Informatics

Comparative Analysis of Retrieval Augmented Generator and Standalone Large Language Models

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Tin Oroz
Matrikelnummer 12122082

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dipl.-Ing. Dr.techn. Georg Gottlob

Wien, 19.August 2024

	[redacted]	[redacted]
	Tin Oroz	Georg Gottlob

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Informatics

Comparative Analysis of Retrieval Augmented Generator and Traditional Large Language Models

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Tin Oroz
Registration Number 12122082

to the Faculty of Informatics

at the TU Wien

Advisor: O.Univ.Prof. Dipl.-Ing. Dr.techn. Georg Gottlob

Vienna, August 19, 2024

[redacted] [redacted]

Tin Oroz Georg Gottlob

Declaration of Authorship

Tin Oroz

I hereby declare that I have written this Master Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.
I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

[redacted]

Vienna, August 19, 2024

Tin Oroz

Danksagung

Ich möchte meinem Betreuer, Professor Georg Gottlob, für seine Anleitung und Unterstützung während dieser Arbeit danken. Sein Wissen und sein Feedback waren wesentlich für die Ausrichtung meiner Arbeit.

Ich bin auch dem Team von happtiq dankbar für die Bereitstellung der Cloud-Umgebung, die diese Forschung möglich gemacht hat. Mein besonderer Dank gilt Muris Kavlak für seine Hilfe bei der Einrichtung der Cloud und seine Unterstützung während des Projekts und der Auswertung. Ich schätze auch die Beiträge von Marcel Fehring und Veronika Priesner zum Evaluierungsprozess.

Ich danke meiner Familie für ihre Unterstützung während meines Studiums, insbesondere meiner Mutter, meinem Vater und meinem Bruder. Mein Dank geht auch an meine Freundin Dora für ihre Geduld und ihr Verständnis. Schließlich möchte ich mich bei all meinen Freunden für ihre Ermutigung auf meinem Weg bedanken.

Acknowledgements

I would like to thank my mentor, Professor Georg Gottlob, for his guidance and support throughout this thesis. His knowledge and feedback were essential in shaping the direction of my work.

I am also thankful to the team at happtiq for providing the cloud environment that made this research possible. Special thanks to Muris Kavlak for his help with the cloud setup and his support during the project and evaluation. I also appreciate the contributions of Marcel Fehring and Veronika Priesner to the evaluation process.

I am grateful to my family for their support during my studies, especially my Mom, Dad and Brother. My thanks also go to my girlfriend Dora for her patience and understanding. Lastly, I would like to thank all my friends for their encouragement along the way.

Kurzfassung

Ziel dieser Arbeit ist es, eine vergleichende Leistungs- und Einrichtungsanalyse von Retrieval Augmented Generator (RAG) [LPP+20] Architekturen und eigenständigen Large Language Models (LLMs) durchzuführen, wobei sich die Tests auf spezielle und spezifische Bereiche konzentrieren. Large Language Models sind fortschrittliche Algorithmen für maschinelles Lernen, die auf Textdatensätzen im Petabyte-Bereich [LCL+24] trainiert wurden, um menschenähnliche Texte zu erzeugen. Sie zeigen hervorragende Leistungen bei einer Vielzahl von Sprachaufgaben, haben aber manchmal Schwierigkeiten mit aktuellen, domänenspezifischen oder privat verfügbaren Informationen. Andererseits stellen RAG-Modelle einen innovativen Ansatz in diesem Bereich dar. Sie kombinieren die breite Wissensbasis von LLMs mit Echtzeit-Informationsbeschaffung aus zusätzlichen Datenquellen. Dieses hybride Modell zielt darauf ab, die Antwortqualität zu verbessern, indem aktuelle und relevante Informationen bereitgestellt werden. Die Studie konzentriert sich auf die Fähigkeit der RAG, die Breite des Wissens von LLMs zu nutzen und gleichzeitig die Aktualität und Korrektheit externer Daten einzubeziehen. Sie zielt darauf ab, die Verbesserungen oder Einschränkungen von RAG im Vergleich zu eigenständigen LLMs in Bezug auf Antwortgenauigkeit, Antwortzeit und Berechnungseffizienz aufzudecken. Diese Forschung soll Einblicke in die Leistung dieser beiden unterschiedlichen Systeme in einem spezialisierten Bereich geben. Die Ergebnisse zeigen, dass die RAG eine signifikante Verbesserung der Antwortqualität bei einer relativ geringen Erhöhung der Antwortzeit und der Rechenlast aufweist. Die spezialisierte Domäne für diese Forschung konzentrierte sich auf das Wissen rund um die Google Cloud Platform-Technologien, die darauf abzielen, die Bedürfnisse der Spezifität und Aktualität einer bestimmten Domäne zu replizieren.

Abstract

This thesis goal is to achieve a comparative performance and setup analysis of Retrieval Augmented Generator (RAG) [LPP+20] architectures and standalone Large Language Models (LLMs), with a testing being focused in specialized and specific domains. Large Language Models, are advanced machine learning algorithms trained on close to petabyte scale [LCL+24] text datasets to generate human like text. They have outstanding performance in a wide variety of language tasks, but sometimes struggle with up-to-date, domain-specific information or privately available information. On the other hand, RAG models represent an innovative approach in the field. They combine the wide knowledge base of LLMs with real-time information retrieval from additional data sources. This hybrid model aims to improve response quality by providing up-to-date and relevant information. The study focuses on the ability of RAG to use the breadth of knowledge from LLMs while incorporating the currentness and correctness of external data. It aims to uncover the improvements or limitations of RAG compared to standalone LLMs, in terms of answer accuracy, response time, and computational efficiency. This research aims to provide insights into how these two distinct setups perform in a specialized domain. The results showed how RAG had a significant improvement in answer quality, while having a relatively small increase in response time and computational load. Specialized domain for this research focused on knowledge surrounding Google Cloud Platform technologies, which aim to replicate the needs of specificity and currentness of a particular domain.

Contents

CHAPTER 1

Introduction

1.1 Motivation

In the dynamic field of natural language processing (NLP), Large Language Models (LLMs) like LLAMA2 [TMS+23] have demonstrated outstanding capabilities in generating human-like text based on a wide array of general information. However, their performance is often limited by the width of their training data, particularly when handling queries that need really specific and correct information. This limitation has led to the idea of a Retrieval Augmented Generator (RAG) [LPP+20] approach, a setup that combines the wide knowledge base of LLMs with relevant information retrieved from external data sources.

The need for RAG approaches comes up from the need to bridge the gap between the wide, but sometimes outdated, generalized or wrong information in LLMs [HYM+23], and the specialized, up-to-date knowledge required in specific domains. By integrating external databases, RAG can access and incorporate the latest, most relevant information, enhancing the accuracy and relevance of the responses, especially in fast changing fields like cloud technology. In this study, the external data integrated by RAG is represented by the continuously updating documentation of Google Cloud services [goo]. However, it's important to note that this particular data set is chosen for its complexity and constant evolving, but could be easily substituted with other data sources without affecting the core focus of the study.

Despite the obvious advantages of RAG, there is a significant gap in research of doing an comprehensive comparison with standalone LLMs in practical applications. The study focuses primarily on the aspect of accuracy or in other words correctness in the question answering setup with questions relating to Google cloud services. Additionally other aspects of comparison will be the comparison of response time, and computational efficiency of both setups. The aim is to understand how RAG models can leverage the broad knowledge of LLMs and combine it with specialized, up-to-date information from external data sources to deliver more accurate, relevant, and correct responses in specialized domains and how "expensive" is that in terms of computation and time.

1.2 Research Questions

1. **How does the quality of responses compare between RAG and standalone LLMs when applied to queries related to a specific domain?**

   Given the challenge in quantifying the accuracy of responses from LLMs, this question will involve domain experts reviewing and rating the quality of the responses. Experts in Google Cloud services will assess the answers for their accuracy, relevance, and overall quality. This approach ensures a more reliable and practical evaluation of the models' knowledge in a specialized domain.

2. **How does the response time compare between RAG and standalone LLMs?**

   This question focuses on the response time of RAG compared to standalone LLMs. It will measure how quickly each model can provide accurate answers and how consistent is that time, a important factor in real-world applications where quick information is crucial.

3. **How does the computational load of RAG compare with standalone LLMs, when delivering an answer?**

   This question evaluates the computational and energy efficiency of RAG versus standalone LLMs. The focus will be on the resources required to run each setup, considering the growing need for sustainable and cost-effective AI solutions.

1.3 Expected outcome

The primary goal of this research is to determine whether it is possible to deepen the knowledge of a foundational language model in a specific and specialised domain. Our domain will be Google Cloud services, but in this setup domain is easily interchangeable. This integration aims to significantly improve the quality of the responses provided by the model, ensuring that they are more accurate and relevant. In achieving this, the study also seeks to discover, in what degree, does the efficiency of the model in terms of response time and computational resources change. An expected outcome is an improved quality of answers, demonstrating an advancement over standalone Large Language Models (LLMs) without a substantial increase in response times, and computational and energy efficiency.

1.4 Thesis structure

This thesis is structured as following to systematically explore the implementation of Large Language Models (LLMs) and Retrieval Augmented Generator (RAG) and to evaluate their performance with a focus on a domain of Google Cloud services:

- **Chapter 1: Introduction** — This chapter introduces the thesis by providing the motivation behind the problem. The research questions are introduced, aiming to compare the effectiveness of RAG and LLMs in terms of response quality, time, and computational efficiency alongside a statement of expected outcomes. This chapter also includes current thesis structure.
- **Chapter 2: Theoretical framework** — This chapter provides the necessary background knowledge on the fundamental technologies used in this research. It starts with the basics of Machine Learning and progresses through to more advanced topics such as Neural Networks, Deep Learning for NLP, Transformers and Attention Mechanisms, Large Language Models, Vector databases, and Retrieval Augmented Generation. This foundational knowledge is crucial for enabling the comparing of the two systems in this research.
- **Chapter 3: Related work** — This chapter reviews existing work on RAG systems, work on evaluation of their performance, and identifies gaps in the state of the art.
- **Chapter 4: Methodology** — This chapter explores the methodology in the research, going over data collection, implementation of the RAG architecture and the retrieval mechanisms, deployment of both systems and evaluation methods and metrics.
- **Chapter 5: Results and Analysis** — This chapter includes the presentation of the empirical results from the research. This includes analysis of the quality of responses, response time comparisons, and assessments of computational and energy efficiency.
- **Chapter 6: Conclusion and Future Work** — This chapter summarizes the findings of the research. Here we discuss the implications of these findings and explore the limitations and challenges encountered. The chapter concludes with recommendations for future research, suggesting ways to further refine and dive deeper in the specifics of this research.

CHAPTER 2

Theoretical framework

This chapter explains foundational theory concepts needed to understand the project of this thesis. We start from the bare basics of machine learning and neural networks then build on top of it introducing concepts like transformers, large language models (LLMs) and the retrieval augmented generator (RAG).

2.1 Basics of Machine Learning

This section starts at the foundations of machine learning. Here we cover the beginnings of machine learning and concepts like types of machine learning algorithms and evaluation of them.

2.1.1 Introduction to Machine Learning

Machine learning (ML) is a subset of artificial intelligence (AI), it is a set of algorithms that have the ability to automatically learn to predict specific features from data without being explicitly programmed. The importance of machine learning is in its ability to process huge amounts of data, learn patterns, and make decisions based on that. This capability has led to its wide range of applications across various fields and many areas of research where data-driven decision-making is crucial.

Machine learning algorithms are broadly categorized into three types based on the nature of the way of learning and wanted output: supervised learning, unsupervised learning, and reinforcement learning.

- Supervised Learning: This type of ML involves training a model from labeled training data, which means that each training example is paired with an output label. The model makes predictions or decisions based on input data and is corrected when its predictions are incorrect. Supervised learning is used for problems where e have some historical data with correct outputs that the model can be then trained on. The foundational concepts of supervised learning were notably discussed in the work of Samuel (1959) [Sam59] in his studies on checkers playing programs, which is often cited as one of the early examples of machine learning in action. Supervised learning can further be divided into regression or classification, depending on the target feature being either continuous (for regression) or discrete (for classification).

- Unsupervised Learning: Unlike supervised learning, unsupervised learning deals with data that has no labels. The system tries to learn the patterns and the rules from the data without any concrete instructions on what to predict. Therefore the naming unsupervised. Unsupervised learning is typically used for clustering and association problems. Early work in unsupervised learning can be traced back to the self-organizing maps introduced by Kohonen in the 1980s [Koh82], providing a way to visualize high-dimensional data.

- Reinforcement Learning: This type of learning main idea is to incorporate iterative learning through using rewards. Reinforcement learning differs from the supervised learning in the way that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. Instead, the focus is on performance, which relies on the notion of agents that perceive and act within an environment, as described by Sutton and Barto (1998) [SB98], who have been pivotal in defining and advancing the field of reinforcement learning.

Each of these learning types has their own specific learning algorithms, addressing different problems and leveraging data in unique ways to learn, adapt, and provide solutions.

2.1.2 Evaluation Metrics

In the domain of machine learning, the evaluation of models is the key to understanding their performance and applicability in real-world scenarios. This evaluation is different depending on the type of machine learning. We introduce how evaluation differs from simpler to more complex machine learning approaches, relevant to this project. We will go over evaluation metrics for: classification, regression, natural language processing (NLP), and finally large language models (LLMs). The metrics used across these different ML families range from simple to complex, evolving to address the different demands of each specific task. Below, we present an overview of these metrics.

Classification Metrics

To evaluate the output of a classification algorithm, the simple goal is to visualize how much of the predictions are correct. **Accuracy** is the simplest and most intuitive metric, calculated as the ratio of correct predictions to the total number of predictions. It provides a the most intuitive evaluation of model performance but can be misleading in imbalanced datasets where the we have one class that has a significantly higher number of data points. To deal with this, precision and recall are used. **Precision** measures the ratio of correctly predicted positive observations to the total predicted positives, focusing on the model's accuracy in predicting positive instances. **Recall**, on the other hand, assesses the ratio of correctly predicted positive observations to the all observations in actual class, emphasizing the model's ability to capture all relevant instances. The **F1 score** harmonizes precision and recall into a single metric by calculating their harmonic mean, offering a balance between them and is particularly useful in cases of class imbalance. [VR79]

Regression Metrics

In regression tasks, the goal is to predict a continuous outcome, therefore the metrics need to describe how close the predicted number is to the expected one. Mean Absolute Error (MAE) measures the average magnitude of errors in a set of predictions. Mean Squared Error (MSE), on the other hand, squares the errors before averaging, penalizing larger errors more severely. [DS98] R^2 (R-squared) provides a measure of how well the observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model. [Kva85]

2.2 Neural Networks

2.2.1 Fundamentals of Neural Networks

Neural networks, a foundation of modern artificial intelligence development, are inspired by the biological neural networks that are the building blocks of our brains. Main part of these networks is the artificial neuron or node, which tries to mimic the functionality of biological neurons. An artificial neuron receives an input (similar to dendrites in biological neuron), processes it through a weighted sum followed by a non-linear activation function, and produces an output (similar to axon in biological neuron). This simple mechanism enables the neuron to perform basic computations and pass information through the network. We can see an example schema of a neuron in Figure 2.1

Neural networks can consist of many layers of these neurons arranged in a hierarchy. With one neuron making simple calculation, a significant number of them, can model complex functions. Usually, there are three types of layers: the input layer, the hidden layer, and the output layer. This layered structure allows neural networks to learn complex patterns and relationships within data, trying to mimic how the neurons work inside a human brain. Neural networks can learn different types of functions, from simple linear functions to complex non-linear relationships. This property is known as the universal approximation theorem [LL20], which essentially states that a neural network with at least one hidden layer of a sufficient number of neurons, and a non-linear activation function can approximate any continuous function to an arbitrary level of accuracy.

