People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
National Polytechnic School of Algiers

Graduation Thesis

To obtain the Industrial Engineering diploma

Option : Data Science & Artificial Intelligence

Reliable, fully local RAG agents with LLaMA3

Written by:
Ghribi Ouassim Abdelmalek

Jury:

President:

Mr. Hakim Fourar Laidi - National Polytechnic School of Algiers

Examiners:

Mr. Ayoub Abbaci - National Polytechnic School of Algiers

Supervisors:

Mr. Oussama Arki - National Polytechnic School of Algiers
Mr. Hachem Betrouni - BIGmama Technology

ENP 2024
10, Avenue des Frères Oudek, Hassen Badi, BP. 182, 16200 El Harrach, Alger, Algérie.

In the Name of God, Most Gracious, Most Merciful

Acknowledgments

I extend my deepest gratitude to my thesis supervisors, Mr. Arki Oussama and Mr. Betrouni Hachem, for their indispensable guidance and unwavering support throughout this project. Their expertise and continuous encouragement have been pivotal in shaping its direction. I am sincerely thankful for their patience, insightful feedback, and generous investment of time, all of which have been crucial in refining this work.

On a personal level, I am profoundly grateful to my family for their unwavering support and encouragement throughout this demanding yet rewarding journey. Their unwavering belief in my abilities has been a constant source of motivation, particularly during challenging times.

Moreover, I would like to express my appreciation to all individuals who directly or indirectly contributed to the completion of this thesis. Your support and encouragement have been invaluable and deeply cherished.

Special recognition goes to Mr. Hadj Khelil for his belief in me over the past two years. Progressing from a junior front-end developer to a team lead, overseeing an exceptional team of ambitious individuals, has been an enriching experience. Together, we have strived to develop software with the potential to make a positive impact on the world.

Abstract

Recent advancements in natural language processing have highlighted the need for systems that can effectively retrieve and generate information to handle increasingly complex queries. Combining retrieval and generation processes addresses the limitations of each approach individually, leading to more comprehensive and accurate responses.

This paper presents the implementation of a Retrieval-Augmented Generation (RAG) agent utilizing Llama3 to enhance the accuracy and relevance of responses in complex query environments. The primary challenge is integrating effective information retrieval with advanced generative capabilities to provide precise and reliable answers. Our approach combines document retrieval, grading, and generation within a cohesive system. Queries are assessed for relevance, retrieving pertinent documents or conducting web searches as needed. The generated answers are rigorously evaluated to ensure they meet high standards of accuracy. This implementation demonstrates the potential of merging sophisticated retrieval mechanisms with powerful generative models, resulting in significant improvements in response quality and reliability.

Table of Contents

List of Figures

Glossary

Here are the main acronyms used in this document. The meaning of an acronym is usually indicated once, when it first appears in the text.

ABC	Asset Based Consulting
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
NLP	Natural Language Processing
LLM	Large Language model
GPT	Generative Pre-trained Transformer
BERT	Bidirectional Encoder Representations from Transformers
RAG	Retrieval-Augmented Generation
NN	Neural Networks
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
LSTM	Recurrent Neural Networks
AGI	Artificial General Intelligence
SLM	Statistical Language Model
NLM	Neural Language Model
PLM	Pre-trained Language Model
RLHF	Reinforcement Learning from Human Feedback
RLAIF	Reinforcement Learning from AI Feedback

Chapter 1

State of Play

"The true power of AI lies in its ability to augment human intelligence, not replace it."
Andrew Ng, Co-founder of Coursera and former Chief Scientist at Baidu

This chapter delves into the multifaceted aspects of BIGmama Technology, exploring its mission, vision, and specific methodologies that differentiate it from other players in the field. The company's commitment to integrating AI with human intelligence, its approach to problematization, and the emphasis on hybridization are examined to provide a comprehensive understanding of its operational ethos. Additionally, the chapter highlights BIGmama's proprietary tools, such as the Hyko platform, showcasing their role in democratizing AI technology and fostering collaboration within the broader technological ecosystem.

1.1 Presenting BIGmama Technology

BIGmama Technology is an innovative startup founded in France with a subsidiary in Algeria, specializing in data science and artificial intelligence solutions. With over 9 years of experience as of 2024, the company has established itself as a leading provider of bespoke predictive applications tailored to meet the unique needs of its clients.

Guided by a distinguished board of directors comprising former CEOs of global conglomerates such as Danone and Safran, BIGmama boasts a team of highly skilled data scientists and software engineers. This multidisciplinary team, consisting of more than a dozen experts in their respective fields, brings a wealth of knowledge and expertise to the company's endeavors.

BIGmama's commitment to innovation and cutting-edge technologies has allowed them to deliver state-of-the-art solutions that leverage the power of data science and artificial intelligence. With a strong focus on providing customized solutions, the company has consistently exceeded its clients' expectations, enabling them to gain valuable insights and make data-driven decisions.

Through its strategic partnerships and collaborations, BIGmama continues to push the boundaries of what is possible in the realms of data science and AI (Artificial Intelligence), positioning itself as a driving force in the ever-evolving technological landscape.

1.1.1 Mission

Beyond speeches, BIGmama's mission is to propose effective methodologies, action plans, and tools to:

- Solve problems with artificial intelligence tools.
- Put people at the heart of technology, i.e., the hybridization of AI.
- Make technology a common good, shareable, and accessible to the maximum number of people who can participate and contribute.

The company's commitment extends far beyond mere rhetoric; it is dedicated to developing practical solutions, methodologies, and actionable plans to leverage the power of artificial intelligence for problem-solving. BIGmama places a strong emphasis on ensuring that technology remains human-centric, fostering a harmonious integration and hybridization of AI with human intelligence.

Moreover, BIGmama recognizes the importance of democratizing access to technology, ensuring that it becomes a shared resource that can benefit society as a whole. The company aims to create an environment where the maximum number of individuals can actively participate and contribute to the advancement of technology.

Ultimately, BIGmama's mission is to position technology as a catalyst for freedom, empowering individuals and communities to unlock new possibilities and overcome barriers through innovative solutions.

1.1.2 Vision

"Standing on the Shoulders of Giants"

We find ourselves at a pivotal moment in human history, where the rapid advancements in artificial intelligence are poised to reshape our world in unprecedented ways. Groundbreaking technologies like ChatGPT, developed by industry giants, are at the forefront of this transformative wave, mobilizing vast resources to redefine the boundaries of what is possible.

However, BIGmama's approach is not to compete directly with these titans but rather to harness their innovations as a springboard for its own technological endeavors. By building upon the foundations laid by these industry leaders, BIGmama aims to leverage their advancements as a catalyst for its own visionary project.

