in the retrieved chunks was to the query, ensuring that the responses were both accurate and contextually appropriate.

On the other hand, Llama3, which was found to be more accurate in scoring, was used for metrics that did not rely on context. These metrics included the answer correctness and the semantic similarity of the answer. Llama3 focused on whether the answers were correct and how similar they were to the expected responses, without considering the broader context in which the answers were given.

4.6.2 Evaluation Set

The evaluation set was designed to align with the Ragas framework. Developed in collaboration with a subject matter expert (SME) from Ericsson, it included 50 questions to assess the model's capability in understanding typical queries used in log file analysis. The evaluation set included the following four key components:

1. **Question:** A set of 50 questions varying in difficulty, to test the model's understanding.
2. **Ground Truth:** Desired answers for each question, established in collaboration with the SME.
3. **Answer:** Responses generated by the large language model being evaluated.
4. **Contexts:** Relevant data segments retrieved by the model based on the questions.

4.6.3 Analysis of Results

In this subsection, we detail the methods and approaches used to analyze the performance of various models and implementations. The analysis is divided into two main components: statistical analysis and visualization.

Statistical Analysis: For statistical analysis, we focused on three key descriptive statistics: mean, median and variance. These measures has been used to analyze the central tendency and variability of the performance metrics presented in section 3.6.2, under different experimental conditions. This approach has helped in understanding the overall performance and consistency of each model and implementation.

Visualization: To aid in the interpretation of results and to visually compare the effectiveness of different models and implementations, various graphical representations was used. These includes bar charts for comparison of performance metrics, line graphs to show trends over multiple runs, kernel density estimate to understand the univariate distribution of metrics for different implementations.

5
Results

In this chapter the results for each of the implementations are presented. All the results are evaluated with the evaluation set described in 4.6.2.

5.1 Evaluation of Baseline Approach

In this section the results for the baseline approach is presented. The results here represent what was achieveable using the most straightforward RAG approach. The details are presented in 4.5.1.

In Table 5.1 we see the evaluation results for the baseline approach. The table includes the mean, median and variance for each metric and model. The evaluation metrics considered are answer correctness, faithfulness, answer relevancy, answer similarity and context precision.

Table 5.1: Statistical Analysis of the baseline approach

Model	Metric	Mean	Median	Variance
Llama3	Answer Correctness	0.277	0.193	0.036
	Faithfulness	0.741	1.000	0.111
	Answer Relevancy	0.457	0.674	0.141
	Answer Similarity	0.706	0.724	0.019
	Context Precision	0.050	0.000	0.030
Mistral	Answer Correctness	0.279	0.199	0.030
	Faithfulness	0.845	1.000	0.081
	Answer Relevancy	0.454	0.686	0.145
	Answer Similarity	0.706	0.743	0.016
	Context Precision	0.051	0.000	0.031
Llama2	Answer Correctness	0.412	0.390	0.060
	Faithfulness	0.719	1.000	0.140
	Answer Relevancy	0.709	0.797	0.064
	Answer Similarity	0.765	0.787	0.015
	Context Precision	0.049	0.000	0.030

In Figure 5.1 we can see the results with a kernel density plot, showing the univariate distribution for each metric between the three models.

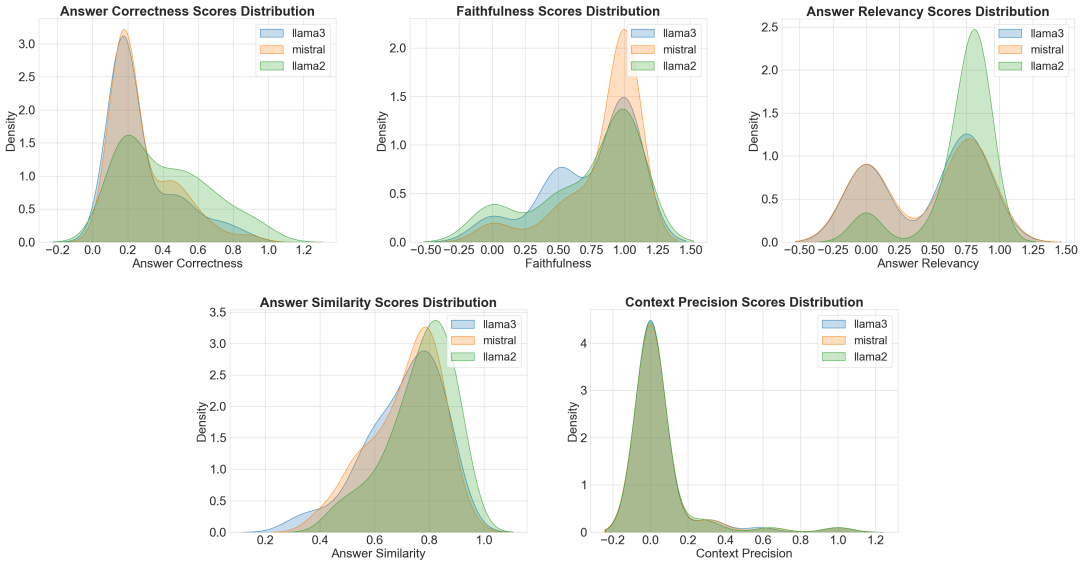


Figure 5.1: Kernel Density Estimate plot for the three models evaluated

In both the table and the figure we can see that the best performing model for the baseline approach is Llama2. It had the highest mean and median scors for answer correctness, answer relevancy and answer similarity. Context precision was low for all models, with no significant differences.

5.2 Results for Implementations

In this section, the results for each implementation, excluding agent-based implementations, are presented. Each implementation is presented in its own subsection, with the results reflecting the performance of each model evaluated with different implementations.

5.2.1 Implementation 1

Table 5.2 shows the statistcal results for implementation 1. These results represents how well the three models can handle the full document elements in pandas dataframe format, see 4.5.2. In the table we can see that all three models have similar performance. With slightly lower results from the Llama3 model.

Table 5.2 displays the statistical results for Implementation 1. These results illustrate the performance of the three models in handling full document elements in the pandas DataFrame format, as described in 4.5.2.