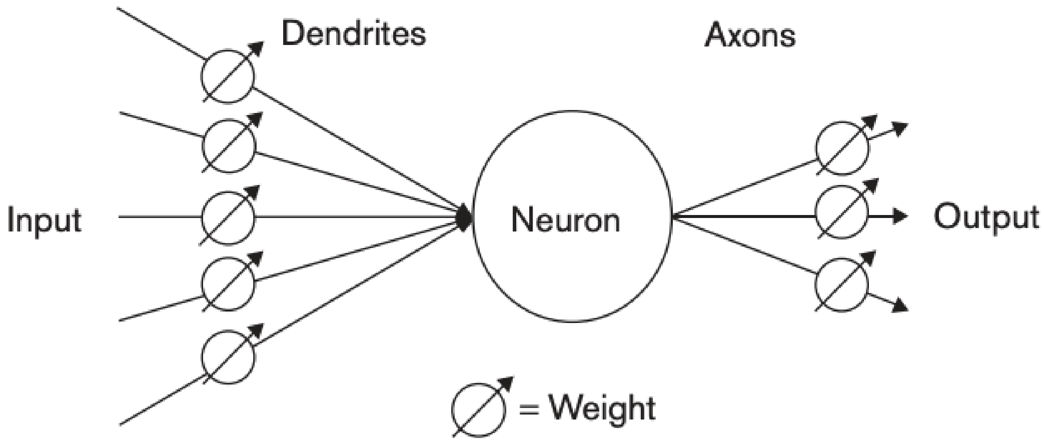

Figure 2.1: Schema of a single neuron [GB08]

The concept of neural networks and the inspiration drawn from biological processes were first introduced by Warren McCulloch and Walter Pitts in 1943[MP43]. This work tried to explain that networks of artificial neurons could potentially mimic brain functions, setting the stage for decades of research and development that have led to the sophisticated neural networks we see today in a myriad of applications, from image and speech recognition to playing complex games and driving autonomous vehicles.

2.2.2 Architecture of Neural Networks

The architecture of a simple neural network as already above mentioned has three primary types of layers: the input layer, hidden layer/s, and the output layer. This structure allows the network to transform input data into meaningful outputs through a series of computational steps, a process that is both sequential and hierarchical.

- Input Layer: The input layer serves as the entrance to the neural network. Each neuron in this layer maps to a feature in the input dataset, directly receiving the values of these features. For instance, in a neural network designed to recognize images, each neuron might represent the color value of a pixel in the image. The primary role of the input layer is to distribute the received data to the neurons in the next layer (first hidden layer).
- Hidden Layers: After the input layer, follow one or more hidden layers, which are the core computational parts of the neural network. Neurons in these layers are not exposed to the input or output, but they receive inputs from the input layer, perform calculations and then pass their results to the output layer. Each neuron in a hidden layer combines its inputs by computing a weighted sum and applying an activation function. The activation function introduces non-linearity into the

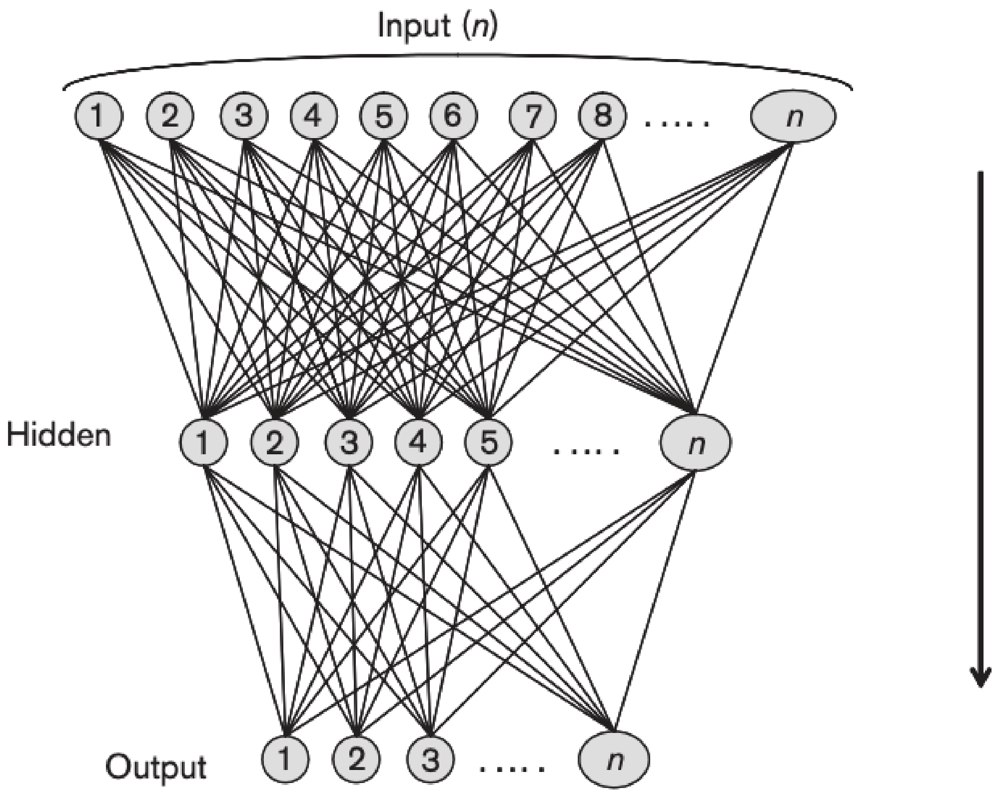

Figure 2.2: Schema of a simple neural network [GB08]

model, enabling the network to learn complex patterns and relationships in the data. Common choices for activation functions include the sigmoid, tanh, and ReLU functions.

- Output Layer: The output layer is the final layer of the neural network. It finalises the network's computations to produce the output. How the output layer looks (its neurons and activation function) depends on the specific task the network has. For regression tasks, the output layer may have a single neuron with no activation function (or a linear activation function) to predict a continuous value. For classification tasks, the output layer could have one neuron per class and uses a softmax activation function for multi-class classification, or a sigmoid activation function for binary classification, to produce probabilities that map to the likelihood of each class.

We can see the schema of a simple neural network in Figure 2.2, with its input, hidden and output layers.

2.2.3 Training Neural Networks

Training neural networks involves adjusting the weights of specific neurons to minimize the difference between the actual output and the predicted output for the given inputs. This process is done with two fundamental algorithms: backpropagation and gradient descent.

Backpropagation [RHW86] calculates the gradient, or the rate of loss decrease, for each weight in the network by employing the chain rule from calculus. The process is called "backpropagation" because it starts at the output layer and progresses backwards towards the input layer, determining the impact of each weight on the overall loss. This method pinpoints the necessary adjustments for each weight to reduce the loss.

Gradient Descent [Bot10], an optimization technique, works together with backpropagation. It adjusts the neural network's weights and biases to minimize the loss. By using the gradients calculated through backpropagation, gradient descent updates the weights by moving in the opposite direction of these gradients. The learning rate, an important parameter, sets the magnitude of each update. It is important to set it not too high nor too low, to optimize the training process.

The most straightforward form of gradient descent, known as **batch gradient descent**, calculates the gradient of the loss function for the entire training dataset for each update. However, this can be computationally expensive and slow for large datasets. To address this, **stochastic gradient descent (SGD)** [Bot10] updates the model based on the gradient of the loss function for a single sample at each iteration. While this introduces more noise into the training process, it can lead to faster convergence and can help to avoid local minima. **Mini-batch gradient descent** [LZCS14] strikes a balance between these two by updating the model based on a small, random subset of the data. This approach is often preferred in practice due to its computational efficiency and convergence properties.

2.2.4 Deep Neural Networks

A neural network is considered "deep" when it contains multiple hidden layers between the input and output layers. The depth of a neural network refers to the number of layers through which data passes in this transformation process. This depth allows the network to model complex relationships in the data by constructing a hierarchy of features or representations [HS06].

2.3 Natural Language Processing

In this section, we explore key concepts and models in Natural Language Processing (NLP). We start by introducing word embeddings, which are techniques that transform words into numerical vectors. This allows computers to understand and process language better. We then explore transformers and the attention mechanisms, advanced models that improve how machines interpret and generate language. We also touch upon how we can evaluate methods in NLP

2.3.1 Word Embeddings

Word embeddings are a family of techniques that try to represent words in a dense, continuous vector space. These representations capture semantic relationships among words based on their co-occurrence in large texts, having a goal that words with similar meanings have similar vector representations. This approach is a significant advancement over traditional bag-of-words models, which represent text as sparse vectors indicating the presence or absence of words, without capturing any semantic information. Two of the most well known word embedding models are Word2Vec and GloVe:

- Word2Vec [MCCD13] is a predictive model introduced by Mikolov et al inside Googles research lab. It generates word embeddings by either predicting a word from its context (Skip-Gram model) or predicting the context given a word (Continuous Bag of Words - CBOW model). The objective is to adjust the word vectors to maximize the likelihood of observing the words in their contexts in the training data. This process results in vectors where words that occur in similar contexts have similar embeddings, effectively capturing syntactic and semantic word relationships.

- GloVe (Global Vectors for Word Representation) [PSM14], developed by Pennington et al. at Stanford, is a count-based model that constructs a large matrix tallying how frequently pairs of words co-occur within a given "context" in the training data. It then factorizes this matrix to yield a lower-dimensional representation (the embeddings), where the distance between word vectors captures both their co-occurrence probability and their semantic similarity. Unlike Word2Vec, which relies on local context information, GloVe incorporates global statistics (the entire corpus) to obtain word embeddings.

Word embeddings have become a important component in many NLP applications, including text classification, sentiment analysis, translation, and question-answering systems. They offer several benefits for representing text data.

Dimensionality Reduction: Embeddings provide a dense representation, significantly reducing the dimensionality compared to sparse representations like one-hot encoding.

Semantic Similarity: Embeddings capture semantic similarities, enabling models to understand synonyms, analogies, and other linguistic patterns.

Transfer Learning: Pre-trained embeddings can be used across different tasks, allowing models to use knowledge learned from large wide datasets in other use cases.

2.3.2 Transformers and Attention Mechanism

The introduction of transformers has started a significant shift in the field of Natural Language Processing (NLP), setting new standards for a wide range of tasks, of which most common are language translation, text summarization, and question answering. Most important part of this breakthrough is the attention mechanism, which allows models to dynamically give "attention" to different parts of the input data. This enhances the ability to understand and generate language. Recurrent Neural Networks (RNNs) were previously the state of the art method for language tasks. They process data sequentially and are challenged by long-range dependencies. On the other hand transformers operate on the entire input data simultaneously. This architecture enables direct modeling of relationships between all parts of the input, irrespective of their positional distances from each other.

Introduced by Vaswani et al. in their paper "Attention is All You Need" [VSP+17] in 2017, the transformer model was introduced along with the attention mechanism, proposing a improvement to recurrence methods and instead using a network of self-attention layers to process data. This significantly different approach not only addresses the limitations of RNNs, such as difficulties with parallelization and handling long-distance dependencies but also significantly improves efficiency and performance on NLP tasks. Transformers have since become the foundation for a new generation of NLP models, including BERT [DCLT18] (Bidirectional Encoder Representations from Transformers) and GPT [RNSS18] (Generative Pre-trained Transformer), changing our approach to understand and interact with language through machines.

In Figure 2.3 we can see the architecture of the transformer model.

Self-Attention and Positional Encoding

In the architecture of transformers, self-attention and positional encoding play important roles in enabling the model to interpret and generate text by understanding the context and relationships in the text of the input data.

Self-Attention is a mechanism that allows each position in the input sequence to attend to all positions within the same sequence in order to compute a representation of the sequence [VSP+17]. This mechanism enables the model to dynamically weigh the significance of each word in a sentence relative to every other word, thereby capturing the nuances of language, such as syntax and semantics, more effectively than previous architectures. The self-attention mechanism can be mathematically represented as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

where Q, K, and V represent the query, key, and value matrices respectively, derived from the input. d_k is the dimension of the key, and the softmax function is applied to

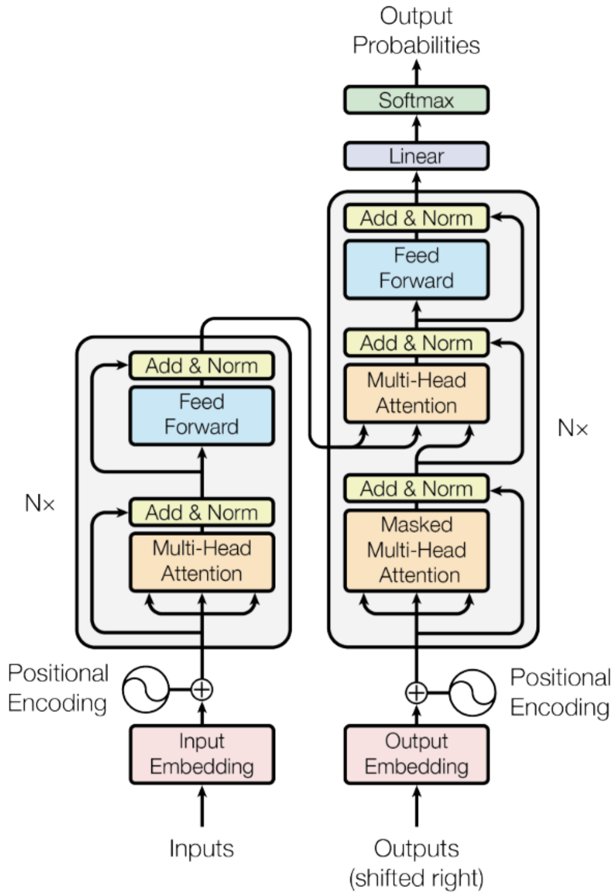

Figure 2.3: Schema of the transformer model architecture [VSP+17]

ensure the weights sum to 1. The output of this calculation is a weighted sum of the values, which incorporates information from the most relevant parts of the input sequence. [VSP+17]

Positional Encoding is used to give the model information about the order of the words in the sentence, as the self-attention mechanism does not by itself capture sequence order. Positional encodings are added to the input embeddings at the bottoms of the transformer model to provide information about the position of each word in the sequence [VSP+17]. The positional encoding vector for position p and dimension i in the sequence can be defined as follows:

$$PE_{(p,2i)} = \sin\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right) \tag{2.2}$$

$$PE_{(p,2i+1)} = \cos\left(\frac{p}{10000^{2i/d_{\text{model}}}}\right) \tag{2.3}$$

where p is the position, i is the dimension, and d_{model} is the dimension of the model's embeddings. These equations alternate between sine and cosine functions to allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{p+k} can be represented as a linear function of PE_p. [VSP+17]

Together, self-attention and positional encoding enable transformers with the ability to process text in a highly flexible and context-aware manner, significantly enhancing the model's understanding of language.