Rather than reinventing the wheel, BIGmama recognizes the value in standing on the shoulders of giants, capitalizing on their groundbreaking work to propel its own unique solutions forward. This strategic approach not only allows for more efficient progress but also fosters an environment of collaboration and synergy within the broader technological ecosystem.

With a keen understanding of the rapidly evolving landscape, BIGmama remains agile and adaptable, poised to seize opportunities presented by the advancements of industry giants. By embracing a collaborative mindset and harnessing the collective wisdom of the technological community, BIGmama is well-positioned to contribute its own innovative solutions, shaping the future in alignment with its vision and mission.

1.1.3 BIGmama Specificities

One of the valuable heritages that BIGmama acquired during the 9 years of actively developing bespoke AI applications to its clients is its unique methodology of work. This methodology is centered around the following ideas :

- **Data science starts with problematization:**

  At BIGmama, the approach to data science does not begin with data science itself but rather with reframing and problematization. The company's clients often arrive with a subject or topic rather than a clearly defined problem. BIGmama has developed a methodology to convert these topics into well-defined problems that data scientists can effectively tackle.

- **The data scientist is a "maverick":**

  BIGmama recognizes that the data scientist is a "maverick" who cannot be restrained. The company understands that it cannot impose specific tools or methods on data scientists when it comes to solving problems.

- **Data science a tool to solve problems:**

  BIGmama views data science not as an end in itself but as a means to solve problems. Often, the solution to their clients' problems lies outside the realm of data science tools.

- **Hybridization:**

  BIGmama believes that the future of AI lies in what is commonly referred to as Hybrid AI (Hybrid Artificial Intelligence). This approach encompasses a set of methodologies and techniques aimed at combining the potential of models with purely human knowledge. The company believes that this hybridization allows for putting humans at the heart of technological development and producing tools that are more efficient, easier to maintain, and less expensive.

1.1.4 Products

Historically, BIGmama is a startup specialized in the development of predictive applications for third parties. The company's approach is specific and based on the

principles of hybridization between human and model capabilities. These specificities have led BIGmama to develop its own internal tools. These tools have evolved as the company's needs have changed, transitioning from Iko[1] to Hyko[2]. BIGmama is now willing to make these tools available to the broader technology ecosystem.

As a startup with a rich history in predictive application development, BIGmama has cultivated a unique approach that emphasizes the seamless integration of human expertise and model capabilities. This distinctive methodology, rooted in the principles of hybridization, has driven the company to develop its own suite of proprietary internal tools.

Over time, as BIGmama's requirements have evolved, these tools have undergone continuous refinement and adaptation. The company has transitioned from its initial Iko platform to the more advanced Hyko solution, ensuring that its tools remain aligned with its ever-changing needs and the dynamic technological landscape.

In recognition of the value these tools hold for the broader tech community, BIGmama has made the strategic decision to make them available to the larger ecosystem. By sharing its internally developed tools, the company aims to foster collaboration, knowledge exchange, and the collective advancement of technological solutions.

1.1.5 Hyko

Hyko is a platform that empowers you to become an AI toolmaker. It can be thought of as using "Lego" pieces from its vast toolkit, filled with AI models (over 130,000), APIs (such as Stability.ai, OpenAI, Cohere), web scraping tools, utility functions, and more.

Hyko is a powerful platform that puts the tools and resources necessary for building AI applications at your fingertips. By offering a comprehensive toolkit and streamlined workflow, Hyko empowers users to unleash their creativity and bring their AI-powered visions to life.

Many companies grapple with numerous manual tasks performed repetitively by employees, often numbering in the hundreds or thousands each month. Traditional automation tools struggled to automate these tasks fully due to their reliance on human reasoning. However, LLMs have revolutionized automation by exposing reasoning capabilities through an API.

In response to this advancement, Hyko emerged as an AI-first platform, designed not merely to streamline employee workflows but to entirely replace them from start to finish. The platform's automation builder offers unparalleled flexibility, enabling the automation of highly complex workflows with ease.

[1] https://big-mama.io/en/iko
[2] https://www.hyko.ai

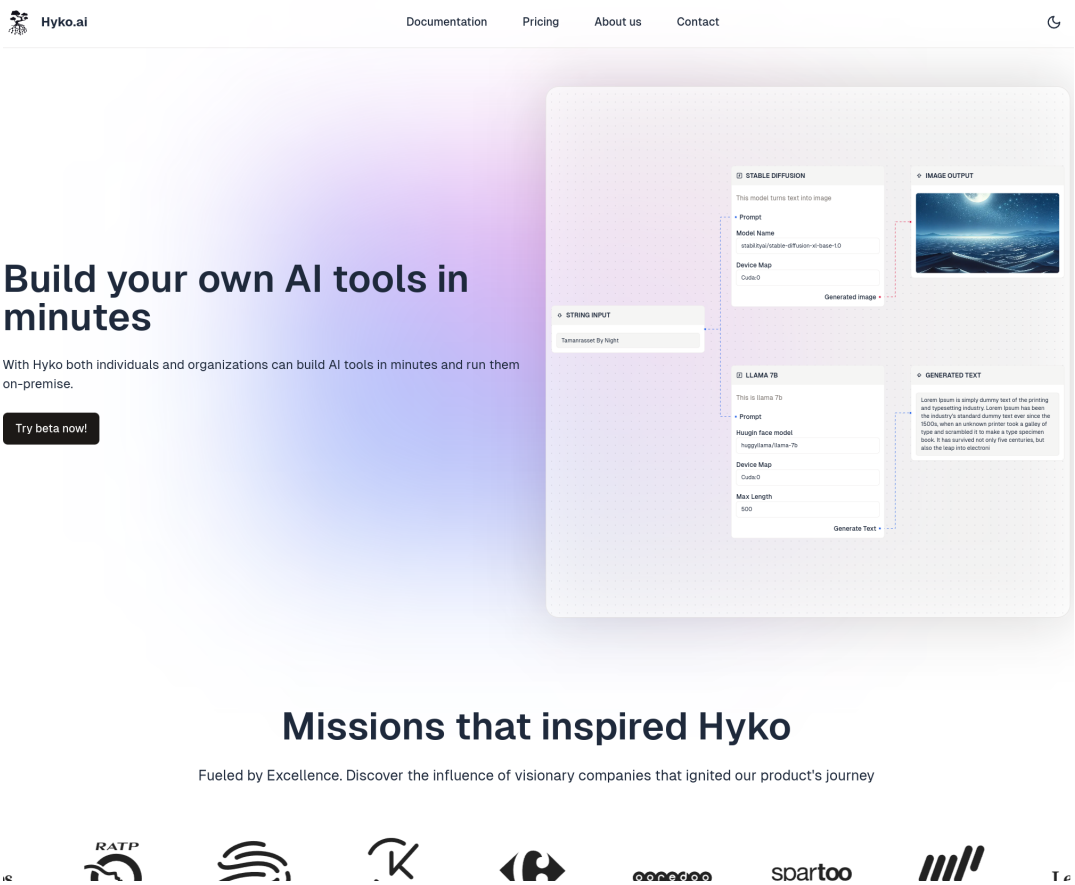


Figure 1.1: *Hyko's Landing page*

1.2 Problem Definition

The future value of AI is anticipated to derive not solely from AI engineers or data scientists, but also from field experts who possess the capability to integrate their domain knowledge with AI technologies to address everyday challenges. This shift underscores the growing importance of domain-specific expertise in harnessing the potential of AI for practical applications.