Table 5.2: Statistical Analysis of implementation 1

Model	Metric	Mean	Median	Variance
Llama3	Answer Correctness	0.489	0.435	0.103
	Faithfulness	0.681	0.750	0.124
	Answer Relevancy	0.634	0.755	0.083
	Answer Similarity	0.765	0.798	0.014
	Context Precision	0.06	0.000	0.005
Mistral	Answer Correctness	0.465	0.484	0.069
	Faithfulness	0.724	1.000	0.143
	Answer Relevancy	0.792	0.823	0.021
	Answer Similarity	0.776	0.820	0.017
	Context Precision	0.548	1.000	0.256
Llama2	Answer Correctness	0.550	0.524	0.069
	Faithfulness	0.729	1.000	0.117
	Answer Relevancy	0.732	0.794	0.048
	Answer Similarity	0.776	0.820	0.018
	Context Precision	0.080	0.000	0.007

In Figure 5.2 we can see the results with a kernel density plot for implementation 1, showing the univariate distribution for each metric between the three models.

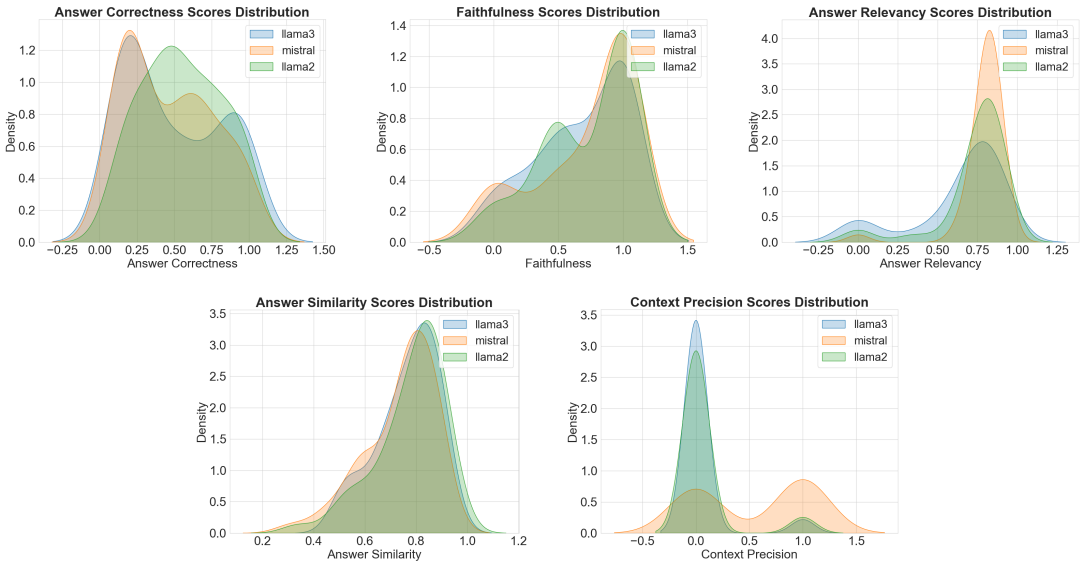


Figure 5.2: Kernel Density Estimate plot for each metric

From the results we can see that Llama 2 outperforms both Mistral and Llama 3 in answer correctness and faithfulness. Mistral shows higher score than Llama 2 and Llama 3 in context precision and answer relevancy. Both Mistral and Llama2 shows strong performance in answer similarity, with Mistral having a slightly lower variance.

5.2.2 Implementation 2

This section compares the results of Implementation 2, which evaluates the performance of the models on document elements divided into sub-tables using the pandas DataFrame format, see 4.5.3.

Table 5.3: Statistical Analysis of implemenation 2

Model	Metric	Mean	Median	Variance
Llama3	Answer Correctness	0.529	0.478	0.097
	Faithfulness	0.666	0.958	0.176
	Answer Relevancy	0.686	0.740	0.051
	Answer Similarity	0.786	0.816	0.015
	Context Precision	0.420	0.000	0.249
Mistral	Answer Correctness	0.616	0.649	0.093
	Faithfulness	0.833	1.000	0.083
	Answer Relevancy	0.620	0.764	0.105
	Answer Similarity	0.787	0.807	0.014
	Context Precision	0.429	0.000	0.250
Llama2	Answer Correctness	0.534	0.530	0.063
	Faithfulness	0.817	1.000	0.099
	Answer Relevancy	0.731	0.804	0.052
	Answer Similarity	0.801	0.833	0.016
	Context Precision	0.490	0.000	0.255

In Figure 5.3 we can see the results with a kernel density plot, showing the univariate distribution for each metric between the three models.

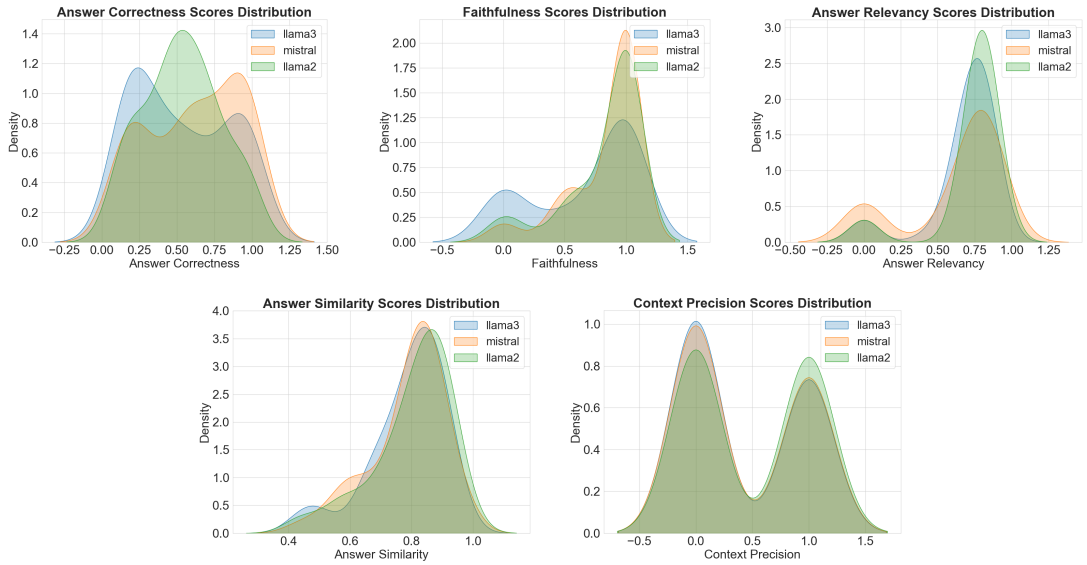


Figure 5.3: Kernel Density Estimate plot for the three models evaluated with implementation 2