Transformer-based Models

The introduction of the transformer architecture has started a new era of models that have significantly advanced the state-of-the-art in Natural Language Processing (NLP). Among these, BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) stand out for their innovative use of transformers to achieve state of the art performance across a wide range of NLP tasks.

BERT, developed by Devlin et al., employs the transformer's encoder mechanism to process words in relation to all other words in a sentence, which allows for a more nuanced understanding of language context and word relationships. BERT's breakthrough was in its pre-training on a large corpus of text on two unsupervised tasks: masked language modeling and next sentence prediction. This pre-training enables the model to understand language patterns and structures, which is then fine-tuned on a smaller dataset for specific tasks, leading to significant improvements in tasks like question answering, sentiment analysis, and language inference. [DCLT18]

GPT, on the other hand, leverages the transformer's decoder for generative tasks. Introduced by Radford et al., GPT's architecture is distinctive for its use of unidirectional self-attention, where each token can only attend to previous tokens in the sequence. This architecture enables effective pre-training on a variety of texts, learning patterns, and nuances of the language, which can then be applied to generate coherent and contextually relevant text. GPT's successive versions have expanded upon this framework, increasing the model size and training data, thereby setting new benchmarks in tasks like text generation, translation, and summarization. [RNSS18]

These transformer-based models have not only demonstrated state of the art performance on NLP tasks but have also set the tone for the development of more advanced Large Language Models (LLMs). As we will discuss in the following sections on LLMs, models like GPT-3 [TBB20] and beyond continue to expand the possibilities of what can be achieved with machine learning in processing and generating human language, marking a significant evolution in the field of artificial intelligence.

Evaluations in NLP

Evaluation in Natural Language Processing (NLP) gets a bit more difficult, because there in not a specific number or a class that needs to be predicted, but human language. These evaluations try to incorporate both traditional and specialized metrics due to the complexity and variety of tasks. In tasks such as next word prediction, we use a measure called perplexity, which is a measure in NLP that captures the degree of 'uncertainty' a model has in predicting, using entropy [JMBB05] For tasks like translation and summarization, metrics such as BLEU assess the similarity between machine-generated text and a set of reference texts, focusing on the precision of n-grams. [PRWZ02]

2.4 Large Language Models (LLMs)

Large Language Models (LLMs) represent a new type of models in the field of natural language processing (NLP), built on the foundational architecture of transformers. These models are presented by their size, often consisting of neural networks with billions of parameters, and their ability to process and generate human-like text across a wide range of tasks and languages.

2.4.1 Definition of LLMs

The concept of a Large Language Model (LLM) is not tightly defined by a specific set of features or architecture but rather by its capacity to understand and generate human language at a scale on a complexity level previously unattainable. While earlier attempts at language models relied on different architectures like RNNs, the evolution of LLMs has been ignited by an increasing usage of the transformer architecture introduced by Vaswani et al. in their paper "Attention is All You Need" [VSP+17]. This architecture has become the backbone for modern LLMs, including the Generative Pre-trained Transformer (GPT) series and BERT, among others. Each iteration of these models has pushed the limits of what's possible, and constantly setting new state of the art performances. [CNW+24]

2.4.2 Architecture of LLMs

The architecture of Large Language Models (LLMs) has changed significantly with the introduction of transformer architecture, adapting to the needs of different NLP tasks. These variations are primarily manifested in how they implement attention mechanisms and the structuring of transformer blocks. A comprehensive analysis by Humza Naveed et al., in "A Comprehensive Overview of Large Language Models," [NKQ+23] goes in depth on these architectural differences, particularly focusing on the patterns of attention these models utilize and their implications on performance and applicability.

Encoder-Decoder Architecture: This dual-component is an architecture that was the first commonly used, where the input sequence is first processed by the encoder, which uses self-attention to understand the entire context as a whole. The encoder's output, an intermediate representation of the input, is then passed to the decoder. The decoder, leveraging attention, processes this representation to generate the output sequentially. This architecture is commonly used for tasks requiring a transformation from one form of data to another, such as for example language translation task, where the model needs to understand the context of the input before generating a corresponding output in another language.

Causal Decoder Architecture: In contrast to the Encoder-Decoder model, the causal (or autoregressive) decoder operates without an encoder, using only the decoder component to process the input and generate output. The attention mechanism in this architecture is made to be unidirectional, meaning that each predicted token can only depend on previously generated tokens. This architecture is commonly used for generative tasks where the sequence of the output matters, such as in text generation or conversational agents, ensuring that the model's output is coherent and follows a logical sequence.

Prefix Decoder Architecture: Also known as the non-causal decoder, this architecture introduces a more flexible approach to attention. Unlike the causal decoder, the prefix decoder allows for bidirectional attention, meaning that the model is not strictly limited to using past information but can also incorporate future context within certain bounds. This approach enables more complex understanding and generation tasks, where the relevance of information isn't strictly linear or dependent on sequence, allowing for a richer interpretation of text.

2.4.3 Model training

The process of developing Large Language Models (LLMs) is done by a series of pre-training objectives followed by fine-tuning stages, designed to pre-train the models to have a comprehensive understanding of language and then fine tune them for specific tasks. This approach is detailed in "A Comprehensive Overview of Large Language Models" by Humza Naveed et al., which provides a structured insight into the mechanisms underlying the training of LLMs. [NKQ+23]

1. **Pre-Training**: Initially, LLMs are pre-trained in a self-supervised manner on extensive text corpora. This foundational step leverages different architectures (e.g., encoder-decoder, decoder-only) and loss functions, aiming to predict subsequent tokens from given inputs.

2. **Fine-Tuning**: After pre-training, LLMs are fine-tuned to excel in specific tasks. This involves various strategies:

   - **Transfer Learning**: Leveraging the general capabilities developed during pre-training, LLMs undergo task-specific fine-tuning to enhance performance on particular tasks.
   - **Instruction-Tuning**: The model is further refined with data formatted as instructions accompanied by input-output pairs. This method aims at

improving the model's responsiveness to user queries and enhancing its zero-shot generalization capabilities.

- **Alignment-Tuning**: This stage focuses on aligning the model's outputs with human values by generating and correcting responses based on feedback. Here Reinforcement Learning with Human Feedback (RLHF) is utilized for alignment, comprising of two main components:
  - **Reward Modeling**: In this phase, a classifier is trained to evaluate model-generated responses against human preferences, aiming to distinguish between preferred and dispreferred outputs.
  - **Reinforcement Learning**: Coupled with the reward model, this phase employs methods like Proximal Policy Optimization (PPO) to iteratively align the model's outputs with human values until convergence.

These stages, including pre-training and fine-tuning, describe the process in the development of LLMs, ensuring they are not only powerful in language understanding and generation but also aligned with ethical standards and human expectations.

2.4.4 Applications and Limitations

LLMs have found applications in a wide variety of tasks, including but not limited to, text generation, translation, summarization, automated customer service... [KHM+23] Their ability to understand and generate human-like text has opened new opportunities for interaction between humans and machines, facilitating more natural and efficient interfaces.

However, LLMs do have their limitations. Their reliability on large datasets for training can embed biases present in those datasets, leading to ethical concerns. Additionally, their performance can degrade on tasks requiring domain-specific knowledge not covered in their training data or when handling very sensitive or private information, due to their generalizing nature. [KHM+23] It is in these contexts that Retrieval-Augmented Generation (RAG) [LPP+20] architectures come into play, offering a way to augment the capabilities of LLMs by combining them with external knowledge sources, thus addressing some of their inherent limitations. RAG will be introduced in the coming sections.

2.5 Retrieval Augmented Generation (RAG)

2.5.1 Concept of RAG

Retrieval-Augmented Generation (RAG) [LPP+20] architecture represent a new concept in the field of natural language processing. It tries to integrate the extensive knowledge within large pre-trained language models (LLMs) with the information retrieval capabilities of non-parametric memory systems. Unlike standalone LLMs that store knowledge purely within their parameters, RAG models include an explicit, external

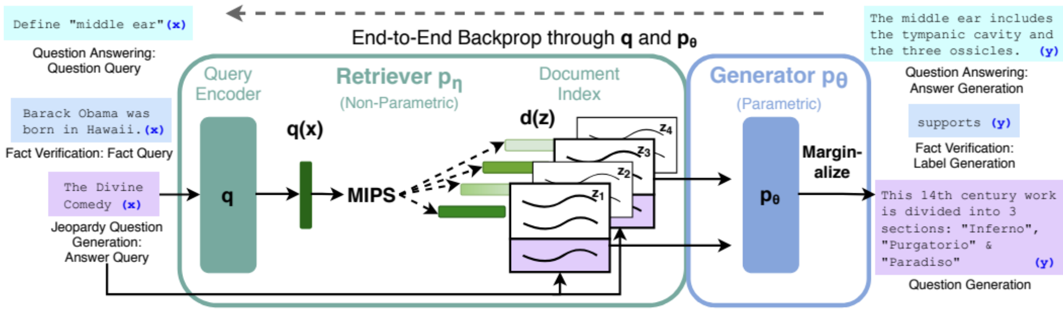

Figure 2.4: Schema of the RAG Model Architecture [LPP+20]

memory component, allowing them to access and utilize up-to-date information and provide references for their outputs.

The development of RAG models is motivated by the limitations of standalone LLMs, most importantly their restricted ability to access knowledge data points precisely and to update their stored information from time to time. This limitation becomes particularly obvious in knowledge-intensive tasks where we need accuracy and the up to date information. Standalone LLMs also have a problem providing insights into how they reached a certain answer. This often results in outputs that are maybe hallucinated or are difficult to trace back to reliable sources.

RAG models address these challenges by combining the generative power of pre-trained LLMs with the information retrieval capabilities of vector databases. This combination enables RAG models to do real-time information retrieval relevant to the input query, and therefore enriching the question answering process with relevant, up-to-date information. Furthermore, RAG models also provide a solution to the issue of information freshness, as the vector database can be updated without retraining of the LLM at all. [LPP+20]

2.5.2 Architecture of RAG Models

The architecture of Retrieval-Augmented Generation (RAG) models is a composite of model and database components designed to leverage both parametric and non-parametric knowledge sources for language generation. Figure 2.4 shows the architecture of a RAG model, which we describe in detail below.

Query Encoder q

Query encoder is a neural network responsible for converting input queries into a vector space representation or in other words it is the embedding layer. Inputs, shown as x, are typically prompts or questions that the model needs to process. The encoder transforms these inputs into dense vector representations, $q(x)$, which are then used to retrieve relevant documents from a knowledge source.

Retriever p_η

The retriever, symbolized by p_η, is a system that retreives the relevant information to the query. Using techniques of vector search such as Maximum Inner Product Search (MIPS), it searches through a document index to identify and fetch documents most relevant to the query. These documents, encoded as dense vectors $d(z)$, serve as the non-parametric memory that RAG models utilize to augment generation with external knowledge.

Document Index $d(z)$

The document index is a collection of information sources, typically encoded into dense vector representations for efficient retrieval. It forms the non-parametric information storage from which the retriever retrieves the information. This is usually implemented as a vector database.

Generator p_θ

The generator p_θ, a parametric module usually implemented by a transformer-based LLM model, is here for the text generation process. It uses both the original input query and the retrieved documents to produce the final output. The output is the combination of the pretrained understanding of language from LLMs and the contextually relevant and up to date information retrieved by p_η.

2.5.3 Challenges

Implementing Retrieval-Augmented Generation (RAG) models necessitates understanding a complex landscape of challenges. The integration of external information retrieval systems with generative language models presents several challenges that researchers and practitioners must have in mind when designing the system.

Data Quality and Biases The accuracy of the information retrieved by RAG models is relying on the quality of the underlying data sources. Ensuring the quality and freshness of the external databases is critical, as biases or inaccuracies in the indexed documents will propagate through the generative outputs of the model. [YG23]

Indexing and Retrieval Efficiency The efficiency of the retrieval component is a critical component for the quality of RAG models. The document indexing must be optimized for fast retrieval to maintain performance, especially when dealing with large-scale databases. Additionally, the retrieval process must balance precision to ensure the retrieved documents are relevant [YG23]

Computational Resources RAG models require additional computational power compared to just usage of LLMs. There is not a lot of relevant research in this category, to conclude how much is the increase in compute usage, so this is one of the goals of our research.

Dynamic Knowledge Updating While RAG models allow for the non-parametric memory to be updated or swapped, ensuring that the model remains up-to-date with current information is a challenge. There must be mechanisms in place to regularly refresh the knowledge database without introducing downtime or performance degradation.

CHAPTER 3

Related work

In this chapter, we will go through the current research related to Retrieval-Augmented Generation (RAG) models and their usage in natural language processing tasks. The primary focus of this review is to understand how RAG models have been used to improve the performance of Large Language Models (LLMs) on domain-specific questions. Moreover, since our goal is to examine the impact of RAG models on response times and the computational resources required, we check the current state of the research for results with these questions. This examination will not only inform and support our implementation of a domain-specific RAG model, but will also shed light on potential areas for improvement, particularly in terms of response accuracy, speed, and computational cost compared to standalone LLMs.

3.1 RAG systems

3.1.1 Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks [LPP+20]

This paper presents a novel approach that tries to combine the generative ability of LLMs with the exactness of external knowledge bases. This is the first paper that introduced the RAG approach. The strength of RAG lies in its ability to generate answers by effectively retrieving exact information sources, allowing it to base its answers on actual evidence and documentation.

RAG models can generate correct answers even if the exact answer is not present in any retrieved document, with a notable accuracy improvement over just data extraction and search methods. This is especially advantageous for abstract question answering, where RAG models outperform BART [LLG+20] on metrics like BLEU [PRWZ02]. Furthermore, RAG models are credited with producing more factually correct and specific text, with a marked increase in diversity of generation compared to BART.

In tasks like Jeopardy question generation, RAG's ability to combine information from multiple documents leads to superior performance. Jeopardy question generation assesses the ability of models to create complex, factual questions where the answer is a known entity, such as generating the clue for "The World Cup" based on its historical facts. This task is particularly challenging as it requires the model to reverse the usual question-answering approach and produce specific and accurate prompts that would lead to the right answer. Human evaluators have found RAG-generated text to be more factual and specific than BART.

The paper also demonstrates that RAG models perform really well in fact verification tasks, achieving scores close to state-of-the-art models that use quite more complex systems and training approaches. They do this by doing the FEVER [TVCM18] task that involves checking if a claim can be supported, refuted, or is unverifiable based on external information sources, requiring both retrieval and reasoning. RAG models solve this by classifying claims without needing retrieval supervision, focusing on their capability for classification tasks in addition to generation. Additionally, the ability to "hot-swap" the retrieval index allows for easy updating of the models' knowledge base, showcasing an efficient way to maintain relevance with current information.

An area of future exploration suggested by the authors is the joint pre-training of the retriever and generator components from scratch, possibly using a denoising objective. This proposal aims to further improve the performance of RAG models. The research also hints at the potential for applying RAG to a broader range of NLP tasks, given its flexibility and robust performance. It also gives a hint of need of more exploration how the approach performs in different domains of knowledge.