This imperative is frequently encountered among Hyko users, who often find themselves in need of extracting vital insights from their diverse data repositories, which encompass web sources, PDF documents, and email archives. In response to this demand, Rag systems emerge as key facilitators, particularly when coupled with open-source LLMs such as Llama3, Phi3, and mixtral7B. Rag systems excel in the retrieval, aggregation, and generation of pertinent information, empowering users to distill valuable insights from their data reservoirs efficiently and effectively. This seamless integration enhances the capacity for nuanced understanding and actionable intelligence extraction, thereby augmenting the utility and versatility of Hyko across

diverse information retrieval and analysis endeavors.

1.3 RAG: Enhancing LLMs with External Knowledge

LLMs showcase impressive capabilities but encounter challenges like hallucination [1], outdated knowledge, and non-transparent, untraceable reasoning processes. RAG has emerged as a promising solution by incorporating knowledge from external databases. This enhances the accuracy and credibility of the generation, particularly for knowledge-intensive tasks, and allows for continuous knowledge updates and integration of domain specific information. RAG synergistically merges LLMs' intrinsic knowledge with the vast, dynamic repositories of external databases [2].

General-purpose language models can be fine-tuned to achieve several common tasks such as sentiment analysis and named entity recognition. These tasks generally don't require additional background knowledge.

For more complex and knowledge-intensive tasks, it's possible to build a language model-based system that accesses external knowledge sources to complete tasks. This enables more factual consistency, improves reliability of the generated responses, and helps to mitigate the problem of "hallucination".

Meta AI researchers introduced a method called Retrieval Augmented Generation (RAG) [3] to address such knowledge-intensive tasks. RAG combines an information retrieval component with a text generator model. RAG can be fine-tuned and its internal knowledge can be modified in an efficient manner and without needing retraining of the entire model.

RAG takes an input and retrieves a set of relevant/supporting documents given a source (e.g., Wikipedia). The documents are concatenated as context with the original input prompt and fed to the text generator which produces the final output. This makes RAG adaptive for situations where facts could evolve over time. This is very useful as LLMs's parametric knowledge is static. RAG allows language models to bypass retraining, enabling access to the latest information for generating reliable outputs via retrieval-based generation.

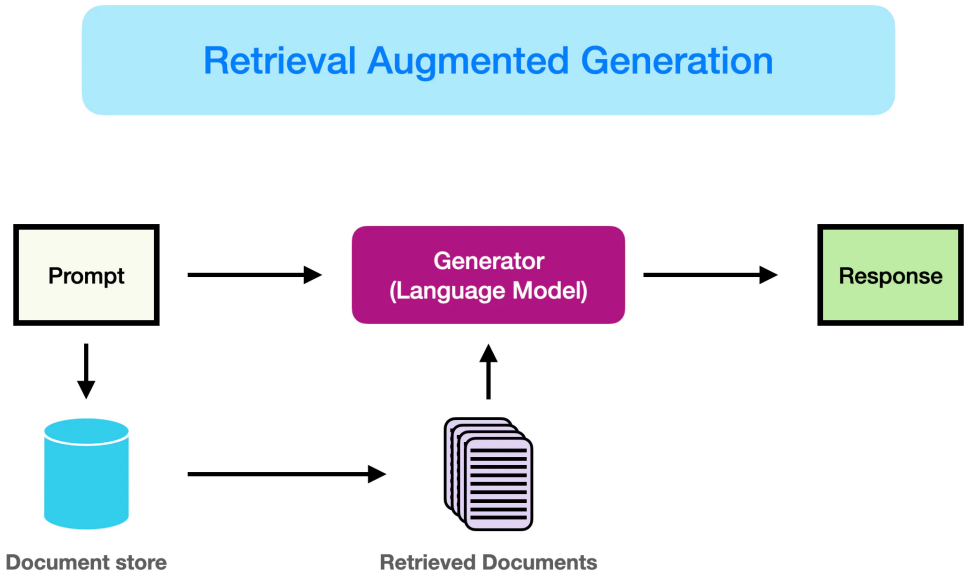


Figure 1.2: *Retrieval Augmented Generation.*

In short, the retrieved evidence obtained in RAG can serve as a way to enhance the accuracy, controllability, and relevancy of the LLM's response. This is why RAG can help reduce issues of hallucination or performance when addressing problems in a highly evolving environment.

1.4 Conclusion

In this chapter, we lay the foundation for the subsequent sections of this thesis. We introduce fundamental concepts in Retrieval-Augmented Generation (RAG) systems and Large Language Models (LLMs), along with a basic overview of the host company, BIGmama Technology.

Chapter 2

State of the Art

In recent years, the remarkable advancements in Natural Language Processing (NLP) have been primarily driven by the development of LLMs. These LLMs, such as GPT (Generative Pretrained Transformer)[4] and BERT (Bidirectional Encoder Representations from Transformers)[5], have demonstrated remarkable capabilities in understanding and generating human-like text. However, despite their impressive performance, LLMs still face challenges in effectively retrieving and incorporating relevant context for generating accurate and coherent responses.

Enter RAG systems, a novel approach that seeks to overcome the limitations of traditional LLMs by integrating retrieval mechanisms with generation models. RAG systems combine the strengths of both retrieval and generation techniques to enhance the quality and relevance of generated text.

This chapter provides a comprehensive exploration of RAG systems, delving into their architecture, components, training processes, applications, advantages, and challenges. We begin by establishing a foundational understanding of LLMs and their evolution, laying the groundwork for understanding the need for RAG systems. We then proceed to dissect the intricacies of RAG systems, discussing the role of retrieval in providing context and the role of generation in producing fluent responses.

Through detailed examination and analysis, we uncover the inner workings of RAG systems, exploring how retrieval and generation components interact within the architecture. Real-world applications and use cases of RAG systems across various domains are elucidated, demonstrating their potential to revolutionize tasks such as question answering, dialogue generation, and content creation. Furthermore, we evaluate the advantages and limitations of RAG systems compared to traditional LLMs and other approaches in NLP.