The results indicate that Mistral continues to perform well relative to the other

models, whereas Llama3 shows comparatively lower performance. For all three models, the scores for context precision, faithfulness, and answer correctness has increased compared to the previous implementation.

5.2.3 Implementation 3

The results for implementation 3 represents how well the three models performs on full-size document elements formatted in HTML, see 4.5.4.

Table 5.4: Statistical Analysis of implementation 3

Model	Metric	Mean	Median	Variance
Llama3	Answer Correctness	0.530	0.541	0.100
	Faithfulness	0.750	1.000	0.148
	Answer Relevancy	0.569	0.686	0.115
	Answer Similarity	0.748	0.772	0.016
	Context Precision	0.666	1.000	0.233
Mistral	Answer Correctness	0.519	0.522	0.066
	Faithfulness	0.735	1.000	0.133
	Answer Relevancy	0.568	0.749	0.111
	Answer Similarity	0.735	0.774	0.019
	Context Precision	0.320	0.000	0.222
Llama2	Answer Correctness	0.455	0.448	0.068
	Faithfulness	0.689	0.667	0.111
	Answer Relevancy	0.736	0.821	0.056
	Answer Similarity	0.787	0.810	0.015
	Context Precision	0.480	0.000	0.255

In Figure 5.4 we can see the results with a kernel density plot, showing the univariate distribution between the three models for each metric.

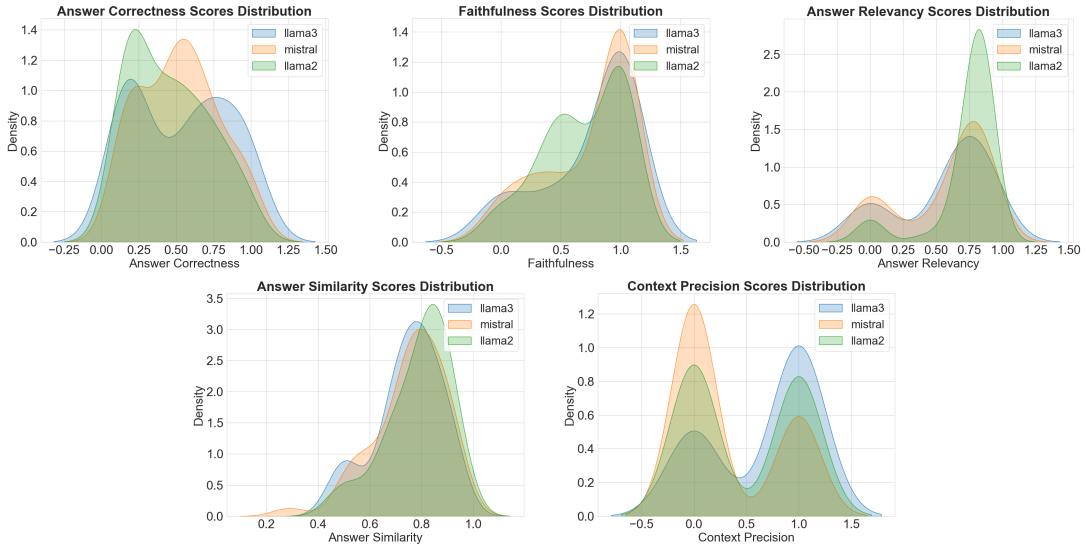


Figure 5.4: Kernel Density Estimate plot for the three models evaluated with implementation 3

In the results presented here, Llama3 shows higher scores across all metrics compared to Mistral. Additionally, it outperforms Llama2 in answer correctness, faithfulness, and context precision. In comparison to previous implementations, the scores for Mistral and Llama2 have generally decreased, whereas Llama3's scores have increased. Furthermore, when comparing context precision with implementation 1, the full-size document elements in HTML format achieve higher context precision scores compared to those in pandas DataFrame format.

5.2.4 Implementation 4

Here we present the results for the last implementation that is not agent based. The results represents how well the three models performs on document elements divided into sub-elements in HTML format, see 4.5.5.

Table 5.5: Statistical Analysis of implementation 4

Model	Metric	Mean	Median	Variance
Llama3	Answer Correctness	0.639	0.696	0.083
	Faithfulness	0.838	1.000	0.074
	Answer Relevancy	0.740	0.777	0.040
	Answer Similarity	0.792	0.826	0.016
	Context Precision	0.660	1.000	0.229
Mistral	Answer Correctness	0.603	0.626	0.082
	Faithfulness	0.927	1.000	0.026
	Answer Relevancy	0.666	0.764	0.074
	Answer Similarity	0.790	0.822	0.015
	Context Precision	0.723	1.000	0.020
Llama2	Answer Correctness	0.513	0.517	0.070
	Faithfulness	0.740	1.000	0.123
	Answer Relevancy	0.696	0.782	0.075
	Answer Similarity	0.790	0.835	0.014
	Context Precision	0.500	0.500	0.025

In Figure 5.5 we can see the results with a kernel density plot, showing the univariate distribution for each metric between the three models evaluated.