The paper concludes by reflecting on the broader impact of RAG models. They are seen as a significant step forward in producing less biased and more factually grounded outputs, offering greater interpretability of AI-generated content. However, the paper also acknowledges the risks associated with any knowledge source, including Wikipedia, which can never be entirely factual or free from bias. Potential misuse of RAG models for generating fake content, as well as their implications for job automation, are concerns that need to be addressed alongside their development. [LPP+20]

3.1.2 Retrieval-Augmented Generation for Large Language Models: A Survey [YG23]

This paper provides an in-depth exploration of the development and diversity of Retrieval-Augmented Generation (RAG) approaches, particularly after the advent of large language models (LLMs) like GPT-3 [TBB20] and GPT-4. Figure 3.1 outlines the evolution of RAG research through four main phases, starting from its inception in 2020 with the focus on enhancing pre-training models (PTMs) alongside the emergence of the Transformer [VSP+17] architecture. It highlights a period of minimal progress followed by a significant leap forward with the introduction of chatGPT, leading to a focus on leveraging LLMs for improved controllability and addressing evolving needs, primarily through inference

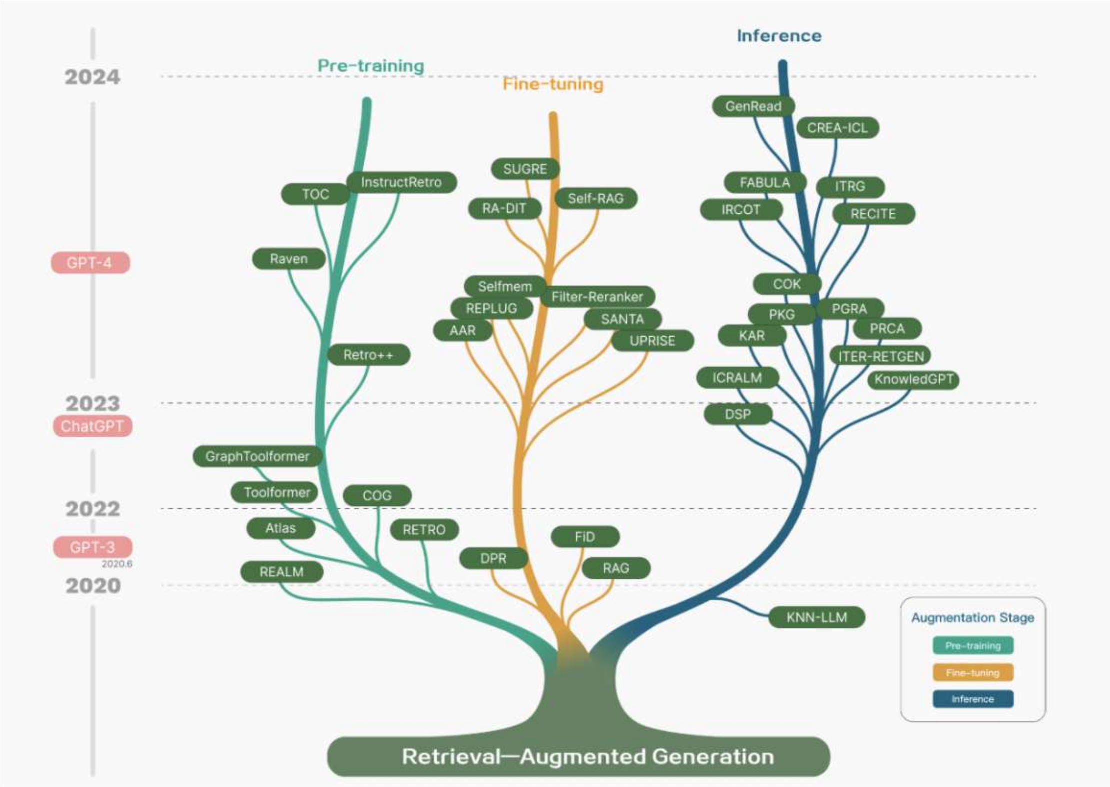

Figure 3.1: Technology tree of RAG research development featuring representative works [YG23]

and fine-tuning strategies. The latest phase sees a hybrid approach that marries the strengths of RAG with fine-tuning, while still refining pre-training methods.

The paper underscores the rapid growth in RAG research and the challenge of combining these advancements into a single framework. It aim is to clarify the entire RAG process, detailing technical principles, developmental history, methodologies, evaluation strategies, and future research directions. By differentiating the components of "Retrieval," "Generator," and "Augmentation," it seeks to offer a comprehensive overview and analysis of different technologies used for each of the components. Additionally, it proposes a robust evaluation framework for RAG and explores future enhancements, multi-modal applications, and the development of the entire RAG ecosystem.

The "RAG Framework" section outlines the evolution of RAG research, spliting it into a few different categories: Naive, Advanced, and Modular RAG. Naive RAG, one of the first approaches characterized by simple indexing, retrieval, and generation processes, was facing difficulties in retrieval accuracy, generation quality, and the integration of retrieved information. It is also characterized as a “Retrieve-Read” framework [MGH+23]. Advanced RAG was introduces as a response, refining indexing with more advanced segmentation of data and metadata use. It also enhanced retrieval through pre-retrieval

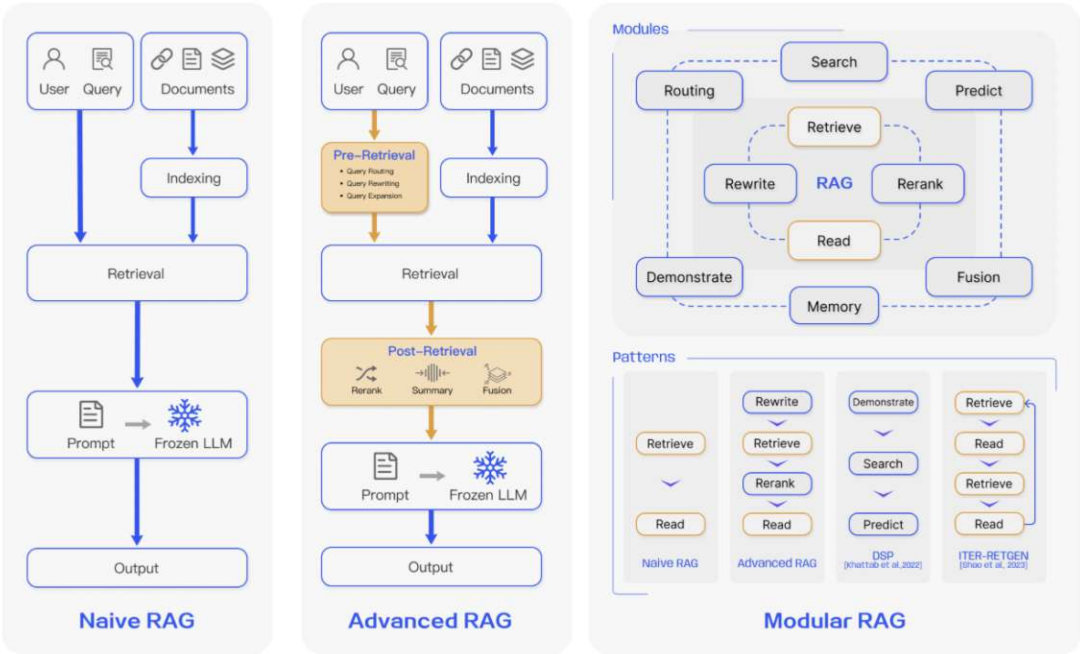

Figure 3.2: Comparison between the three paradigms of RAG [YG23]

optimizations and dynamic embedding models. It focuses on improving the relevance and efficiency of the information retrieval stage to better support the generation task. Modular RAG, on the other hand, offers a different framework by incorporating different functional modules like search and memory modules, allowing for customized improvements and a different approach for different tasks. This findings are summarized in Figure 3.2 from their paper.

All in all, this paper presents a comprehensive overview on the different methods and approaches of Retrieval-Augmented Generation (RAG), showing how it merges the general and wide knowledge of LLMs with specifics and precision of external data for improved performance on knowledge-intensive tasks. Future improvements aim towards extending the context window size, enhancing model robustness, and exploring hybrid approaches that effectively combine RAG with fine-tuning. Moreover, expanding the application scope of RAG into multimodal domains, such as image, audio, video, and code, presents new opportunities for RAG technology. These advancements suggest RAG's significant potential for practical applications.

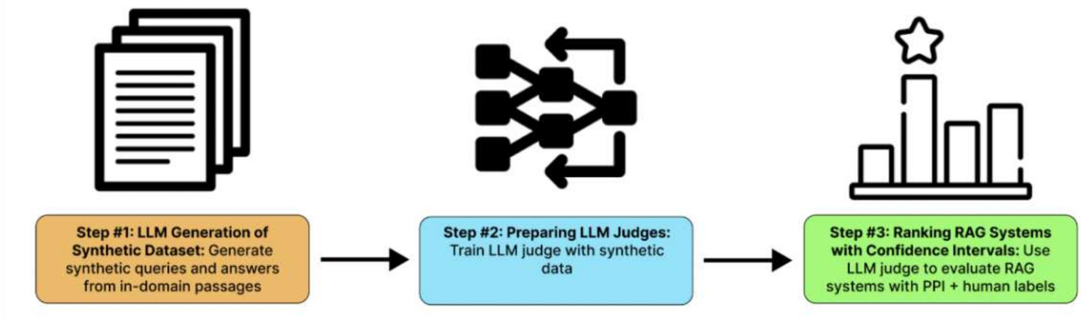

Figure 3.3: Overview of ARES [JSF23]

3.2 Evaluation of RAG systems

3.2.1 ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems [JSF23]

The ARES paper presents an Automated Evaluation System designed to evaluate Retrieval-Augmented Generation (RAG) systems, presenting a cost-effective and relatively precise alternative to traditional, labor-intensive evaluation methods done by humans. ARES does the full system evaluation by generating specific evaluations for each component of a RAG pipeline, significantly enhancing precision and accuracy over existing methods. Utilizing model-based evaluation and a minimal set of inputs, ARES quickly assesses RAG systems on context relevance, answer faithfulness, and answer relevance, enabling effective tuning and comparison of RAG configurations without the need for extensive human annotations.

ARES framework can be summarized in three distinct steps, we can see from Figure 3.3

- Synthetic Dataset Generation: ARES start by creating synthetic queries and answers from a corpus of text using LLMs. This dataset, comprising both positive and negative examples of query-passage-answer triples, is important for training LLM judges to assess various aspects of RAG system performance.

- Training LLM Judges: Using the synthetic dataset, ARES fine-tunes lightweight LLM judges on three key metrics: context relevance, answer faithfulness, and answer relevance. These judges classify the query-passage-answer triples, enabling a in depth evaluation of RAG systems performance.

- Ranking RAG Systems with PPI: The final stage involves using the trained LLM judges to score RAG systems, using Prediction-Powered Inference (PPI) and a human preference validation set to estimate confidence intervals for the scores. This process allows ARES to provide precise evaluations of RAG systems, highlighting their capacity to generate contextually relevant and accurate responses.

The ARES paper evaluates the effectiveness of the ARES framework against the current popular RAG evaluation system, RAGAS, and a GPT-3.5 model prompted with few-shot examples. The findings demonstrate ARES's superior accuracy in ranking RAG systems across multiple datasets. ARES, utilizing a DeBERTa-v3-Large model fine-tuned with synthetically generated data and augmented by Prediction-Powered Inference (PPI) with a 300-datapoint human preference validation set, outperforms both RAGAS and the few-shot GPT-3.5 judge in metrics of context relevance and answer relevance. Specifically, ARES achieves higher accuracy scores—59.3 percentage points above RAGAS for context relevance and 14.4 percentage points for answer relevance—indicating a more precise system for automatically evaluating RAG configurations. This precision allows ARES to guide the development of RAG systems more effectively, making it a valuable tool for optimizing RAG configurations with minimal human annotation.

3.3 Gap in current research

Current research in the Retrieval-Augmented Generation (RAG) field primarily explores the integration and performance of RAG models, emphasizing general usability and performance rather than domain-specific nuances. This generalized focus leaves a gap in understanding how RAG models perform in specialized domains, where the quality and specificity of responses can significantly impact their utility. My project aims to dive into this underexplored area by specifically assessing RAG models in a specialized domain (Google cloud platform [goo]), contrasting their performance with standalone Large Language Models (LLMs).

Specialized Domain Application: Most of the existing RAG research does not investigate the application of RAG models in highly specialized domains in depth, where domain-specific terminology and context play a crucial role in the accuracy and relevance of responses. My research will fill this gap by evaluating RAG and traditional LLMs in a specialized domain, involving domain experts in the evaluation process to ensure the accuracy and quality of model responses.

Response Time and Computational Resources: Another area less covered in current research is in the are of response times and computational efficiency in RAG models. While RAG models are known for their ability to generate more accurate and contextually relevant responses, there is limited empirical evidence on how they compare with standalone LLMs in terms of response time and computational load. My project seeks to provide a comprehensive comparison of RAG and standalone LLMs investigating these points, comparing how much is the increase in response time and computational load.

CHAPTER 4

Methodology

In this chapter, we explain the methodology used in our comparative analysis of Retrieval Augmented Generators (RAG) and standalone Large Language Models (LLMs).We start with explaining the process of data gathering and processing, focusing on how Google Cloud documentation serves as the domain knowledge base for the RAG system. Additionally, the setup of the RAG architecture is described in depth, covering components such as our generator - the LLM, vector database, and vector search mechanisms. Integration of all of these components is achieved using the LangChain framework, which combines retrieval and generation capabilities. The evaluation methods used to measure answer accuracy, response time, and compute consumption are also explained, ensuring a clear understanding of how the performance of the RAG compares to standalone LLMs.

4.1 Data

4.1.1 Google Cloud Platform

Google Cloud Platform (GCP) [goo] is a comprehensive set of cloud computing services developed by Google, that enables developers and organizations to build and operate applications and services in a highly scalable and secure cloud environment. GCP offers a wide range of services, including infrastructure provisioning, compute services, machine learning tools, data management solutions, security features, and networking options. The platform's complexity and the specific knowledge required to effectively use each service make it an ideal use case for a Retrieval Augmented Generator (RAG). The details and complexity of GCP's services bring up the need for detailed and up-to-date documentation, which serves as the knowledge base for our RAG. By leveraging this documentation, our RAG can provide precise and secure answers, pulling directly from the most relevant information found in these resources.

The structure of Google Cloud documentation is designed to support a wide range of users, from beginners to advanced users, with detailed guides, tutorials, quickstarts, and reference materials. These documents are organized around specific products and services, providing in depth explanations that include configuration steps, best practices, and troubleshooting tips. Such a structured and comprehensive dataset is important for our RAG, enabling it to understand and generate responses that are not only accurate but also contextually relevant to the specific queries stated by users.

4.1.2 Data Gathering

In our study, the data gathering phase involved scraping the Google Cloud documentation using Python, specifically with the aid of the Beautiful Soup [Ric07] library. Beautiful Soup is a Python library designed to simplify the process of parsing HTML and XML documents. It provides tools for navigating, searching, and modifying the parse tree, making it ideal for web scraping because it can extract data from HTML, which is essential for our purposes.