In summary, this chapter serves as a comprehensive guide to RAG systems, offering readers a deep dive into one of the most promising advancements in NLP. As we navigate through the complexities and potentials of RAG systems, we pave the way for understanding their role in shaping the future of human-computer interaction and language understanding.

2.1 Machine Learning

2.1.1 Definition

Machine Learning, often abbreviated as ML, is a subset of artificial intelligence (AI) that focuses on the development of computer algorithms that improve automatically through experience and by the use of data. In simpler terms, machine learning enables computers to learn from data and make decisions or predictions without being explicitly programmed to do so [6].

At its core, machine learning is all about creating and implementing algorithms that facilitate these decisions and predictions. These algorithms are designed to improve their performance over time, becoming more accurate and effective as they process more data.

In traditional programming, a computer follows a set of predefined instructions to perform a task. However, in machine learning, the computer is given a set of examples (data) and a task to perform, but it's up to the computer to figure out how to accomplish the task based on the examples it's given.

For instance, if we want a computer to recognize images of cats, we don't provide it with specific instructions on what a cat looks like. Instead, we give it thousands of images of cats and let the machine learning algorithm figure out the common patterns and features that define a cat. Over time, as the algorithm processes more images, it gets better at recognizing cats, even when presented with images it has never seen before.

This ability to learn from data and improve over time makes machine learning incredibly powerful and versatile. It's the driving force behind many of the technological advancements we see today, from voice assistants and recommendation systems to self-driving cars and predictive analytics.

2.1.2 Relationships to other fields

Artificial Intelligence (AI) encompasses the field of computer science dedicated to creating systems capable of emulating human-like intelligence, problem-solving, and decision-making. Machine Learning (ML) is a subset of AI focused on enabling computers to learn from data without being explicitly programmed. Within ML, Deep Learning stands out as a subfield that employs neural networks with multiple layers to learn complex representations of data, particularly effective in tasks like image and speech recognition. ML and Deep Learning are integral components of AI, providing the framework for developing intelligent systems capable of learning, reasoning, and adapting to new information, thereby advancing the capabilities of AI across various domains.

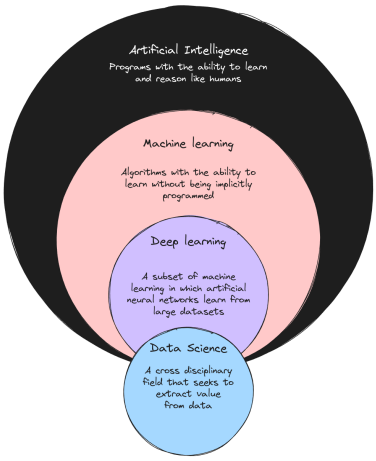


Figure 2.1: *Machine learning as subfield of AI [6].*

2.1.3 Types of Machine Learning

Machine Learning can be broadly categorized into several types based on the learning approach, the availability of labeled data, and the feedback mechanism. The main types of machine learning are:

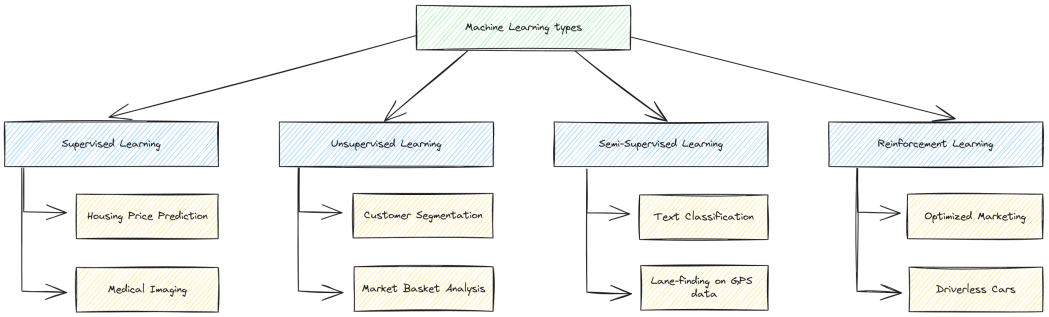


Figure 2.2: *Types of machine Learning.*

Supervised Learning

Supervised machine learning learns patterns and relationships between input and output data. It is defined by its use of labeled data. A labeled data is a dataset that contains a lot of examples of Features and Target. Supervised learning uses algorithms that learn the relationship of Features and Target from the dataset. This process is referred to as Training or Fitting [7].

Unsupervised Learning:

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled data sets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Unsupervised learning models are utilized for three main tasks—clustering, association, and dimensionality reduction [8].

Semi-Supervised Learning:

Semi-supervised learning is a relatively new and less popular type of machine learning that, during training, blends a sizable amount of unlabeled data with a small amount of labeled data. Semi-supervised learning is between supervised learning (with labeled training data) and unsupervised learning (unlabeled training data).

Semi-supervised learning offers a lot of real-world applications. There is a dearth of labeled data in many fields. Because they involve human annotators, specialized equipment, or expensive, time-consuming studies, the labels (target variable) could be challenging to get [7].

Reinforcement Learning:

Reinforcement learning (RL) is a machine learning technique that trains software to make decisions to achieve the most optimal results. It mimics the trial-and-error learning process that humans use to achieve their goals [9].

2.1.4 Limitations of Machine Learning: Challenges and Considerations

Although machine learning is a powerful technique for extracting knowledge from data, it also has certain limitations that are important to consider:

- **Data dependency:** Machine learning heavily relies on the quality and quantity of training data. If the data is poorly labeled or unrepresentative, it can lead to errors in predictions.
- **Overfitting:** When the model is too complex compared to the training data, it can overfit and not generalize well to test data. This can also result in poor performance for new data.
- **Explainability:** Machine learning models can be very complex and difficult to understand. It can be challenging to understand how the model makes decisions and to explain these decisions to users.

- **Biased data:** Training data can be biased due to factors such as data selection, human biases, or measurement errors. This can lead to biased predictions for test data.
- **Lack of diversity:** Machine learning models may lack diversity in the types of data they can handle. For example, machine learning models may struggle to process unstructured data such as images, sounds, and texts.
- **Computational cost:** Machine learning algorithms may require high computational power and significant storage resources to process large amounts of data. This can be costly and time-consuming to train and implement models.

2.2 Deep Learning

2.2.1 Definition

Deep learning is a type of machine learning that teaches computers to perform tasks by learning from examples, much like humans do. Imagine teaching a computer to recognize cats: instead of telling it to look for whiskers, ears, and a tail, you show it thousands of pictures of cats. The computer finds the common patterns all by itself and learns how to identify a cat. This is the essence of deep learning.