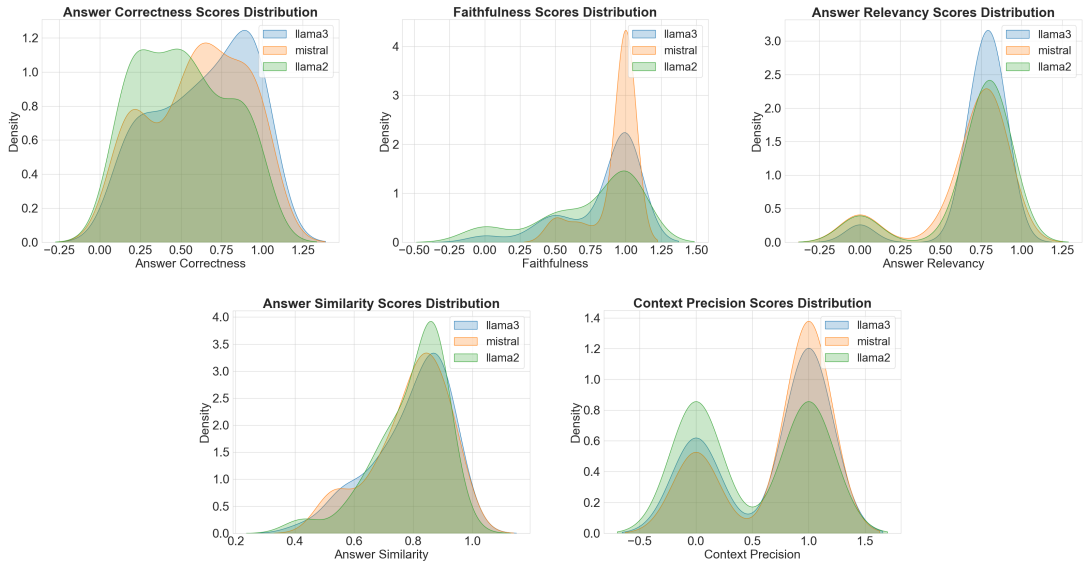


Figure 5.5: Kernel Density Estimate plot for the three models evaluated with implementation 4

5.3 Results for Agent-based Solutions

Of the three models we have evaluated for the implementations, Mistral is the only model capable of implementing an agent-based approach. For that reason, in this section, we are presenting the results for the agent implementation using different summary models. In one approach, the summaries are generated with the Llama3 model, and in the other approach, the summaries are generated with a fine-tuned version of Llama3, which we call Llogama3.

Table 5.6: Statistical Analysis of agent-based implementations

Summary Model	Metric	Mean	Median	Variance
Llama3	Answer Correctness	0.448	0.220	0.116
	Faithfulness	0.503	0.500	0.169
	Answer Relevancy	0.525	0.592	0.097
	Answer Similarity	0.691	0.665	0.037
	Context Precision	0.040	0.000	0.039
Llogama3	Answer Correctness	0.424	0.278	0.101
	Faithfulness	0.406	0.333	0.184
	Answer Relevancy	0.478	0.590	0.120
	Answer Similarity	0.688	0.629	0.040
	Context Precision	0.020	0.000	0.020

In Figure 5.6 we can see the results with a kernel density plot, showing the univariate distribution between the two summary models with Mistral as generation model for each metric.

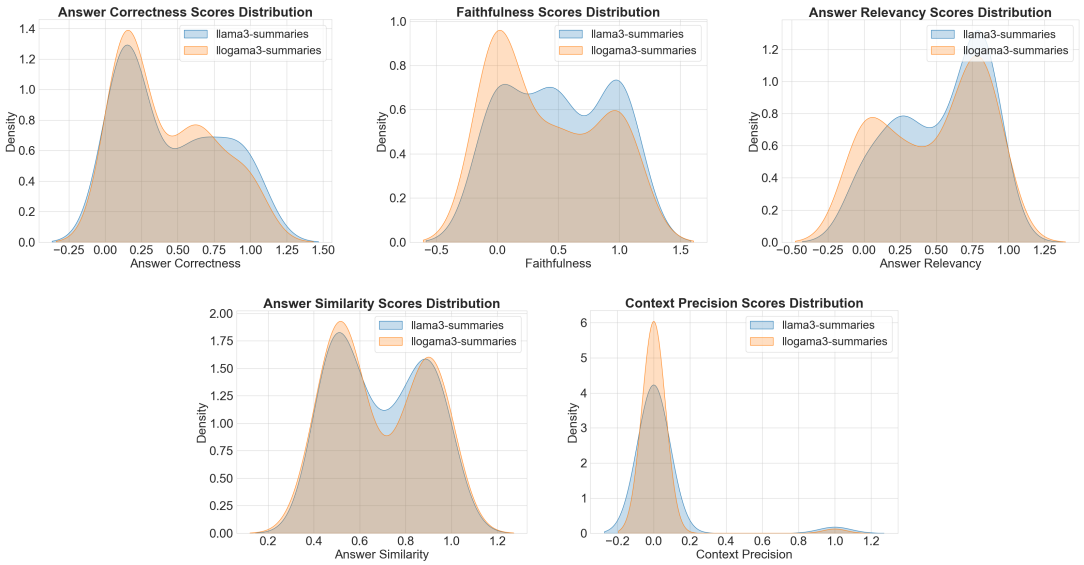


Figure 5.6: Kernel Density Estimate plot for the two summary models evaluated with agent implementations

5.4 Comparison

In this section, we will present results comparing different implementations with each other, as well as comparing our best results with the baseline to identify improvements in overall scores.

Figure 5.7 displays the combined weighted scores of all metrics for each implementation. The results indicate that the best performing model is Implementation 4 with Llama3. The worst performing implementations are the agent-based ones. Excluding the agent-based implementations, the baseline approach with Llama3 performs the poorest. Therefore, we have chosen to compare the worst non-agent implementation (Baseline Llama3) with the best performing non-agent implementation (Implementation 4 with Llama3).