The scraping process was done as follows:

- **Initial Link Collection**: We began by collecting all the primary service links from the main page of the Google Cloud documentation. This page serves as the gateway to more detailed documents related to each service offered on the Google Cloud Platform.
- **Recursive Link Gathering**: From each primary service link, we employed a recursive scraping approach. This involved navigating to every linked page from a service's main page, gathering links to further documentation. This recursion was controlled by depth parameters to ensure comprehensive coverage without redundant deep dives into linked pages.
- **Content Extraction**: For every documentation page we gathered through our recursive link gathering, the main content of the page was extracted. This typically included detailed descriptions, usage examples, and configuration instructions.
- **Data Cleaning and Storage**: Extracted content was then cleaned to remove any HTML tags, scripts, and irrelevant information, ensuring that only meaningful text was retained. The clean text was then stored for further processing.

This structured approach ensured that we systematically captured as much of the documentation as possible, covering the broad and complex services of Google Cloud. By focusing on the main content sections of each page, we were able to compile a comprehensive dataset that accurately reflects the scope and depth of information typically wanted by users of Google Cloud services. The algorithmic process can be seen in Algorithm 4.1.

Algorithm 4.1: Recursive Scraping of Google Cloud Documentation

```
Input: Base URL of Google Cloud Documentation
Output: Extracted text files from documentation pages
Collect initial set of service links from the main documentation page;
for each service link do
    RecursiveFetch(service link, current depth=1, maximum depth);
end
Function RecursiveFetch(URL, CurrentDepth, MaxDepth):
    if CurrentDepth > MaxDepth then
        return;
    end
    Fetch and parse page at URL;
    Extract and filter document links matching specific patterns;
    for each valid link do
        Recursive call to RecursiveFetch(link, CurrentDepth+1, MaxDepth);
    end
Extract and clean text from each documentation page;
Save the extracted content;
```

In our data gathering process, understanding the layout of the Google Cloud documentation pages was crucial. As illustrated in Figure 4.1, a typical Google Cloud documentation page is structured with a navigation menu on the left side and the main content body in the center. The navigation menu contains links to other relevant documentation pages, which were used for our recursive scraping method to gather as much of the possible links of the documentation. The main content body, displayed in the middle of the page, is the detailed documentation. This is the part of the page that we targeted for scraping. The content here is structured in textual format, often containing code snippets, diagrams, and tables, which provides a rich source of information. For our purposes, to simplify the data gathering process, this main body was extracted as plain text, ensuring that we capture the essential information needed for RAG.

4.1.3 Data Processing

Chunking

In the data processing stage, chunking is a crucial step to prepare the documentation content for use in the RAG system. Chunking involves splitting the larger text documents into smaller, more manageable pieces or "chunks." This is necessary because LLMs often have limitations on the length of text they can process at once. Keeping documents of similar and smaller lengths ensures that the data fed into the LLM is both manageable and retains meaningful context.

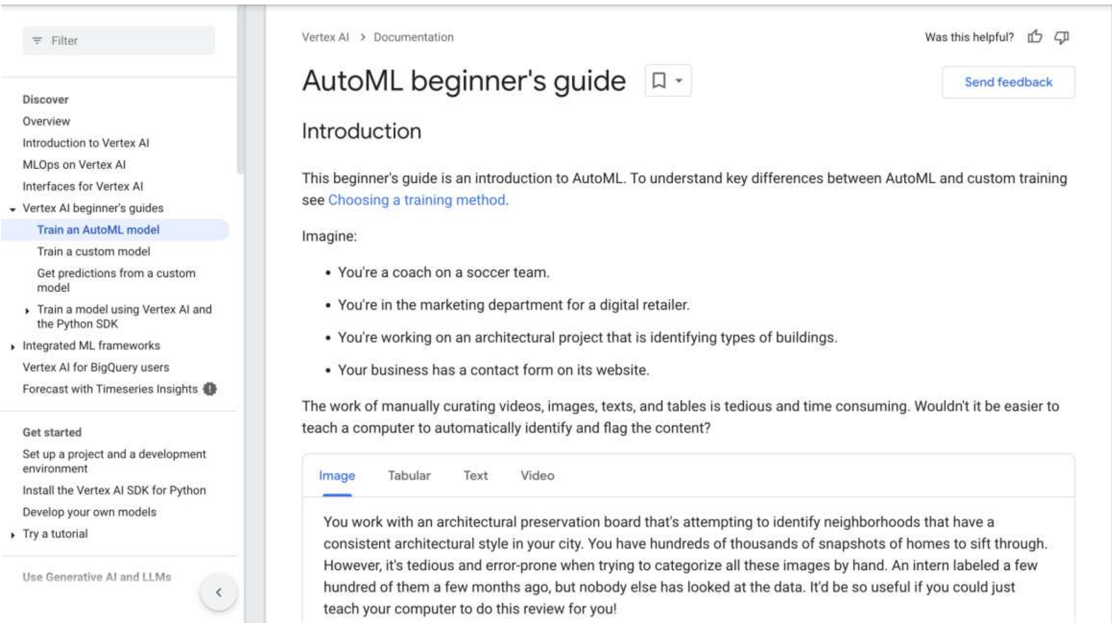

Figure 4.1: Example of Google Cloud documentation page [goo]

The logic behind our chunking approach is to maintain semantic coherence within the chunks as much as possible. To achieve this, we use a **hierarchical splitting strategy** based on a list of delimiters. The process begins by attempting to split the text into large segments using bigger delimiters such as double line breaks (\n\n). If chunks still are larger than the desired size, the splitter then moves to smaller delimiters like single line breaks (\n), spaces ( ), and eventually no delimiter, ensuring that even the longest paragraphs are actually broken down to a max chunk size limit. The size of each chunk is described by the number of characters it consists of. We set a maximum character limit per chunk to ensure consistency across the data set and prevent the model from being overwhelmed by too much information at once. Additionally, to ensure that no critical information is lost or context is misunderstood due to the splitting, we incorporate an overlap between consecutive chunks. This overlap means that each new chunk starts not only after the previous chunk ends but extends back a bit to include some of the characters from the end of the previous chunk.

Embeddings

Embeddings are an important aspect of our methodology, transforming text into a vector space to enable semantic search on the text itself. We employ the *textembedding-gecko* [LDR+24] model, a embedding model developed by Google.

The Gecko embedding model performs the retrieval by introducing a unique approach: it distills knowledge from large language models (LLMs) into a retriever. This process involves generating diverse, synthetic paired data initially using an LLM. The data undergoes a further refinement step where positive and hard negative passages are relabeled after candidate retrieval. Remarkably, Gecko maintains a high level of effectiveness even with a compact structure. For instance, a Gecko model with 256 embedding dimensions outperforms models with larger sizes or higher dimensional embeddings, as demonstrated in the Massive Text Embedding Benchmark (MTEB) [LDR+24].

After embedding the Google Cloud documentation with the Gecko embedding model, the generated embeddings are stored with their metadata. This metadata includes the document name and the source link of the actual documentation page. Storing embeddings with detailed metadata ensures that each embedding is aligned with its textual counterpart, enabling utilization in retrieval tasks.

4.1.4 Data Storage

For the purpose of storing our dataset, we have chosen Google Cloud Storage, which provides reliable and scalable cloud storage solutions. This service is used as a file storage on cloud, suited for handling large volumes of data, which makes it a good options to to store chunked documents and embeddings effectively. Chunks and embeddings are stored in two separate buckets in Google Cloud Storage. The organization is straight forward, each service's documentation is sorted under a folder named after the service itself.

The naming of the documents and embeddings follows a standardized format based on the original documentation URLs. The structure typically includes 'https://cloud.google.com/service-name/docs/document-name', where 'service-name' is used as the folder name and 'document-name' serves as the filename. This systematic approach helps streamline the management and retrieval of data, ensuring that each item is easily traceable back to its source.

The whole process of data processing and storage is described under Algorithm 4.2

4.2 Implementation of the RAG Architecture

The Retrieval Augmented Generator (RAG) architecture we implemented is a hybrid model that improves the performance of a standalone Large Language Model (LLM) by integrating external knowledge sources. At its core, the RAG architecture uses a dynamic retrieval of information based on user queries. It operates by retrieving the most relevant documents that are relevant to the query. Afterwards it feeds these relevant documents into the generator - the LLM. The LLM processes this information to generate a response that is both contextually relevant and informed by the external data. This approach allows the model to provide answers that are not just based on its pre-trained knowledge, but also enriched by specific, up-to-date information contained in the retrieved documents.

The following sections explain each component of the RAG architecture and how they come together to form the complete system. This includes a description of the base Large

Algorithm 4.2: Data Processing and Storage

```
Input: Raw text from Google Cloud Documentation pages
Output: Processed data stored in Google Cloud Storage
foreach document text do
    chunks ← ChunkText(document text);
    foreach chunk in chunks do
        embedding ← GenerateEmbedding(chunk);
        metadata ← extract metadata from document URL;
        StoreData(embedding, metadata);
    end
end
Function ChunkText(Text):
    Split Text into smaller pieces using delimiters [\n\n, \n, " ", ""];
    Maintain semantic coherence by controlling chunk size and overlap;
    return chunks;
Function GenerateEmbedding(Chunk):
    Convert Chunk into an embedding using the textembedding-gecko model;
    return embedding;
Function StoreData(Embedding, Metadata):
    Determine service and document names from Metadata;
    Save Embedding with Metadata in designated folders;
```

Language Model, the construction and utilization of the vector database for document retrieval, the search algorithms used to identify relevant documents, and the coordination of these components using the LangChain [lan] library for Python. Additionally, we discuss the frontend application development and deployment processes, detailing the use of technologies such as ChainLit [cha] for frontend and Cloud Run for deployment.

4.2.1 Large Language Model (LLM)

The Large Language Model serves as the generator component of the Retrieval Augmented Generator (RAG) architecture. Its task is to generate responses once the relevant documents have been retrieved. As outlined in section 2.5, LLMs leverage huge amounts of pre-training data to understand and predict language patterns (see section 2.5 for the theoretical foundation of large language models).

In our implementation, the role of the LLM is to use the information from the documents provided by the retrieval component to produce accurate and contextually relevant answers. We tested multiple models to identify the optimal one for our RAG system. The candidates included Google PaLM [CND+22], Claude 3 [Ant23], and the family of LLama models [TMS+23] the 7B, 13B, and 70B variants.

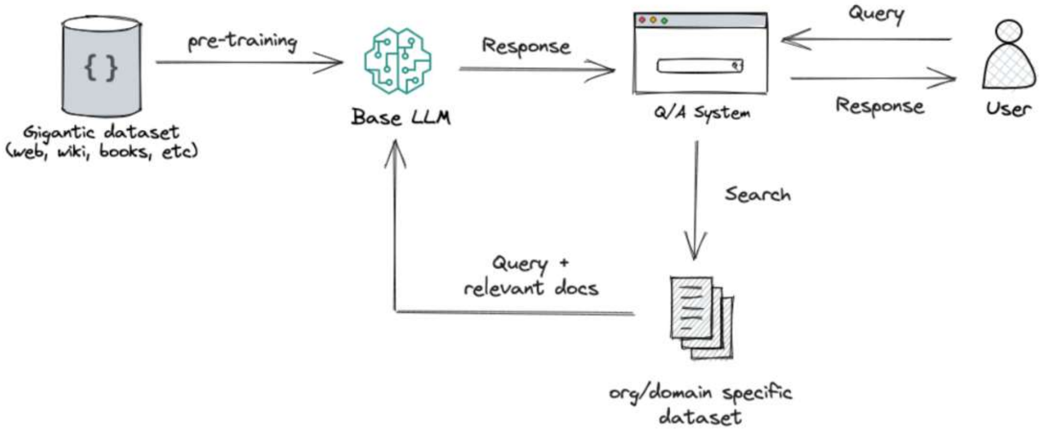

Figure 4.2: Simplified schema of the RAG Architecture

We selected LLama 70B for our final implementation, primarily because it was the best performing open-source model available at the time. While PaLM and Claude offered better performance, their architectures and specifics were not fully disclosed, which led to the decision of using an open-source model where the system's inner workings were transparent and well-documented.

For deployment of the model, we utilized Vertex AI, a managed service offered by Google Cloud, which streamlines the deployment of machine learning models (detailed information about Vertex AI deployment can be found at https://cloud.google.com/vertex-ai/docs/general/deployment). We chose to deploy the LLama 70B model on a 'g2-standard-96' machine type, designed to handle the computational requirements of running large-sized LLMs.

This particular machine type is equipped with 96 vCPUs, where 'vCPU' stands for virtual CPU. These virtual CPUs are slices of physical CPUs on a server, allowing for multitasking and concurrent processing of several tasks. Operating at a clock speed of 2.2 GHz, they provide the computational power needed for the model's data processing tasks. The machine's 384 GB of RAM is crucial for handling the extensive in-memory operations typical in LLM inference, because it loads the whole model in memory. Additionally, the deployment is accelerated by 8 NVIDIA L4 Tensor Core GPUs [NVI23]. These GPUs are engineered for accelerating machine learning workloads. They excel at parallel processing, which is a critical capability for performing the matrix operations that deep learning models run. By employing these powerful GPUs, we can significantly speed up the calculations required by the LLama 70B model, ensuring better response times.

4.2.2 Vector Database

The Vector Database within our RAG architecture is used for storing and retrieving the embeddings that represent the chunked documentation text. This database concept is important for the retrieval of relevant documents for user queries. It is implemented using the Matching engine service from Google cloud.

To build the index for our Vector Database, we used the Alphabetic Huffman Tree (AH-Tree) [YGL+13], a data structure well-suited for indexing datasets with skewed access frequencies. Such characteristics are typical for web-based services dealing with unevenly distributed query patterns, which is often the case with online documentation access. By leveraging the AH-Tree algorithm, we constructed a k-ary index that optimizes both space and time complexity, achieving efficient query processing.

This index was deployed as an endpoint on Google Cloud using the Vertex AI Search service, hosted on an ‘e2-standard-2‘ machine type. This machine, featuring 2 vCPUs at a clock speed of 2.25 GHz and 8 GB of RAM, was selected for its balance of performance and cost-efficiency. While less powerful than the machines used for deploying the LLM, it is well-suited for the less intensive workload of vector lookup and retrieval.

4.2.3 Vector Search

In our RAG architecture, the retrieval of similar documents is conducted using the ScaNN [GSL+20] algorithm, which is a built-in feature of Vertex AI's search capabilities. ScaNN, short for Scalable Nearest Neighbors, is an optimized method for conducting vector similarity searches efficiently at scale, particularly suitable for our application where rapid matching of query vectors to document vectors is very important. ScaNN optimizes for both search space pruning and quantization for Maximum Inner Product Search (MIPS), and it also supports a variety of distance functions, including the Euclidean distance. One of the key advantages of ScaNN is its configurability, which allows it to adapt to datasets of varying sizes and distributions. This flexibility, combined with its robust performance on large datasets, makes it an excellent choice for the vector search component in our system.

4.2.4 Coordination of RAG Components via LangChain

LangChain [lan] is a framework developed for orchestrating different language tasks to perform complex operations, such as question answering (QA) with document retrieval. In the context of our RAG system, LangChain is orchestrating the interaction between the Large Language Model (LLM) and the Vector Database, forming a cohesive workflow that handles QA processes.