In technical terms, deep learning uses something called "neural networks," which are inspired by the human brain. These networks consist of layers of interconnected nodes that process information. The more layers, the "deeper" the network, allowing it to learn more complex features and perform more sophisticated tasks [10].

2.2.2 Deep Learning vs. Machine Learning

Deep learning stands apart from traditional machine learning in its approach to data and learning methods. Machine learning algorithms typically rely on structured, labeled data for predictions, where specific features are defined and organized into tables. While machine learning can handle unstructured data, it often requires preprocessing to structure it. In contrast, deep learning streamlines this process by directly processing unstructured data such as text and images. It automates feature extraction, reducing reliance on human experts. For instance, in categorizing pet photos, deep learning algorithms autonomously identify key features, like ears, crucial for distinguishing between animals. In contrast, machine learning requires manual feature hierarchy establishment by human experts.

2.2.3 Deep Learning Applications

Deep learning has a wide range of applications across various domains due to its ability to learn complex patterns from large volumes of data. Some of the different

types of applications for deep learning include:

Image Recognition and Computer Vision:

- Deep learning is extensively used for tasks such as image classification, object detection, facial recognition, and image segmentation.
- Applications include self-driving cars, medical image analysis, surveillance systems, and augmented reality.

Natural Language Processing (NLP):

- Deep learning is employed for understanding and generating human language, enabling tasks such as sentiment analysis, machine translation, text summarization, and chatbots.
- Applications include virtual assistants, language translation services, social media sentiment analysis, and customer support chatbots.

These are just a few examples of the diverse applications of deep learning, demonstrating its versatility and impact across various industries and fields.

2.3 Natural Language Processing (NLP)

Natural Language Processing (NLP) serves as a crucial technology within artificial intelligence, facilitating communication between humans and computers. It represents a multidisciplinary field empowering machines to comprehend, analyze, and produce human language, thus facilitating seamless human-machine interaction. The importance of NLP manifests in its diverse applications, spanning automated customer support to instantaneous language translation, showcasing its pivotal role in modern computing.

2.3.1 What is Natural Language Processing?

Natural Language Processing (NLP) is a sub-discipline of computer science providing a bridge between natural languages and computers. It helps empower machines to understand, process, and analyze human language. NLP's significance as a tool aiding comprehension of human-generated data is a logical consequence of the context-dependency of data. Data becomes more meaningful through a deeper understanding of its context, which in turn facilitates text analysis and mining. NLP enables this with the communication structures and patterns of humans [11].

NLP encompasses the task of enabling machines to comprehend, interpret, and generate human language in a manner that is not only valuable but also meaningful. OpenAI[1], renowned for pioneering sophisticated language models such as ChatGPT[2], underscores the significance of NLP in fostering the development of intelligent systems capable of comprehending, responding to, and generating text. This advancement in technology serves to enhance user-friendliness and accessibility across various applications.

2.3.2 How Does NLP Work?

NLP is a fascinating field that delves into the intricate mechanisms underlying human language comprehension and generation by machines. This section aims to unravel the complexities of NLP, shedding light on the fundamental principles and techniques that drive its functionality. By exploring the inner workings of NLP, we gain insight into how machines process and analyze natural language data, paving the way for groundbreaking applications in artificial intelligence and human-computer interaction. Through this exploration, we embark on a journey to discover the algorithms, models, and methodologies that empower machines to navigate the vast landscape of human language with precision and sophistication.

[1] https://openai.com/
[2] https://chat.openai.com

Components of NLP

Natural Language Processing is not a monolithic, singular approach, but rather, it is composed of several components, each contributing to the overall understanding of language. The main components that NLP strives to understand are Syntax, Semantics, Pragmatics, and Discourse.

Syntax: Syntax pertains to the arrangement of words and phrases to create well-structured sentences in a language.

Semantics: Semantics is concerned with understanding the meaning of words and how they create meaning when combined in sentences.

Pragmatics: Pragmatics deals with understanding language in various contexts, ensuring that the intended meaning is derived based on the situation, speaker's intent, and shared knowledge.

Discourse: Discourse focuses on the analysis and interpretation of language beyond the sentence level, considering how sentences relate to each other in texts and conversations.

NLP techniques and methods

NLP employs a diverse array of techniques and methodologies to analyze and comprehend human language. Below are some foundational techniques utilized in NLP:

Tokenization: This process involves segmenting text into individual units, such as words, phrases, or symbols, known as tokens.

Parsing: Parsing entails examining the grammatical structure of a sentence to extract its meaning and syntactic relationships.

Lemmatization: This technique involves reducing words to their base or root form, facilitating the grouping of different word forms with the same meaning.

Named Entity Recognition (NER): NER is utilized to identify and classify entities within text, such as persons, organizations, locations, and other named items.

Sentiment Analysis: This method enables the assessment of the sentiment or emotion expressed in a piece of text, aiding in understanding the underlying mood or opinion.

What is NLP Used For?

With some of the basic concepts now defined, one can explore how natural language processing is applied in the modern world.

Automatic Translation: Automatic translation systems use NLP techniques to translate texts from one language to another.

Chatbots and Virtual Assistants: Chatbots and virtual assistants use NLP to understand user's natural language and provide appropriate responses.

Automatic Summarization: NLP algorithms can be employed to summarize lengthy documents into a few sentences.

Sentiment Analysis: NLP is utilized to analyze sentiments expressed in text, which can be beneficial for businesses in assessing customer satisfaction.

Information Extraction: NLP systems can extract important information such as names, locations, and dates from texts.

Speech Recognition: Speech recognition systems utilize NLP techniques to convert speech into text.

Autocorrection: NLP algorithms are utilized in autocorrection programs to suggest grammatical and spelling corrections.

Text Analysis: NLP is used to analyze large volumes of text to detect trends, themes, and patterns.

Automatic Text Generation: NLP enables the automatic generation of text for various applications, such as report writing or content creation.

2.4 Neural networks

Neural Networks (NNs) are computational models composed of layers of neurons that can learn from data [12]. They are versatile and robust models capable of learning directly from raw data, without the need for manually selected features [13]. NNs employ a training algorithm known as backpropagation, which adjusts the model's parameters based on a loss function [13].

2.4.1 Feedforward networks

A feedforward neural network is characterized by its architecture, devoid of cycles, where the output of layer i can be computed using the outputs from layer $i-1$ [13]. The architecture of a neural network encompasses its structure, including the number of hidden layers and neurons, as well as the functions employed for computations. These selections, known as hyperparameters, are parameters whose values are not determined by the learning algorithm [14]. Figure 2.3 shows a typical feedforward network architecture.