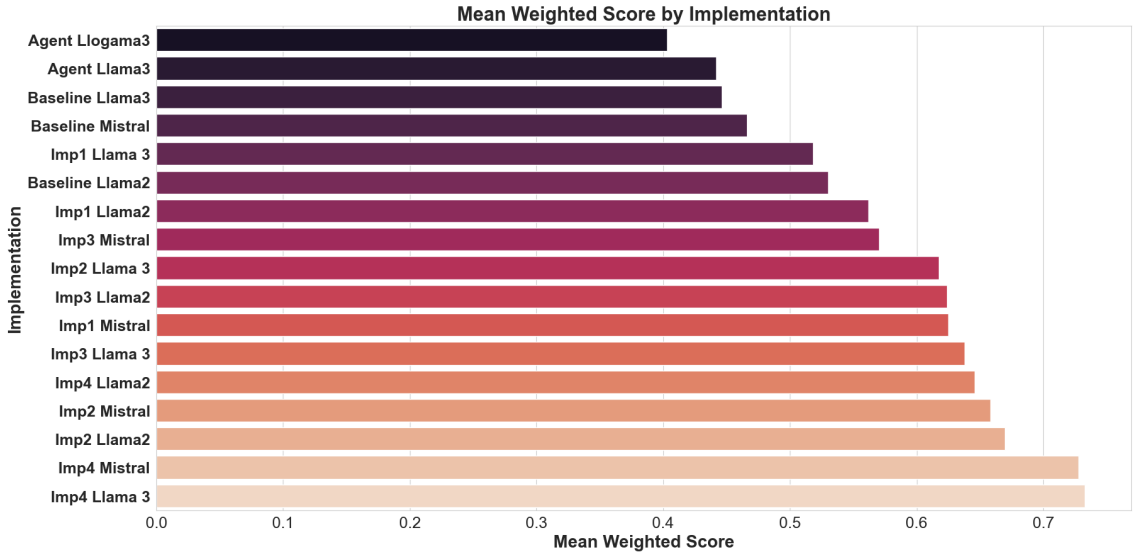


Figure 5.7: Weighted mean of all scores for every implementation

In Figure 5.8 we can see the results with a kernel density plot, showing the univariate distribution for the evaluation metrics between the baseline implementation with Llama3 and implementation 4 with Llama3.

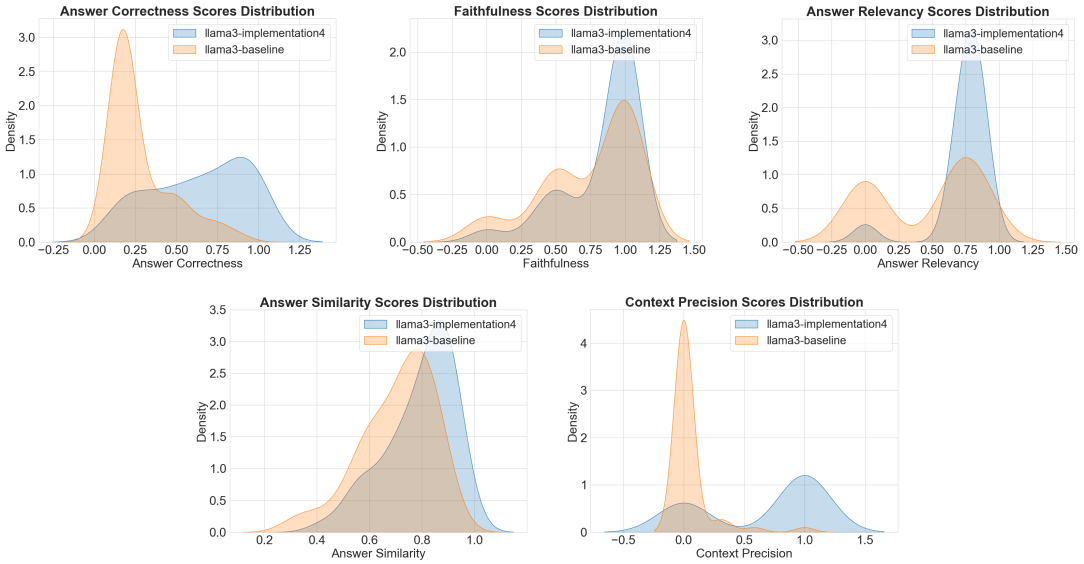


Figure 5.8: Kernel Density Estimate plot for the metrics considered

5.5 Performance by Question

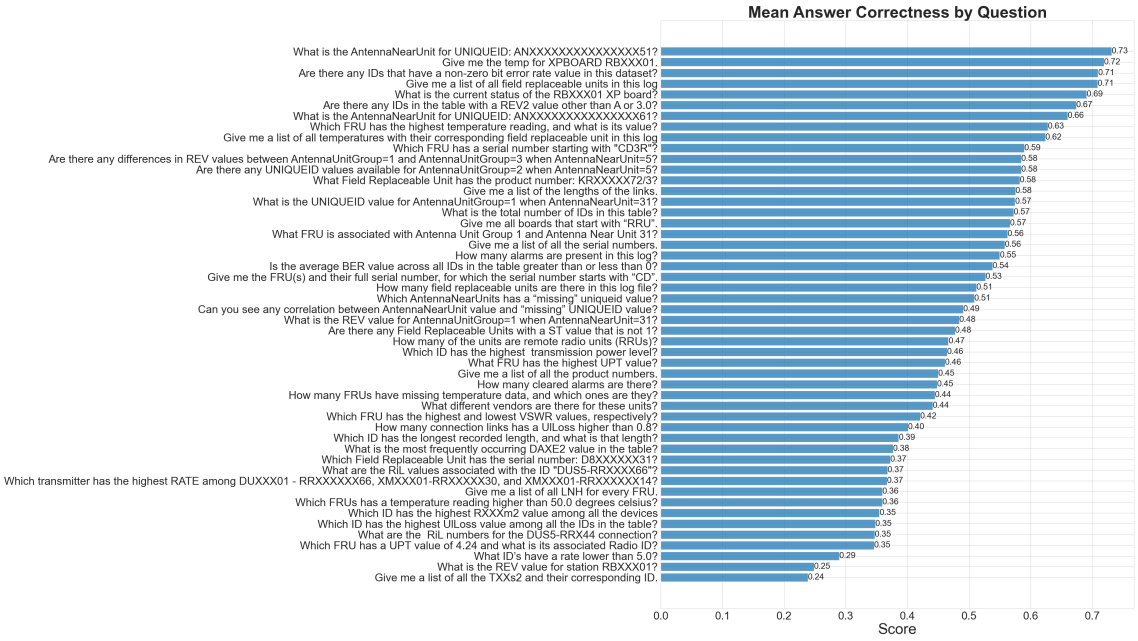


Figure 5.9: Average answer correctness by question, showing the questions that on average has the highest answer correctness