The orchestration works as follows: When a user submits a query, LangChain's RetrievalQA chain starts a search using the Vector Database. The vector database identifies and retrieves documents that are semantically related to the query. These documents are then compiled and passed to the LLM along with the query itself. The LLM utilizes this

```
# Prompt Template instance
template = """
You are helping customers that use google cloud.
Provide the most specific and useful answer that you can make.
Reference the resources provided to you as much as you can.

Question: {question}

References:
=============
{context}
=============

Question: {question}

Helpful Answer, format so that it is readable, do not provide links:
"""
```

Figure 4.3: Python Code for Prompt Template

augmented input—the original query plus context from relevant documents—to generate a response. This is all formatted by a premade template in Figure 4.3, that helps the guide the LLM in what way to respond and use the retrieved documents.

The system is designed such that the user only needs to provide their question; the underlying LangChain-managed components handle the rest, delivering a text response that is backed by relevant source documents.

The RetrievalQA chain, a component of LangChain, is configured with the number of documents to be retrieved and a threshold for search distance, ensuring the relevance of retrieved documents. By doing so, LangChain helps in constructing an efficient and robust RAG system. This leads to an AI assistant that can provide users with informative and reliable answers based on a database of documents.

4.2.5 Frontend Application

The frontend application for our RAG system was developed using Chainlit [cha], an open-source Python framework based on Streamlit. This framework enables quick and easy development of conversational AI applications. Chainlit simplifies the integration of complex LLM functionalities into user-friendly interfaces, making advanced language models like ours accessible to a broader audience outside the typical programming environment.

Chainlit is particularly well-suited for applications similar to ChatGPT [TBB20], allowing customization with specific business logic and data integrations. Additionally, Chainlit's

Algorithm 4.3: Coordination of RAG Components via LangChain

```
Input: User Query
Output: Response grounded in relevant documents
retrievedDocuments ← SemanticSearch(User Query);
context ← CombineContextWithQuery(retrievedDocuments, User Query);
response ← GenerateResponse(context);
return response
Function SemanticSearch(Query):
    Use Vector Database to find documents semantically related to Query;
    return relevant documents;
Function CombineContextWithQuery(Documents, Query):
    Prepare prompt including Query and context from Documents;
    return combined context;
Function GenerateResponse(Context):
    Use LLM to generate a response based on the combined Context;
    return generated response;
```

ability to maintain data persistence and visualize multi-step reasoning helps in refining the model's outputs based on user interactions.

4.2.6 Deployment

For deploying our frontend application, we used Google Cloud Run, a fully managed platform that enables containerized applications to run in a scalable and serverless environment. Cloud Run is particularly suitable for our needs due to its ability to automatically scale up or down based on traffic, ensuring cost-efficiency while maintaining performance during varying load conditions. Its integration with container technologies also simplifies deployment processes.

A Dockerfile plays a central role in the deployment process by specifying the environment in which our application runs. Dockerfile is a script containing a series of instructions for building a Docker image, which is a portable executable package that includes everything needed to run an application: code, runtime, libraries, environment variables, and configuration files.

In our Dockerfile, we performed several key tasks:

- Installation of necessary packages: Ensures that all dependencies required by the Chainlit application and the Langchain integration are available.
- Setup of the Chainlit application: Configures our specific logic and data integration within the Chainlit framework.

- Exposure of the application to an internet-accessible address: Configures the Docker container to listen on a specified port, allowing the application to receive user queries from the internet.

4.3 Evaluation

In this section, we describe the methods that were used to evaluate the performance comparison of the Retrieval Augmented Generator (RAG) system versus standalone Large language model across three areas: the quality of answers, response time, and compute consumption. We will go over the evaluation data, metrics, and processes for each evaluation category.

4.3.1 Response Quality Evaluation

Evaluation dataset

To evaluate the Retrieval Augmented Generator (RAG) system performance against a standalone Large Language Model (LLM), we designed a dataset of specific questions within the Google Cloud Platform domain. Our aim was to cover a wide range of services and their features with the fewest possible questions, but still evaluating the precision and breadth of knowledge required for the use of Google Cloud services.

The dataset was organized into five different categories that represent key areas of Google Cloud Platform offerings:

- **Machine Learning**: Includes information about services related to machine learning models, their training and deployment.
- **Big Data**: Includes information about tools and solutions for processing and analyzing large datasets.
- **Storage Solutions**: Includes information about data storage services that include file storages, relational databases, nosql databases.
- **Infrastructure**: Includes information about the core computing resources such as Compute Engine and Google Kubernetes Engine.
- **Security and Networking**: Includes information about security measures and network configurations for applications and data.

For each category, we made five precise questions that required in-depth knowledge of the respective category to generate an answer. These questions were then given to both the RAG and the LLM, and their answers were saved. Because of the specialized nature of the questions, evaluating the answers required the expertise of domain specialists. This specialists were people working with a google cloud partner company, consulting different industries on how to use google cloud services. They were given questions and answer pairs about the services they felt comfortable using.

An example of how one question and both answers looked like. We did not include the full answers length because it would take up too much space.

> **Category: Machine Learning**
>
> **Question:** *AutoML Model Optimization Parameters: What are the specific hyperparameters that can be manually adjusted when using Vertex AI's AutoML for training a model for tabular data, and how do these parameters impact the model's performance and training time?*
>
> **LLM Answer:** *When using Vertex AI's AutoML for training a model for tabular data, you can manually adjust the following hyperparameters: 1. max train steps: This parameter specifies the maximum number of training steps that the AutoML algorithm will take. The default value is 100,000. Increasing this value may improve the model's accuracy, but it will also increase the training time. 2. batch size: This parameter specifies the number of examples that are used in each training batch. The default value is 100. Increasing this value may improve the model's training speed, but it may also lead to overfitting...*
>
> **RAG Answer:** *Hyperparameters that can be manually adjusted when using Vertex AI's AutoML for training a tabular data model: 1. Data Splitting Parameters: - Split Type: Choose between random, chronological, or stratified data splitting methods. - Percentage: Set the percentage of data rows for training, validation, and test sets. 2. Feature Engineering Parameters: - Transformations: Select transformations for numeric, categorical, and date-time features. - Invalid Data Handling: Specify how to handle invalid data (e.g., drop, impute, or error)...*

This format presents the data clearly and also allows for a direct comparison between the responses from the LLM and RAG, enabling a detailed analysis of each model's effectiveness in generating relevant and accurate answers.

Evaluation metrics

To evaluate the answers generated by Retrieval Augmented Generator (RAG) and standalone Large Language Model (LLM), we used a set of three metrics: Accuracy, Comprehensiveness, and Relevance of References. Each metric is rated on a scale from 1 to 5, where higher scores indicate better performance. Here is a detailed breakdown of each metric and the criteria for scoring:

Accuracy: Measures the factual correctness of the answer.

- **1:** Completely incorrect or irrelevant answer.
- **2:** Mostly incorrect but contains some elements of truth.
- **3:** Partially correct but contains significant inaccuracies.
- **4:** Mostly correct with minor inaccuracies.
- **5:** Fully correct and accurate.

Comprehensiveness: Evaluates the completeness of the answer.

- **1:** Provides no useful information regarding the question, talks about something completely unrelated.
- **2:** Answers a small part of the question with significant gaps.
- **3:** Covers more than half of the necessary content but misses several key points.
- **4:** Almost complete but lacks a few minor details.
- **5:** Fully comprehensive, addressing all aspects of the question.

Relevance of References: Assesses how relevant the documents retrieved by the RAG model are to the answer.

- **1:** None of the documents are relevant.
- **2:** Few documents are relevant (less than 25%).
- **3:** Some documents are relevant (25% - 50%).
- **4:** Most documents are relevant (50% - 75%).
- **5:** All referenced documents are highly relevant (more than 75%).

These metrics allow us to strictly evaluate the performance of the RAG and LLM in delivering correct and complete answers. The last category is just related to RAG, since only this system actually retrieves documents.

Evaluation process

To evaluate the performance of both Retrieval Augmented Generator (RAG) and the standalone Large Language Model (LLM), we distributed those questions about the Google Cloud Platform domain to both systems and recorded their answers. The evaluation of these answers was done by previously mentioned specialists that work in Google Cloud consulting, who are experts in various fields within the platform. Their expertise and contributions are thanked for in the acknowledgments section of this thesis.

The evaluation process was structured to use the expertise of each consultant effectively. Consultants that specialize in different areas such as Data and AI, Infrastructure, or Security, evaluated answers related to their field of expertise. This approach allowed for a accurate assessment of each answer.

All answers provided by both RAG and LLM were compiled and formatted into an evaluation spreadsheet. This spreadsheet served as the tool for the evaluation, where each expert graded the answers based on the previously defined metrics of Accuracy, Comprehensiveness, and Relevance of References in areas they felt they had enough knowledge.

4.3.2 Evaluation of response time

Evaluation dataset

For the evaluation of response time, we used the same dataset we constructed, containing specific questions about the Google Cloud Platform domain. To get a data set of decent quantity for response time measurements, we replicated each question eight times, resulting in a total of 200 queries. This replication was made to get a sufficient number of data points for our analysis. The categories of the questions and questions themselves remained the same: Machine Learning, Big Data, Storage Solutions, Infrastructure, and Security and Networking. Each question, while replicated, was slightly rephrased using LLMs to get bigger diversity in the amount of text that was in the inputs and consequently outputs as well.

We measured the response time for each question and answer for both LLM and RAG, recording the duration needed to generate a response. Since response time is largely influenced by the number of tokens processed, we have also kept track of amount of token in both the input query and the answer.

Evaluation metrics

The primary metric for evaluating response time was the duration in seconds (s) required for each system to generate an answer.

Evaluation process

We measured the response time for each system using the Python time library. For each query, we recorded the time taken to generate a response, as well as the number of tokens in both the input and output, using the Llama tokenizer [TMS+23].

The process involved:

1. Tokenizing the input query to count the number of input tokens.
2. Recording the start time before generating a response.
3. Generating the response using either the LLM or the RAG system.
4. Recording the end time and calculating the difference to determine the response time in seconds.

5. Tokenizing the generated response to count the number of output tokens.

This data, consisting of response times and token counts for both input and output, for both RAG and LLM, was then used to generate different insights and graphs, which will be discussed in the results section.

4.3.3 Evaluation of compute consumption

Evaluating compute consumption was a challenging task, given that all our deployment was in google cloud and we could not get access to precise compute (energy or power) usage metrics due to the abstraction of many compute details in the cloud. To address this, we had two different approaches. The idea is to see, if we look the problem from two different angles would we get similar insights.

The first approach involved a theoretical calculation of the compute based on the specifications of the virtual machines used for our deployments. The second approach utilized billing data, as there is a common correlation between the amount of compute used and the cost incurred. While this comparison is not perfect, it provided the best insights we could obtain given the available cloud abstract nature.

Evaluation dataset

The evaluation consisted of two parts. The first used the same 200 queries as in the previous response time subsection, measuring the cost associated with retrieving each answer. The second part of the dataset was done with the analysis of information and relevant data about similar machines used in our system setup and their power consumption. This information provided the basis for calculating the compute consumption of operations for both the LLM and RAG configurations.

Evaluation metrics

The evaluation metrics we used for the two approaches were: cost in dollars and power consumption in Watts.

Evaluation process

For cost, we got the expense information based on the duration for which each machine was used per request. This included the machine on which the LLM was deployed, and for the RAG, it additionally included the cost for the machine performing the data retrieval. This approach provided a direct measurement of operational costs associated with each query processed by the systems.

For power consumption, we have researched the power consumption of similar machines to the ones used in the cloud. This included finding similar or exact configuration of machines online and getting the information about their power consumption under certain loads. This was done for each machine in process, for their processors (CPUs and GPUs) and memory.

CHAPTER 5

Results and Analysis

This chapter presents the results and analysis of our comparative research diving deep into performance differences and similarities of the Retrieval Augmented Generator (RAG) and the standalone Large Language Model (LLM). The evaluation covers three areas as presented in the chapter about Methodology: answer accuracy, response time, and computational and energy efficiency. Each of these dimensions provides unique insights into the capabilities and limitations of both setups. The findings not only reflect on the performance metrics, but also offer implications for practical deployments and future research directions in the field of LLMs and RAG.

5.1 Response Quality Evaluation

We have executed the process presented under the Methodology chapter. Our evaluation was executed by people that work for several years with services and technologies covered in the questions, therefore could be classified as experts in the field. There were in total 5 subdomains of the questions with 5 questions in each, totaling with 25 questions. In the evaluation process we had 5 experts, that had knowledge in different areas, and graded only the questions they feel had necessary knowledge for. In the end we had at least 2-3 experts grading each question.

Table 5.1 is the core result artifact of this research. In this table we observe significant improvements in the metrics for RAG, in terms of both accuracy and comprehensiveness. We can notice, that RAG demonstrates enhanced accuracy across all of the different categories, indicative of its effectiveness in providing correct information retrieved from sources. This trend is especially seen in categories such as "Machine Learning" and "Security & Networking" where RAG's accuracy scores significantly higher than LLM's, at 3.17 versus 1.50 and 2.50 versus 1.50, respectively. These results suggest that RAG's incorporation of external documents enables it to address gaps in LLM's training data, because for example in case of category machine learning, is quite new and evolving,

Table 5.1: Main results table with answer quality of LLM vs RAG

Problem Category	Accuracy		Comprehensiveness		Relevance of References in RAG
	LLM	RAG	LLM	RAG	
Big data	2.13	3.13	2.67	3.53	3.87
Infrastructure	3.50	5.00	3.50	5.00	5.00
Machine learning	1.50	3.17	2.58	3.92	3.25
Security & Networking	1.50	2.50	1.25	2.25	3.25
Storage solutions	3.20	4.90	3.70	4.70	4.60
Total	**2.21**	**3.60**	**2.70**	**3.80**	**3.86**

therefore there might be some missing information in the training data of the LLM. On the other hand category such as security & networking, might need some really specific and detailed information in the answers, that might get distilled just by the breadth of the LLMs training data.

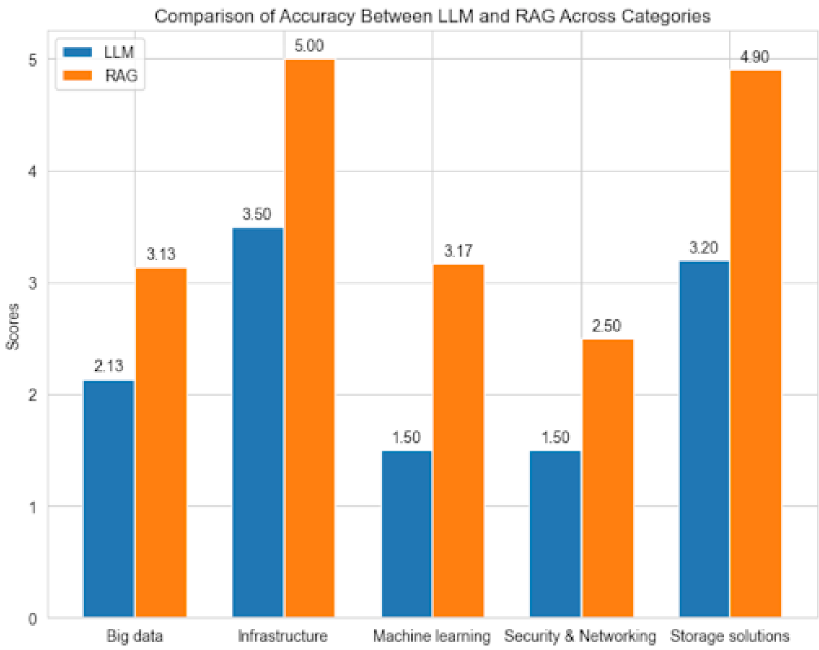

Figure 5.1: Graphic of accuracy distribution of responses of LLM vs RAG for each category

Moreover, the improved comprehensiveness of RAG's responses, which can be seen in the higher mean scores across all categories (e.g., 3.53 vs. 2.67 in "Big Data" and 4.70 vs. 3.70 in "Storage Solutions"), indicates that the referenced documents guide RAG in covering relevant aspects thoroughly within its answers. This is aligned with the observation that categories with specific and technical demands, such as "Machine Learning" and "Security & Networking," benefit significantly from the document-guided approach, enhancing the relevance and depth of the responses.