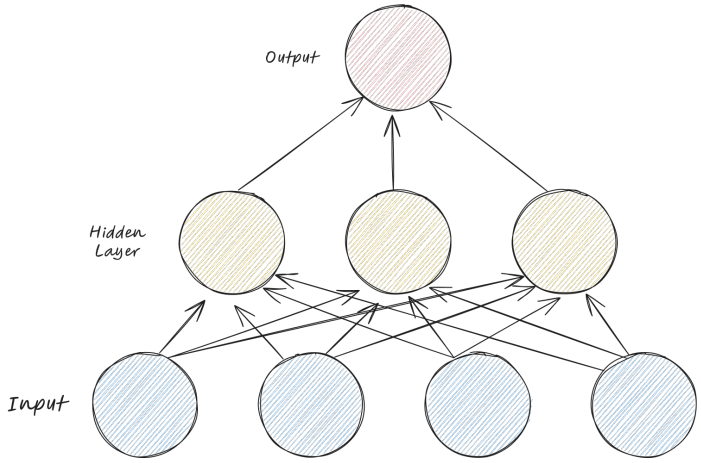


Figure 2.3: *Example of a feedforward neural network architecture. The network has an input layer with four neurons, one hidden layer with three neurons and an output layer with a single neuron [13].*

As depicted in Figure 2.3, neural networks comprise an input layer, n hidden layers, and an output layer. The input layer receives the data that the network needs to process, which then traverses through the hidden layers before reaching the output layer. The output layer provides the model's result for the given input data. When a neural network contains multiple hidden layers, it qualifies as a deep neural network (DNN) and falls within the domain of deep learning [12]. Each layer consists of i neurons, with connections between each neuron in a layer and every neuron in the previous layer, except for the input layer.

Neurons serve as fundamental computational units and, as previously indicated, establish connections with all neurons in the preceding layer. Each connection is characterized by a numerical parameter referred to as a weight [13].

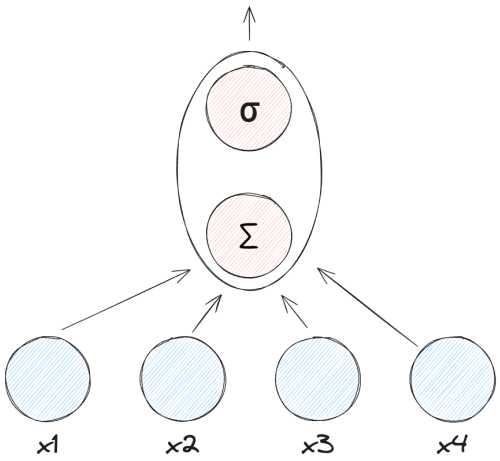


Figure 2.4: *Example of computation within a single neuron, where the weighted sum of inputs is passed through an activation function to get the output of the neuron [13].*

As show in Figure 2.4, the neuron receives inputs from neurons of the preceding layer, denoted as $x_1, x_2, ..., x_n$. Each neuron-to-neuron connection is assigned a weight, representing the strength of the connection. The output from a preceding neuron is multiplied by the weight of the connection and then summed for each connected neuron, yielding the weighted sum of values.

Although not visually represented in Figure 2.4, a bias parameter is introduced to the weighted sum, thereby altering a_j to

$$a_j = \sum_{j'} w_{jj'} x_{j'} + b \tag{2.1}$$

The bias serves to adjust the neuron's output independently of the input. This capability enables the model to alter the neuron's output during the learning process without necessitating adjustments to the weights, thereby facilitating finer control [14].

The weighted sum is subsequently forwarded to an activation function, which determines the neuron's output or its degree of "activation." An illustration of such an activation function is the sigmoid function denoted as σ, as depicted in Figure 2.4. The mathematical expression for the sigmoid function is:

$$\sigma(a_j) = \frac{1}{1 + e^{-a_j}} \tag{2.2}$$

According to Nielsen [14], two other common activation functions include $\tanh \phi$,

which is defined as

$$\phi(a_j) = \tanh(a_j) = \frac{e^{a_j} - e^{-a_j}}{e^{a_j} + e^{-a_j}} \tag{2.3}$$

and Rectified Linear Unit (ReLU), which is defined by equation

$$R(a_j) = max(0, a_j) \tag{2.4}$$

Symbol a_j refers to the result of adding together the weighted sum and bias. The outputs of these functions are visualized in Figure 2.5.

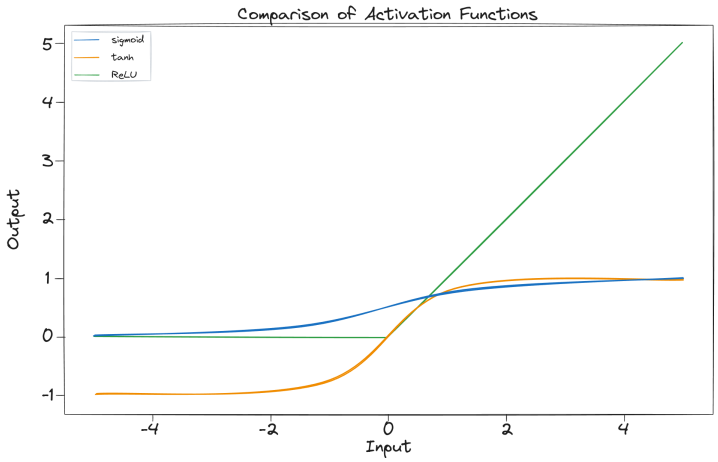


Figure 2.5: *Visualization of the sigmoid, tanh and ReLU activation function outputs.*

The outputs produced by these activation functions exhibit non-linearity, signifying that the input parameter does not linearly determine the output value, as evident from Figure 2.5. This concept holds significance in neural networks, enabling them to learn non-linear systems [15]. While several activation functions are commonly employed, ReLU has gained prominence in deep learning due to its demonstrated efficacy in enhancing the performance of numerous neural networks [13]. Typically, the same activation function is applied across all neurons, except for those in the output layer.

The choice of activation function for the output layer hinges on the specific task assigned to the neural network. In regression-based tasks, a linear activation function is employed to obtain the summed weighted output of the preceding neurons. Conversely, for classification problems involving k output neurons, a softmax function is often utilized. The softmax function is defined as:

$$\hat{y} = \frac{e^{a_k}}{\sum_{j=1}^{J} e^{a_j}} \tag{2.5}$$

and ensures that all neuron outputs sum to one on the output layer [13].