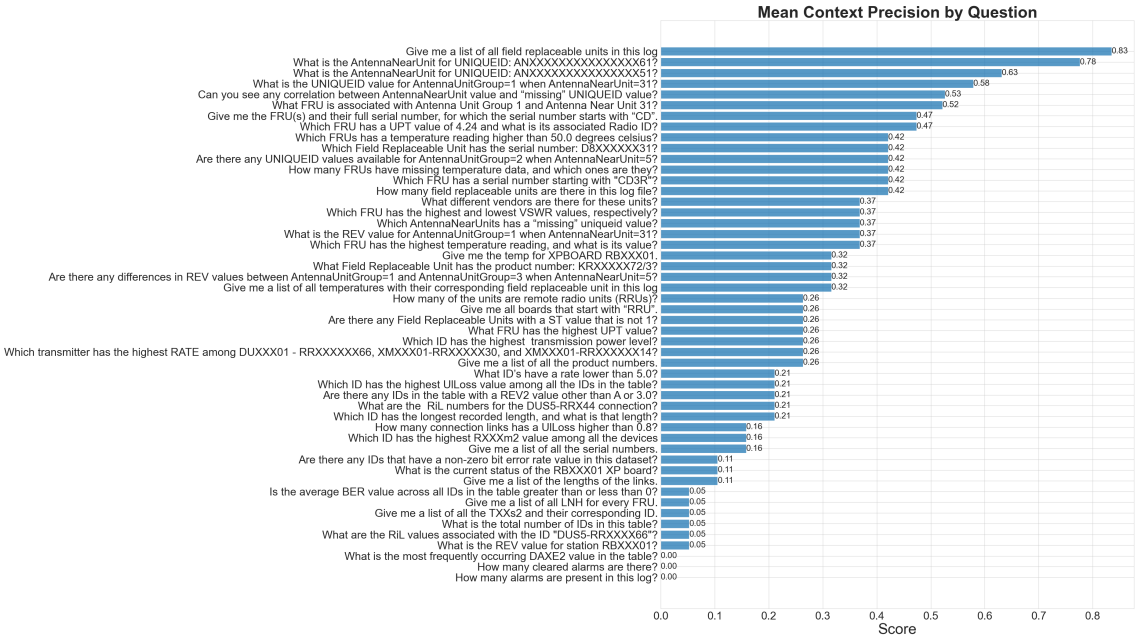


Figure 5.10: Average context precision by question, showing the questions that on average have the highest context precision

6
Discussion

In this chapter, we dive into the results and the important decisions made throughout the project, examining the relation between context length and accuracy, the performance of agent-based implementations, the outcomes of fine-tuning, the reliability of evaluation metrics, and the comparative analysis of different implementations. This chapter also addresses future work directions based on our findings.

6.1 Implementation Performance

From the final implementation comparison presented in Figure 5.7, Implementation 4 with Llama3 as model had the best performance according to the evaluation set. Both in regards of weighted score but also in all metrics individually. The implementation achieved the highest mean score for answer correctness, with 0.639. Which also is the highest increase of answer correctness comparing with the baseline evaluation, where Llama 3 had a mean score of 0.277. This can be explained by the implementations use of HTML sub-elements, that allows the model to process data in more manageable and contextually rich segments. The structured format of HTML enhances the model's ability to retrieve and piece together relevant information accurately. One thought is that this benefit likely arises because the model was trained on extensive web data, resulting in better understanding of markup languages such as HTML. For the same reason answer relevancy improved from 0.457 in the baseline to 0.740 with Implementation 4.
The faithfulness metric, which measure the degree to which the model's output remains true to the source information saw improvements from 0.741 in the baseline approach to 0.838 in Implementation 4.

Throughout all implementations, the metric for answer similarity has maintained high scores, but with the highest score in Implementation 4. This trend can be observed in the detailed performance metrics of each implementation in Table 6.1.

Table 6.1: Answer Similarity Scores Across Implementations with Llama3

Implementation	Baseline	1	2	3	4
Mean Answer Similarity	0.706	0.765	0.786	0.748	0.792

These consistently high scores suggest that the models are proficient in generating responses that closely match the expected answers in terms of wording and struc-

ture. However, a high answer similarity score does not guarantee that the generated answer is correct. Similarity in wording can occur even if the factual content or the context of the response is incorrect. It is also important to mention that a very high score in answer similarity indicates that a lot of words are similar and that the answer could be correct, but it should be complemented with other metrics such as answer correctness and faithfulness.

Context precision sees a dramatic improvement, with Implementation 4 achieving 0.660 versus the baseline's 0.050. This highlights the effectiveness of Implementation 4 in maintaining contextual accuracy and providing more precise information within the given context. This is explained by the contextual chunking together with generated summaries for each chunk enhances the retrieval mechanism and assists it in finding the correct chunk based of a query. Adding a Multi-vector retriever enabled the storing of multiple vectors for each chunk, further assisting the retrieval of the correct chunk.

Interestingly, context precision slightly decreased from 0.666 in Implementation 3 to 0.660 in Implementation 4. While still high, this slight decrease may suggest that while the overall context is maintained well, the sub-tables introduces some complexity in maintaining exact context precision. This can be explained by the choice of keeping k=1 for the retrieval. Some questions in the evaluation set are designed to require answers from two separate chunks, while the table is in full size this is not an issue but when the document elements are split into sub-tables, two chunks could mean two rows from the same table. With k fixed at 1, it only retrieves the most similar chunk, meaning that the model is not able to answer the second question in the evaluation question since the chunk is not retrieved. The reason for this decision is shown in Table 6.2, where a value of k higher than 1 generally results in more hallucinations from the model, ultimately resulting in lower answer correctness.