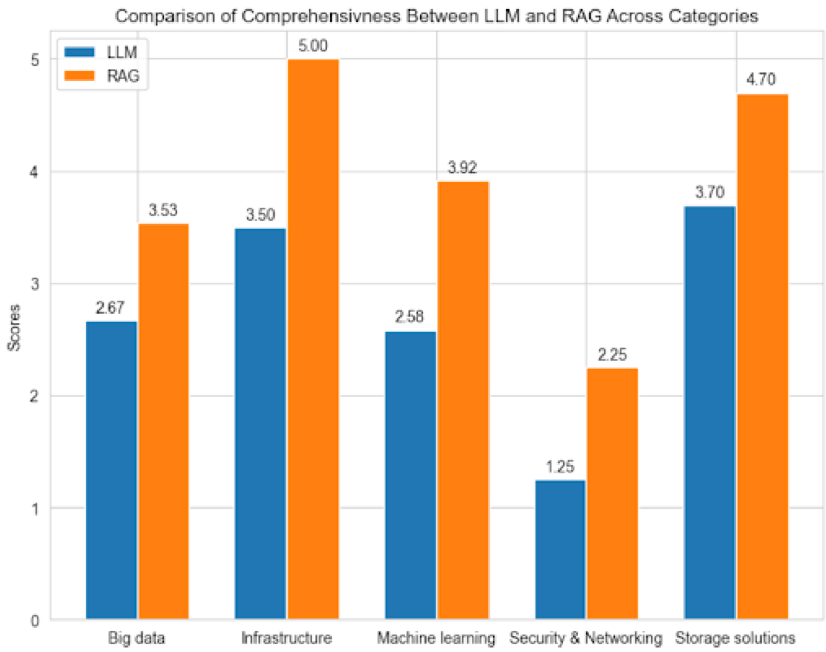

Figure 5.2: Graphic of comprehensiveness distribution of responses of LLM vs RAG for each category

In Figures 5.1 and 5.3 we can see a visual representation of the results table of accuracy and comprehensiveness 5.1. Here it is more obviously seen how RAG improves answers correctness and fullness and to what degree is the change.

The "Relevance of References in RAG" metric also underscores this point, with a mean value of 3.86 suggesting a strong correlation between the quality of references and the overall quality of RAG's responses. When the references are relevant and accurate, RAG's answers tend to be more comprehensive and precise, which is particularly critical in categories requiring up-to-date or specialized knowledge. This relationship is crucial for understanding the enhanced performance of RAG in delivering more accurate and relevant answers compared to LLM. This relationship can be seen in Figure 5.3, and can be classified as a linear relationship or maybe even exponential.

5.2 Response Time Comparison

For the response time comparison, we followed the procedures outlined in the methodology chapter. We collected data from 200 requests for LLM and 200 for RAG. Each response

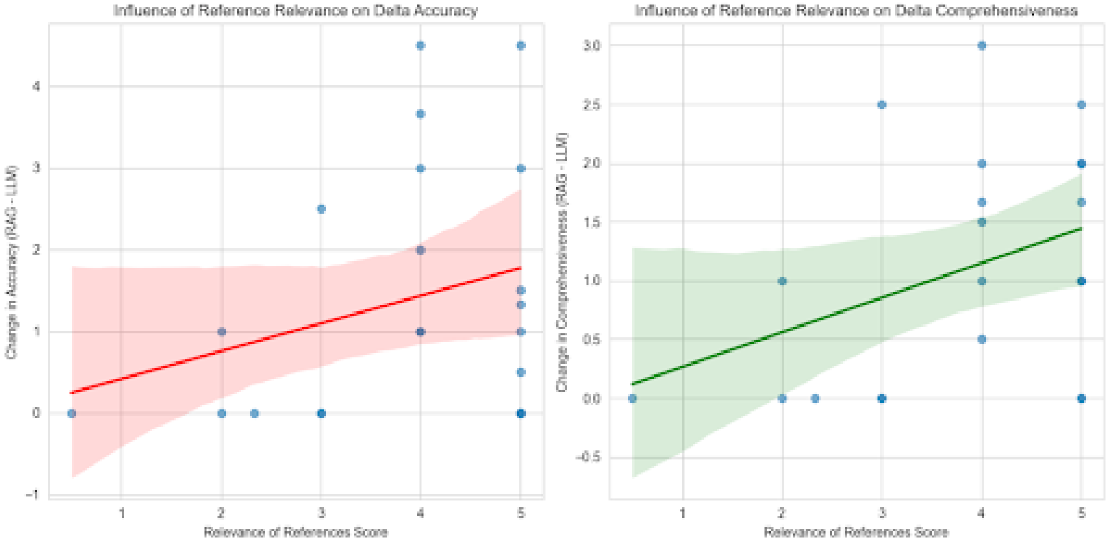

Figure 5.3: Influence of relevance of references in accuracy and comprehensiveness

was analyzed for its answer content, token count, and response time, with a subset of these results displayed in the Table 5.2.

Table 5.2: Examples of answers and their response times and tokens generated

LLM			RAG		
Answer (truncated)	**Response Time**	**Tokens**	**Answer (truncated)**	**Response Time**	**Tokens**
**Entity Matching in Vertex AI Feature Store*...	21.57	756	**Entity Matching in Vertex AI Feature Store...	22.41	723
Implementing feature cross column transformat...	16.33	563	**Steps to Implement Feature Cross Column Tr...	23.78	739
Optimizing query performance and cost in BigQ...	16.53	574	**Optimizing Query Performance and Cost in B...	17.50	486
**1. Enable the BigQuery Cost Optimization AP...	16.77	516	**Configure cost-based access controls in Bi...	16.04	416
Configuring automated data discovery in Datap...	14.59	506	**Configuring Automated Data Discovery in Da...	20.01	589

Additionally, Figure 5.4 illustrates a line graph comparing the response times for both LLM and RAG against the number of tokens generated. It is evident from the graph that the response time for both models is linearly dependent on the number of tokens generated. However, there is a consistent increase in response time for RAG, influenced primarily by the time taken to retrieve documents, which remains relatively constant across requests and different token counts.

In Figure 5.5, we see the distribution of the differences in response times between LLM

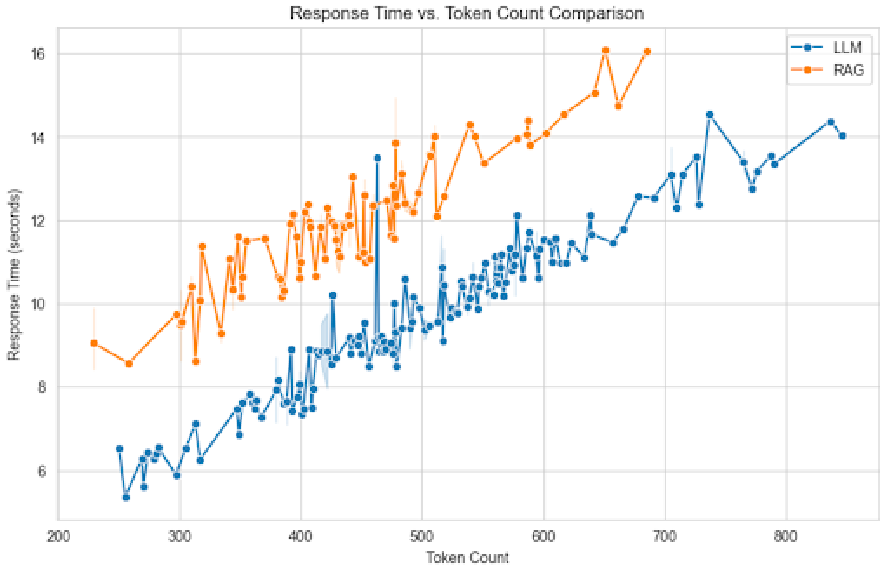

Figure 5.4: Response times of RAG vs LLM compared to tokens generated

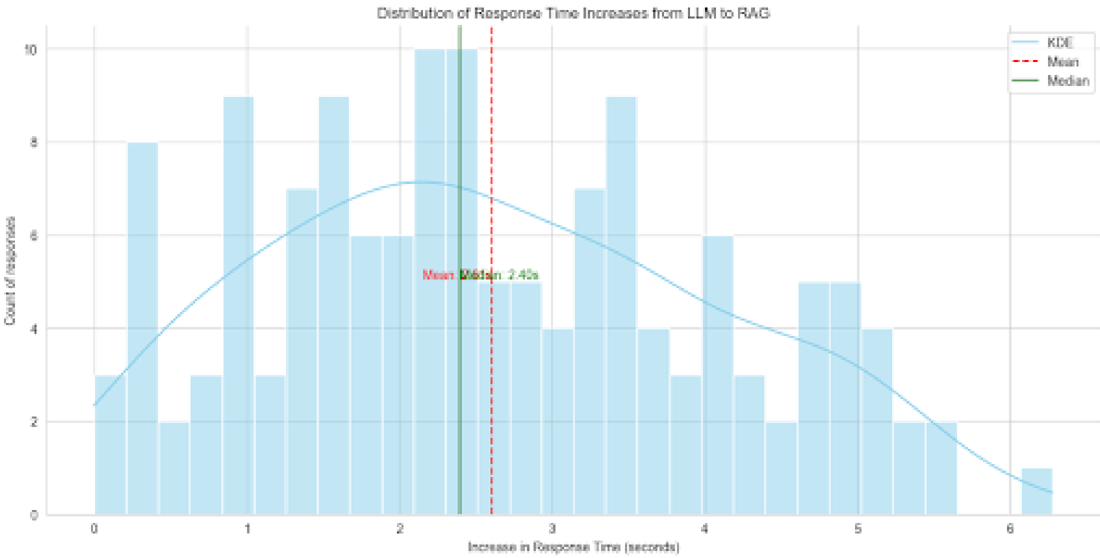

Figure 5.5: Distribution of response time increases from LLM to RAG

and RAG across all requests. The distribution appears normal, with both the mean and median difference being around 2.5 seconds. This finding suggests that while RAG incurs an additional time overhead due to document retrieval, the additional time is stable and predictable, leading to a consistent response time difference. The spread of the times, largely is impacted by the uncertainty of the tokens generated. For example for same query RAG and LLM can have a different count of tokens generated, because of the added documents, because that changes the context of the input query. As we have seen the response time is mostly influenced by output token count, so that is the reasoning.

5.2.1 Input vs Output token count experiment

Another thought we examined was how the input text size (token count) influences the response time of both systems. It is obvious that RAG has a significantly larger input token size, because of the added documents and would be logical that it has some influence on the response times. Based on the architecture of LLMs [VSP+17], on the other side generating output tokens is order of magnitudes more complex task than ingesting the input tokens.

To investigate how this influences response time in real world scenario we have set up a simple experiment. We have limited it to 2 different parameter settings: limit of tokens generated and example queries. We have limited the LLM to generate only 100 and 50 tokens and given it two different queries. Queries were formatted in that way that they are giving the same task, but one is short and clear (10 tokens) and the other one is giving a lot of abstract text and unnecessarily long (1000 tokens). We have given the two prompts for 50 times to the same model and received following results in Table 5.3. We see how from output doubling in token count there is a significant increase in response time and when we increased the input orders of magnitude more (from 10 to 1000) there is no significant change. With this it is clearly displayed, how the only significant influence on response time is output token count and input token count does not influence it significantly. This was important to prove, since in RAG we have a significantly larger input token count and it is important to know this does not influence response time. We can now surely say the only difference in RAG and LLM response time is the time needed to retrieve documents and communication loss in between.

Table 5.3: Average Response Times for LLM Queries

Output token count	**Short input time (s)**	**Long input time (s)**
100 Tokens	1.1450	1.1429
50 Tokens	0.8048	0.8335

5.3 Computational and Energy Efficiency

In this section, we are going to go through results of comparison of the computational and energy efficiency of the Retrieval Augmented Generator (RAG) and the Large Language Model (LLM). Since Google Cloud, where these models are operating, does not provide direct information about the computational and energy consumption, we cannot directly measure this. To work around this, we have chosen two approaches. First, we use theoretical calculations based on the approximations of the virtual machines we are using. Second, we look at billing data as a substitute measure to represent compute consumption. We have covered methods used to achieve this in the methodology chapter

5.3.1 Power calculations

During these calculations we tried to approximate what is the power consumption increase from using just a standalone LLM to using RAG. We have done this by getting approximations of power consumption of what each machine underlying our technologies uses.

Firstly lets just focus on the standalone LLM. Our model was deployed on a machine in google cloud called g2-standard-96. The name itself does not tell too much without understanding the naming schema. When looking in the documentation [Goo24] it is mentioned that it is a machine with 96 virtual cores of CPU and 384 GB of working memory. One of the rare equivalent of machines that has 96 cores of CPUs is AMD Ryzen Threadripper PRO 7995WX and from the following reference [Muj24] it is approximated to have 1000 watts of power drawn. To approximate memory power consumption is pretty hard, but from the following paper [KS17] we can take away that a bigger memory store of around 128 GB draws approximately 20 watts. Therefore lets simplify and conclude our setup with 384 GB would draw approximately 60 watts. This machine has GPU accelerators which are NVIDIA L4 GPU accelerators. They have power consumption of 72 watts per GPU [NVI23]. We can see the total calculation in Figure 5.6 and the final consumption of the machine is 1636 watts.

LLM Configuration Components:

- Machine Type: g2-standard-96
- GPU: 8 × NVIDIA L4 accelerators (72 Watt each)

Power Consumption for LLM:

$$\text{GPU} = 72 \text{ watts per GPU} \times 8 \text{ GPUs} = 576 \text{ watts}$$
$$\text{Machine} = 1000 \text{ watts (CPU)} + 60 \text{ watts (Memory)}$$
$$\textbf{Total} = \textbf{1636 watts}$$

Figure 5.6: Power consumption estimates for the LLM configuration

When looking at the underlying setup of the RAG, we can say it just builds on top of the LLM. So the power consumption is the LLM power consumption with additional power consumption from the information retrieval and langchain orchestration. For simplicity we can say langchain orchestration is negligible, since it can run with minimal memory and compute, for example as a task on the machine that does the information retrieval. For information retrieval we need a small sized machine, we have chosen optimally e2-standard-2 [Goo24] a machine that has 2 cores of CPU and 8 GB of working memory. Virtual machines on google cloud use a concept called virtual cores where from a larger physical machine, they use a specific amount of cores to power a virtual machine and its virtual cores. To approximate this virtual machines power consumption we use a machine that is listed under used physical machines for this class Intel® Xeon® Scalable Platinum 8173M Processor [Int24], and it has a total consumption of 165 watts for 28 cores. This scales to 12 watts for 2 cores. Memory used by this machine can be approximated by an 8 GB RAM memory stick, which on average consume about 3 watts of power. This totals for the additional machine to only 15 watts.