2.4.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are specialized neural networks designed for processing sequential data. They operate by incorporating iterations that retain past states, enabling the network to leverage previous inputs as context for the current input [16]. Text serves as a prime example of sequential data, where individual words can be viewed as singular data points within the sequence. Consequently, RNNs can effectively process textual input by utilizing preceding words to predict the subsequent ones. Figure 2.6 provides an illustration of a basic RNN architecture.

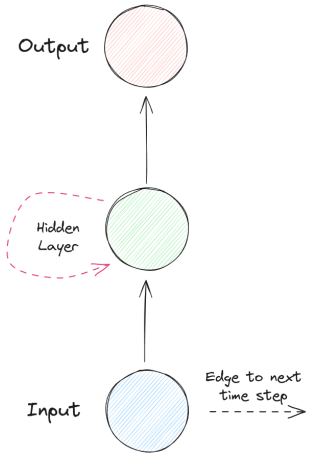


Figure 2.6: *Visualization of an RNN with one input neuron, one hidden neuron and one output neuron [13].*

The neuron on the hidden layer stores the hidden state, as visualized by the dashed line. The activation of the hidden state can be defined as

$$H_t = \phi_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{2.6}$$

where H_t is the hidden state at time step t, ϕ_h is the activation function of the neuron, X_t is the neuron input at time step t, W_{xh} is a weight matrix, W_{hh} is the weight matrix of the hidden state and b_h is bias [16].

RNNs are trained utilizing the backpropagation through time (BPTT) algorithm, which is an adaptation of the standard backpropagation algorithm tailored for networks with a sequential order of computations. In this setup, the output at time step t depends on the states from preceding steps [17]. During training, the forward pass computes through all time steps, and the resultant loss is utilized to update the parameters across all time steps. One way to conceptualize this process is by envisioning an unrolled RNN, where the recurrent loops are eliminated, resulting in a network structure akin to a feedforward neural network. However, due to the nature of the hidden state dynamics, RNNs are notorious for encountering gradient-related

issues during training. As illustrated in Figure 2.7, if the weight associated with the dotted line is less than one, future values may decrease exponentially [13]. Conversely, weights that start to grow can lead to the exploding gradient problem [16]. When gradients become too small or too large, it can impede the training process of the model.

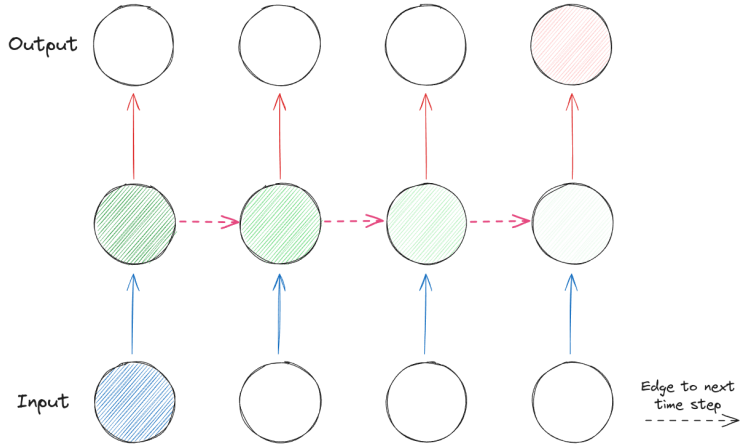


Figure 2.7: *Visualization of the vanishing gradient problem, which is indicated by the hidden neuron color fading away [13].*

Despite offering a robust framework for sequential learning, RNNs are hindered by significant limitations arising from the aforementioned issues. One critical limitation stems from the gradient problem encountered during training, which can constrain the effective handling of lengthy sequences. Consequently, RNNs may struggle to capture dependencies between inputs across extended sequences. To address these challenges, numerous techniques and alternative architectures have been proposed. Among these, the Long Short-Term Memory (LSTM) architecture stands out as one of the most prominent examples, and its discussion follows.

2.4.3 Long Short-Term Memory

The Long Short-Term Memory (LSTM) [18] architecture addresses the gradient-related challenges inherent in RNNs by incorporating constant error, memory cells, and gate units. This design enables the network to effectively handle sequences spanning over 1000 time steps. The memory cells employ three gate units: an input gate I_t, an output gate O_t, and a forget gate F_t, which regulate the flow of information. Specifically, the input gate facilitates the addition of information to the memory cell, the output gate facilitates information retrieval from the cell, and the forget gate facilitates cell resetting [16]. Moreover, the memory cells possess an internal state with a self-connected recurrent edge featuring a constant weight, ensuring consistent error propagation across time steps and mitigating previously discussed gradient-

related issues [13]. Additionally, a candidate memory cell $\tilde{C}$, utilized for proposing new information, is integrated with the old memory content C_{t-1} through the gates to govern the preservation of old memory in the new memory C_t [16]. Figure 2.8 illustrates the complete architecture of the LSTM memory cell.

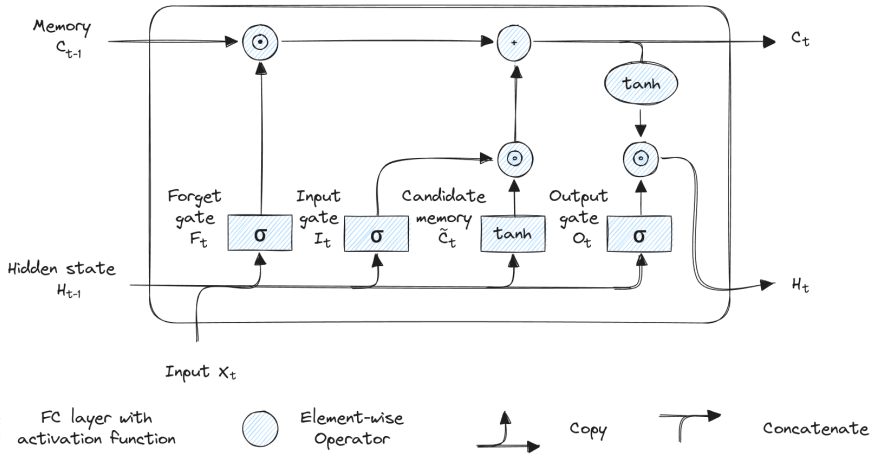


Figure 2.8: *Visualization of an LSTM memory cell [16].*

Although the LSTM architecture enhances performance compared to the RNN architecture discussed in subsection 2.4.2, it still possesses limitations. Like the RNN architecture, models utilizing LSTM must process the previous time step before computing the next, rendering them computationally intensive and impeding parallelization. Additionally, LSTM typically lacks an explicit attention mechanism and tends to prioritize the most recent words. These constraints are addressed by the Transformer model, which will be introduced next.