Table 6.2: Mean Answer Correctness vs. k Value for Implementation 4 with Llama3

k Value	1	2	3
Mean Answer Correctness	0.639	0.605	0.585

Comparing the baseline results for all three models, Llama2 outperforms Mistral and Llama3 in all metrics. This outcome was somewhat expected and can be explained by the size differences of the models. The Llama2 model has 13B parameters while Llama3 and Mistral has 8B and 7B respectively. On top of that they are all quantized to 4 bits. Therefore quantization treats them all similarly.

6.2 Context or Accuracy?

The challenge of context length has been a major obstacle for this project. Nearly every problem we've faced has been related to either context length or accuracy.

Our results points out a clear dilemma: prioritizing context or accuracy. The project needs long contexts to allow the models to fully understand the content through our content-aware chunking method. However, using a model with a 16k context window generally results in lower performance compared to models with 4k-8k context windows. While there are better-performing models with longer context windows, using them exceeds the project's hardware limitations.

6.3 Agent Based Implementations

The agent-based implementations (Implementations 5 & 6) did not perform well in evaluation, unlike in testing. This isn't too surprising because these agents tend to either give a perfect answer or no answer at all. The module was released during the course of the project and is very lightly (if at all) documented. Based on observed examples, there are two common issues:

1. The retriever fetches the wrong chunks, causing problems for any RAG approach.
2. The model struggles with coding in Python, and keeping the correct format. So even if it knows what to do, it does not always succeed.

Although agent-based approaches received low scores in evaluation, they could still be a viable option. These approaches excel in an area we didn't extensively test: combining different results from various chunks.
Addressing the poor python code, could be as simple as changing model to one that has been trained on more code.

6.4 Fine Tuning Results

The fine-tuned version of Llama3 did not perform as well as we had hoped. Although it seemed to catch more domain-specific terms, this did not lead to any noticeable improvement, in fact, its overall performance was worse than with Llama3.

Due to time constraints, this model was only tested in combination with Implementation 6, which utilized metadata to build summaries. This limited time for testing may have impacted the results. It would have been interesting to see if Llama3 could have helped increase context in other implementations that lacked metadata for summary building, particularly those that had chunks divided into 'sub-tables.' Exploring this could have provided good insights into the model's potential to perform better in other situations.

Additionally, the fine-tuning process was done under time pressure, with the goal of generating results as quickly as possible. This urgency led to the use of a very small dataset, generated with the assistance of large language models. The limited size of this dataset likely impacted the effectiveness of the fine-tuning, potentially leading to the bad performance observed. Given more time, a larger and more diverse dataset could have been used, potentially leading to better outcomes.

6.5 Reliability in Evaluation Metrics

During the course of the project it was evident that the metrics have tendencies to be unreliable. This came from the context window problem discussed in section 6.2.

Table 6.3: Sample from evaluation results, showing the desired and actual output, and its correctness score with Vicuna & Llama3

Evaluation Model	Ground truth	Answer	Answer Correctness
Vicuna	No, the table shows a "Missing" value.	"No"	0,140898
Llama3	No, the table shows a "Missing" value.	"No"	0,640898

As can be seen from the example in table 6.3, the ground truth is "No, the table shows a "Missing" value" and the generated answer "No". This results in a answer correctness of 0,14. Even though the response holds the similar sentiment, the self evaluation model, vicuna could not see this.
To compensate for this, the answer correctness, and answer semantic similarity were re-generated with Llama3. After re-generation the answer correctness increased from 0.14 to 0.64.
This shows that even if the results are actually adequate, they might be portrayed as if they were poor, and of course also, the opposite.

6.6 Other Work

In papers like [25] which focuses more on anomaly detection rather than a chatbot, the researchers evaluate their work using benchmark datasets such as BGL and Thunderbird [11]. These datasets are specifically designed for log anomaly detection, allowing the researchers to use standard metrics like the F1 score for evaluation. With the F1 score being a common and straightforward metric in binary classification problems, it is perfectly suitable for evaluating the effectiveness of anomaly detection models.

However, this project differs significantly in its objectives and context. The primary focus here is on troubleshooting faults in radio units using domain-specific system log data. Unlike the general-purpose anomaly detection datasets used in previous studies, the log data in this project is highly specialized and tailored to the specific needs of radio unit fault analysis. Therefore, relying on generic metrics like the F1 score would not adequately capture the specifics of the results. Apart from the complexity, a new definition of the F1 score would have to be found to fit the generated responses, since it lacks two labels. Given these differences the project had to adopt a more appropriate testing method tailored to the domain-specific nature of the data, and desired output.

6.7 The Questions in the Dataset

In figure 5.9, we see the average answer correctness for all implementations. This means the questions at the top are those with the highest answer correctness overall, while the ones at the bottom tend to lack correct answers. Despite the differences in model performance on this metric, we can still gain insights into the types of questions the model tends to get right or wrong.

At the top, many questions contain specific product numbers or serial numbers. These strings can be directly paired with specific chunks of information. If the correct chunk is matched, the generator has a good chance of producing a correct response.

Figure 5.10 displays the average context precision for all implementations. This plot helps us identify which chunks are difficult to use when trying to reach the ground truth from the retrieved contexts. Notably, the two questions at the very bottom both refer to the same chunk, which is one of the longer ones. The most likely explanation is that the evaluation model struggles to assess the context precision for these questions due to the context window problem.

6.8 Future Work

Analyzing our results reveals that the pre-processing approach has led to improvements across all models when compared to the baseline approach, demonstrating the effectiveness of our implementations. This observation naturally raises the question: "What outcomes might have been achieved with a more advanced model from the start of the project?" The data suggests that utilizing a superior model could have yielded even higher overall scores.

However, it is important to note that the results for Llama2 present a counterargument. Llama2 achieved the highest score in the baseline approach, which consequently resulted in the smallest overall score improvement with our implementations. This indicates that while advanced models hold promise, the extent of their impact may vary depending on the specific model and implementation context.

Unfortunately, due to constraints on the size of language models we could use, we were unable to fully explore this question within the scope of this project. Moving forward, it is of interest to use more advanced models with our implementations to gain deeper insights and make further improvements.