RAG Configuration Components:

- Includes all components from the LLM configuration.
- Vector Search Machine: e2-standard-2
- Langchain Orchestration: Negligible power consumption (ignored for simplicity)

Power Consumption for RAG:

$$\begin{aligned}\text{Vector Search Machine} &= 12 \text{ watts (CPU)} + 3 \text{ watts (Memory)} = 15 \text{ watts}\\ \text{LLM} &= 1636 \text{ watts}\\ \textbf{Total} &= \textbf{1651 watts}\end{aligned}$$

Figure 5.7: Power consumption estimates for the RAG configuration

In the end in Figure 5.8 we can see that the total increase from LLM to RAG is only 0.9%. This reflects the additional energy overhead introduced by incorporating the retrieval component into the language model processing pipeline.

Calculation of Percentage Increase in Power Consumption:

$$\begin{aligned}\text{Power consumption increase} &= \left(\frac{\text{Total RAG Power} - \text{Total LLM Power}}{\text{Total LLM Power}}\right) \times 100\\ &= \left(\frac{1636 \text{ watts} - 1651 \text{ watts}}{1651 \text{ watts}}\right) \times 100\\ &\approx 0.90\%\end{aligned}$$

Figure 5.8: Calculation of Increase in Power Consumption

5.3.2 Billing Data

To complement our theoretical power calculations, we utilized billing data as a practical approach to estimate the computational and energy efficiency of both the Large Language Model (LLM) and the Retrieval Augmented Generator (RAG). Given the lack of direct energy consumption metrics available from Google Cloud, billing data provides an indirect yet insightful proxy for understanding the resource utilization differences between these two setups.

We collected detailed billing records corresponding to the operations of LLM and RAG to identify any significant cost differences which might reflect their differences in energy and computational demands. This analysis focused on the incremental cost impact of integrating RAG's additional document retrieval functionalities with the baseline LLM setup.

The review of the billing data indicated only a marginal increase in costs associated with deploying RAG compared to LLM. We can see from Figure 5.9 that the increase is normally distributed around 0.5%, with maximums around 1 %. This suggests that the additional capabilities of RAG come with minimal financial overhead under typical usage scenarios.

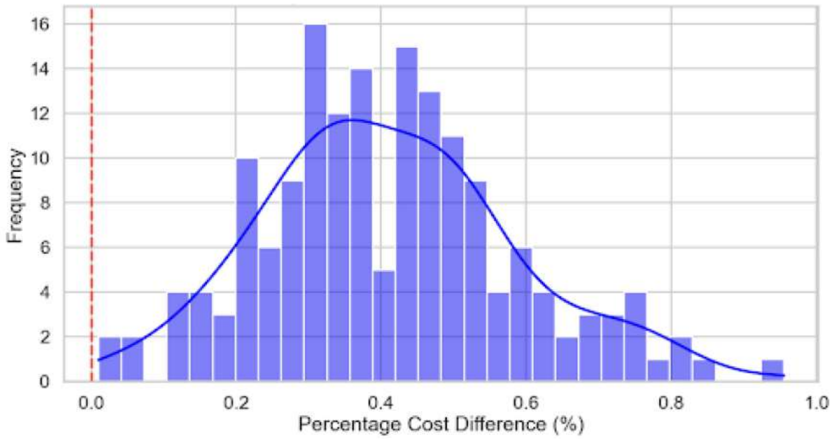

Figure 5.9: Percentage increase in costs from LLM to RAG

CHAPTER 6

Conclusion and Future Work

6.1 Summary of findings

This thesis main goal was to compare the performance of Retrieval Augmented Generators (RAG) and standalone Large Language Models (LLMs) in addressing queries specific to the domain of Google Cloud services. Here are the summarized key findings for each of the research questions:

1. **Quality of responses** - RAG significantly outperformed the standalone LLM in terms of accuracy and comprehensiveness of responses. Accuracy improved by about 62%, and comprehensiveness by approximately 41% when using RAG, demonstrating that the retrieval of relevant documents significantly enhances the quality of the answers. This increase is correlated with the relevance of the retrieved documents, emphasising the importance of accurate information retrieval in improving model responses.

2. **Response time** - The increase in average response times from LLM to RAG was around 2.5 seconds. The study showed that response times were mostly of constant differences and are mostly influenced by the number of output tokens. Input token count did not have an influence on response times as shown in the experiment under 5.2.1. This finding was significant for understanding, because in RAG input token count is significantly higher, so influence on response time would be critical. The consistency in response time suggests that RAG can maintain operational efficiency, making it suitable for real-time applications. All in all we can say that response time is linearly dependant on output token count, with RAG having a constant addition of 2.5s for information retrieval step.

3. **Compute consumption** - To gain insights into compute consumption differences between standalone LLM and RAG, we employed two methods: Power consumption

calculations and billing data insights. The computational overhead introduced by RAG, in comparison to standalone LLMs, was minimal proven by both methods. The increase discovered in power consumption was only about 0.9% when incorporating RAG. Moreover billing data approach mirrored these findings, showing only a 0.5% increase on average in operational costs. This minimal increase in computational load for significant gains in accuracy and comprehensiveness presents RAG as a cost-effective solution in deploying AI models for specialized knowledge domains.

All in all, the substantial improvement in the performance of RAG compared to the standalone LLM, with minimal increases in response time and computational load, emphasises the efficiency of integrating retrieval-based methods in language models. These findings suggest that for specialized knowledge domains such as Google cloud, where accurate and comprehensive responses are critical, RAG offers a promising enhancement to standalone LLMs.

6.2 Limitations and Challenges

During the process of analysing differences of performance of RAG and standalone LLM, we have been faced with few limitations and challenges.

Dependence on Document Quality is known as the primary limitation of the RAG approach. Its dependence on the quality and relevance of the documents stored in the vector database as well as that appropriate documents are actually retrieved is critical. Since the performance of RAG is directly tied to the retrieved documents, any gaps, inaccuracies, or outdated information in the database can significantly impact the accuracy and relevance of the responses. This makes the system as good as the data it can access, as well as retrieve. This requires continuous updates and maintenance of the documentation to ensure that the model can retrieve and utilize the most accurate and current information available.

Moreover one limiting factor is the **Subjectivity in Response Evaluation**. Although the evaluation of the response quality was carried out by experts in the Google Cloud domain to maintain objectivity, the inherent subjectivity of human judgment can influence the results. Different experts may have varying interpretations of what constitutes a "correct" or "comprehensive" answer, leading to potential inconsistencies in the evaluation scores. This subjectivity is a common challenge in research involving qualitative assessments and highlights the difficulty in standardizing the evaluation of AI-generated content across different reviewers.

Influence of Network Latency on Response Time was on additional limiting factor in our research. The response time measurements, crucial for assessing the response time efficiency of RAG, are impacted by network latencies and other cloud infrastructure-related delays. Since all operations are done in a cloud environment, factors such as server response time and network traffic can unpredictably affect the time metrics. This variability introduces an external factor that is difficult to control or quantify, potentially skewing the response time data. From our measurements it was seen that it had not been too much of an impact, but still has to be taken into consideration.

Final limitation factor is important to point out in our research are the **Approximations in Compute Consumption Measurement**. Due to the abstract nature of cloud computing resources, direct measurement of compute consumption was not feasible. Instead, the study relied on theoretical power calculations and billing data as proxy methods to estimate the computational overhead. While these methods provide a reasonable estimation, that resulted in similar insights, they lack the precision of direct measurement techniques, leaving a gap in the accurate quantification of computational resources used by RAG and LLM setups.

6.3 Recommendations for Future Research

Our study highlights several promising avenues for future research and development. These contain either different improvements to the RAG system or diving deeper into different aspect of the comparison. First recommendation would be specific improvements to the RAG functionality such as:

- **Self-Corrective RAG**: Future research could explore the development of a self-corrective RAG system that can automatically detects quality of information in the retrieved documents and based on that retries the search with adapted query. This adaptation could significantly improve the system's performance.
- **RAG with Internet Search Capabilities**: Integrating RAG with real-time internet search functionalities could expand its access to up-to-date information, particularly useful in rapidly evolving domains. This would allow RAG to dynamically retrieve the latest data beyond the static vector database, thereby enhancing the relevance and timeliness of its responses.

Another path one could take into expanding this research would be explore different **embeddings and retrieval methods**, which would focus on comparing multiple types of embeddings and retrieval algorithms to determine which combinations provides the most relevant and accurate document retrievals. This involves experimenting with different embedding models, such as transformer-based, graph embeddings, or hybrid models, and various information retrieval algorithms to optimize the quality of retrieved documents. Additionally to this, to update the data, which is the backbone of information retrieval, one could implement **Automated data update system**. This system would monitor and integrate new and updated content continuously, thereby removing the manual effort required to do this and ensuring the model accesses the most up to date data available.

Finally, interesting addition to this research would be to include the comparison with **Fine-Tuned Models**. Fine tuning large language models is another method of "updating" the models with additional information. A comparative study between RAG and fine-tuned models on specific domains could provide deeper insights into the advantages and limitations of retrieval-augmented approaches versus models tailored through fine tuning on domain-specific data. Fine-tuning requires significant computational resources; thus, research in this direction would also need to consider the cost-benefit analysis of such an approach.

Overview of Generative AI Tools Used

Generative AI tools have been used for grammar checking and reformatting of some text passages to provide a more scientific formulation of my own originally created sentences. Moreover Generative AI tools have been used to expand and brainstorm on ideas for project setup and workflows.

Generative tools have also been used on formatting certain LaTeX objects such as bullet point lists, tables and algorithms - for example Tables 5.1, 5.2 and 5.3 and Algorithms 4.1, 4.2 and 4.3

Following tools have been used in my work:

ChatGPT - Accessed from 1st September 2023 to 1st May 2024
Perplexity AI - Accessed from 1st February 2024 to 1st May 2024
Gemini - Accessed from 1st February 2024 to 1st May 2024

List of Figures

List of Tables

List of Algorithms

Bibliography

[Ant23] Anthropic. Model card: Claude 3, 2023.

[Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *International Conference on Computational Statistics*, 2010.

[cha] Chainlit documentation. https://docs.chainlit.io/get-started/overview. Accessed: 2nd May 2024.

[CND+22] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

[CNW+24] Zhibo Chu, Shiwen Ni, Zichong Wang, Xi Feng, Chengming Li, Xiping Hu, Ruifeng Xu, Min Yang, and Wenbin Zhang. History, development, and principles of large language models-an introductory survey, 2024.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[DS98] Norman R. Draper and Harry Smith. *Applied Regression Analysis*. Wiley-Interscience, 3 edition, 1998.

[GB08] Enzo Grossi and Massimo Buscema. Introduction to artificial neural networks. *European journal of gastroenterology hepatology*, 19:1046–54, 01 2008.

[goo] Google cloud documentation. https://cloud.google.com/docs. Accessed: 15th Jan 2024.

[Goo24] Google Cloud. Machine families, 2024. Accessed: 2024-05-26.

[GSL+20] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, 2020.

[HS06] G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science (New York, N.Y.)*, 313:504–7, 08 2006.

[HYM+23] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, 2023.

[Int24] Intel Corporation. Intel® xeon® platinum 8176m processor specifications, 2024. Accessed: 2024-05-30.

[JMBB05] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America*, 62(S1):S63–S63, 08 2005.

[JSF23] Christopher Potts Matei Zaharia Jon Saad-Falcon, Omar Khattab. Ares: An automated evaluation framework for retrieval-augmented generation systems. *arXiv*, Nov 2023.

[KHM+23] Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models, 2023.

[Koh82] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[KS17] Alexey Karyakin and Kenneth Salem. An analysis of memory power consumption in database systems. pages 1–9, 05 2017.

[Kva85] Tarald O. Kvalseth. Cautionary note about r2. *American Statistician*, 39(4):279–285, 1985.

[lan] Langchain documentation. https://python.langchain.com/docs/. Accessed: 5th March 2024.

[LCL+24] Yang Liu, Jiahuan Cao, Chongyu Liu, Kai Ding, and Lianwen Jin. Datasets for large language models: A comprehensive survey, 2024.

[LDR+24] Jinhyuk Lee, Zhuyun Dai, Xiaoqi Ren, Blair Chen, Daniel Cer, Jeremy R. Cole, Kai Hui, Michael Boratko, Rajvi Kapadia, Wen Ding, Yi Luan, Sai Meher Karthik Duddu, Gustavo Hernandez Abrego, Weiqiang Shi, Nithi Gupta, Aditya Kusupati, Prateek Jain, Siddhartha Reddy Jonnalagadda, Ming-Wei Chang, and Iftekhar Naim. Gecko: Versatile text embeddings distilled from large language models, 2024.

[LL20] Yulong Lu and Jianfeng Lu. A universal approximation theorem of deep neural networks for expressing probability distributions, 2020.

[LLG+20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[LPP+20] Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv*, May 2020.

[LZCS14] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J. Smola. Efficient mini-batch training for stochastic optimization. Association for Computing Machinery, 2014.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[MGH+23] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models, 2023.

[MP43] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[Muj24] Hassan Mujtaba. Amd ryzen threadripper pro 7995wx 96-core cpu delivers world record performance at 5.2 ghz with liquid cooling, 1000w, 2024. Accessed: 2024-05-26.

[NKQ+23] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. A comprehensive overview of large language models, 2023.

[NVI23] NVIDIA Corporation. Nvidia l4 gpu product brief, 2023. Accessed: 2024-05-30.

[PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. volume 14, pages 1532–1543, 01 2014.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. 1986.

[Ric07] Leonard Richardson. Beautiful soup documentation. *April*, 2007.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

[Sam59] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[TBB20] Nick Ryder Melanie Subbiah Jared Kaplan Prafulla Dhariwal Arvind Neelakantan Pranav Shyam Girish Sastry Amanda Askell Sandhini Agarwal Ariel Herbert-Voss Gretchen Krueger Tom Henighan Rewon Child Aditya Ramesh Daniel M. Ziegler Jeffrey Wu Clemens Winter Christopher Hesse Mark Chen Eric Sigler Mateusz Litwin Scott Gray Benjamin Chess Jack Clark Christopher Berner Sam McCandlish Alec Radford Ilya Sutskever Dario Amodei Tom B. Brown, Benjamin Mann. Language models are few-shot learners. *arXiv*, May 2020.

[TMS+23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[TVCM18] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification, 2018.

[VR79] C. J. Van Rijsbergen. *Information Retrieval*. Butterworths, London, 2 edition, 1979.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[YG23] Xinyu Gao Kangxiang Jia Jinliu Pan Yuxi Bi Yi Dai Jiawei Sun Haofen Wang Yunfan Gao, Yun Xiong. Retrieval-augmented generation for large language models: A survey. *arXiv*, Dec 2023.

[YGL+13] Yongtian Yang, Xiaofeng Gao, Xin Lu, Jiaofei Zhong, and Guihai Chen. Distributed ah-tree based index technology for multi-channel wireless data broadcast. In *Database Systems for Advanced Applications*, pages 176–192, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.