Future research should focus on integrating state-of-the-art language models to assess their impact on the performance of retrieval-augmented generation pipelines. By doing so, we can better understand the upper bounds of achievable performance.

7
Conclusion

Our research has aimed to address two primary questions: the effectiveness of pre-trained large language models in analyzing and summarizing system log files, and methods to enhance chatbot performance through enriched context.

We have demonstrated that pre-trained LLMs can indeed be utilized to analyze and summarize system log files. The models were able to extract information from different types of log data, identifying log events and summarizing log file sections with a reasonable degree of accuracy. Our findings suggest that LLMs can to a promising extent, process log data, and provide insights to a user. This application of LLMs presents a possible approach to automating log file analysis, potentially improving efficiency in system troubleshooting.

In addressing our second research question, we explored various methods to enrich chatbot input data with additional context. By incorporating domain-specific terms and preprocessing data to capture the complexities of niche areas, we found significant improvements in the chatbot's performance. Enhanced context allowed the chatbot to better grasp and respond to user queries, demonstrating a deeper understanding of specialized content. This enrichment of input data underscores the importance of context for LLMs, and suggests that domain-specific preprocessing can lead to more effective and accurate conversational agents.

The implications of our research are twofold. First, the successful application of LLMs to log file analysis could lead to more automated and intelligent system monitoring tools. Second, enriching chatbot input data with contextual information can substantially improve their performance, making them more useful in specialized domains.

However, our study also highlights some limitations. For instance, the effectiveness of LLMs in summarizing log files can vary depending on the quality and structure of the log data. Similarly, the methods of adding context would require source-specific refinement to work optimally.
In conclusion, our research demonstrates the potential of LLMs in log file analysis and the importance of context in chatbot performance. Although niched, the work in this report could contribute to easy access insight in areas with complex data.

Bibliography

[1] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, p. 386, 1958.

[2] J. Weizenbaum, "Eliza—a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[3] J. Denker, W Gardner, H. Graf, *et al.*, "Neural network recognizer for hand-written zip code digits," *Advances in Neural Information Processing Systems*, vol. 1, pp. 323–330, 1988.

[4] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[5] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[6] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer, "The mathematics of statistical machine translation: Parameter estimation," *Computational Linguistics*, vol. 19, no. 2, pp. 263–311, 1993.

[7] M. Wahde, "Introduction to neural networks," 1996.

[8] V. N. Vapnik, V. Vapnik, *et al.*, "Statistical learning theory," 1998.

[9] C. Manning and H. Schutze, *Foundations of statistical natural language processing*. MIT press, 1999.

[10] A. Singhal, "Modern information retrieval: A brief overview," *IEEE Data Engineering Bulletin*, vol. 24, no. 4, pp. 35–43, 2001.

[11] A. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, IEEE, 2007, pp. 575–584.

[12] M. Collins, "Probabilistic context free grammars," 2011, Course Notes for Natural Language Processing (COMS E6998-10). [Online]. Available: `https://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/pcfgs.pdf`.

[13] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference On Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[14] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," 2016. arXiv: `1607.01759 [cs.CL]`.

[15] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, pp. 5999–6009, 2017.

[16] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[17] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, IEEE, 2019, pp. 3009–3018.

[18] T. Brown, B. Mann, N. Ryder, *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[19] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[20] F. Cuconasu, G. Trappolini, F. Siciliano, *et al.*, "The power of noise: Redefining retrieval for rag systems," 2021. arXiv: 2401.14887 [cs.CL].

[21] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models," 2021. arXiv: 2106.09685 [cs.CL].

[22] H. Liu, D. Tam, M. Muqeeth, *et al.*, "Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1950–1965, 2022.

[23] N. Ding, Y. Qin, G. Yang, *et al.*, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, 2023.

[24] A. Q. Jiang, A. Sablayrolles, A. Mensch, *et al.*, "Mistral 7b," 2023. arXiv: 2310.06825 [cs.CL].

[25] J. Pan, S. L. Wong, and Y. Yuan, "Raglog: Log anomaly detection using retrieval augmented generation," 2023. arXiv: 2311.05261 [cs.CR].

[26] Roie Schwaber-Cohen, *Chunking strategies for llm applications*, Accessed: 2024-04-05, 2023. [Online]. Available: https://www.pinecone.io/learn/chunking-strategies.

[27] H. Touvron, L. Martin, K. Stone, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. arXiv: 2307.09288 [cs.CL].

[28] M. AI, *Meta ai introduces llama-3: Advancements in large language models*, Accessed: 2024-05-16, 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3/.

[29] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[30] Hopsworks, *What is a context window for llms?* Accessed: 2024-04-10, 2024. [Online]. Available: https://www.hopsworks.ai/dictionary/context-window-for-llms.

[31] RAGAS, *Answer correctness*, Accessed: 2024-05-16, 2024. [Online]. Available: https://docs.ragas.io/en/stable/concepts/metrics/answer_correctness.html.

[32] RAGAS, *Answer relevancy*, Accessed: 2024-05-16, 2024. [Online]. Available: https://docs.ragas.io/en/stable/concepts/metrics/answer_relevance.html.

[33] RAGAS, *Context precision*, Accessed: 2024-05-16, 2024. [Online]. Available: https://docs.ragas.io/en/stable/concepts/metrics/context_precision.html.

[34] RAGAS, *Faithfulness*, Accessed: 2024-05-16, 2024. [Online]. Available: https://docs.ragas.io/en/stable/concepts/metrics/faithfulness.html.

[35] RAGAS, *Semantic similarity calculation*, Accessed: 2024-05-16, 2024. [Online]. Available: https://docs.ragas.io/en/stable/concepts/metrics/semantic_similarity.html#calculation.

[36] Y. Sui, M. Zhou, M. Zhou, S. Han, and D. Zhang, "Table meets llm: Can large language models understand structured table data? a benchmark and empirical study," in *The 17th ACM International Conference on Web Search and Data Mining (WSDM '24)*, 2024, pp. 645–654.