2.5 Transformers

LLMs are primarily based on the Transformer architecture, which has become the foundation for various state-of-the-art NLP models. In this section, we will discuss the main components of the Transformer architecture.

2.5.1 Transformer Architecture

The transformer architecture is a groundbreaking neural network architecture designed for NLP tasks. It was introduced by Vaswani et al. in the paper "Attention is All You Need"[19] The architecture relies on the self-attention mechanism to process and generate sequences, making it highly efficient and scalable compared to traditional RNNs and LSTM models. The problem in RNNs and LSTMs is that their network sequence makes it hard to process long sentences and the ability to perform tasks in parallel is affected by sequential computation. Transformer approaches these problems by using encoders, decoders, and self-attention.
The transformer model is shown in Figure 2.9 below.

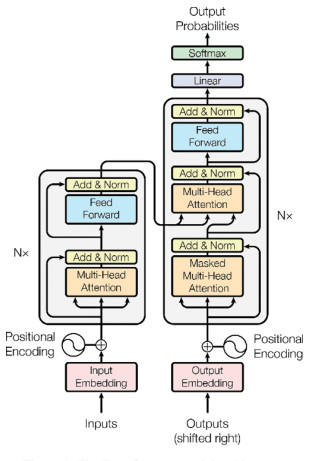


Figure 2.9: *Transformer architecture model [19].*

2.5.2 Components of the Transformer Architecture

Figure 2.9 depicts the sequence flow beginning at the base of the embedding block, where input and output tokens undergo conversion into fixed-size vectors represented by numerical values corresponding to the tokens. Subsequently, positional encoding is applied to the tokens, assigning them position-specific values to track their sequential order. This step compensates for the absence of recursion and convolution within the network [19]. Progressing along the input pathway, the encoder stack follows, comprised of N identical layer structures, each containing two sub-layers: a multi-head attention network and a feedforward network. The output from the encoder

is then forwarded to the subsequent encoder. On the output side of the sequence, a decoder stack is present, also composed of N layers. Each decoder layer features a masked multi-head attention network for outputs, followed by multi-head attention incorporating encoder stack outputs. Analogous to the encoder, a feedforward network concludes the decoder. Normalization is performed between all sub-layers. Subsequently, the decoder output is linearized, and the softmax activation function is applied to transform output token values into a probability distribution. The resulting sequence yields a list of token probabilities, representing the transformer's predictions for suitable tokens given the input sequence.

2.5.3 Self-Attention Mechanism

In the article "Attention is all you need" the self-attention network is called Multi-Head Attention and consists of linearly projecting inputs, Scaled Dot-Product Attention layers, output concatenation, and finally, projecting the output values [19]. The self-attention network is introduced in Figure 2.10 below.

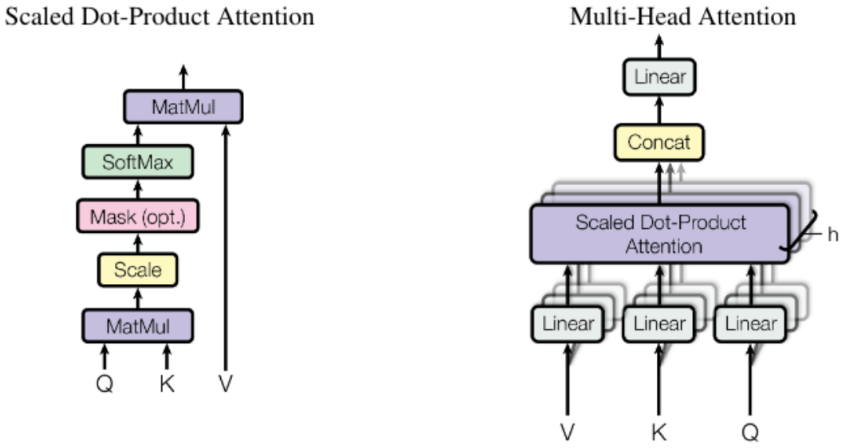


Figure 2.10: *Multi-Head attention network [19].*

The network's inputs comprise matrices containing query, key, and value vectors for each token. The matrix Q represents query vectors, K corresponds to keys, and V represents values. To simplify these terms, envision the query as akin to a search engine query initiated by a user, where the key value signifies the title of the search results, and the value encapsulates the content within the titles. These matrices correspond to sentences composed of tokens. Utilizing the scaled dot-product attention mechanism, the network produces tokens with associated attention scores. Multiple self-attention layers, denoted by h, operate concurrently, a process referred to as multi-head attention. This mechanism enables the model to track token positions effectively. The attention scores generated by self-attention can be expressed using

Equation 2.7.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt[2]{d_k}})V \tag{2.7}$$

The dot product, performed on matrices Q and K, is divided by the square root of d_k, representing the dimensionality of the key vectors [19]. This division serves as a scaling factor. Subsequently, the $softmax$ activation function is applied to this result, which is then multiplied by V to obtain the attention matrix. This process is repeated h times to generate multiple multi-head attention matrices. These matrices are subsequently concatenated into a single linearized matrix, which is then fed into the feedforward network.

The feedforward network employed within the transformer architecture shares similarities with the network outlined in Figure 2.9, but with variations in the number of layers, input and output dimensions, activation function, and other parameters across different models. In the "Attention is all you need" article, two linear transformations with ReLU activation were utilized, featuring output and input dimensions of $d_{model} = 512$, and an inner-layer dimensionality of 2048 [19]. Within the realm of LLMs, some of these parameters can be regarded as hyperparameters, exhibiting variability across models. For instance, in the Chinchilla model, the dimensionality is $d_{model} = 8192$, with the feedforward network size consistently four times the former value [20].

2.5.4 Variants of Transformer Architecture

Various LLMs are built on top of the Transformer architecture with slight modifications or adaptations. Some popular variants include:

GPT: The Generative Pre-trained Transformer (GPT) is an autoregressive model that utilizes only the decoder part of the Transformer architecture to generate text [4].

BERT: Bidirectional Encoder Representations from Transformers (BERT) is based on the encoder part of the Transformer architecture and is pre-trained using masked language modeling and next sentence prediction tasks [5].

T5: The Text-to-Text Transfer Transformer (T5) adapts the original Transformer architecture to a unified text-to-text format, enabling it to be used for various NLP tasks with minimal task-specific modifications [21].

2.6 Large language models (LLMs)

2.6.1 Introduction

Language modeling is a long-standing research topic, dating back to the 1950s with Shnnon's application of information theory to human language, where he measures how well simple n-gram language models predict or compress natural language text [22]. Since then, statistical language modeling became fundamental to many natural language understanding and generation tasks, ranging from speech recognition, machine translation, to information retrieval [23] [24].

Recent advances in transformer-based large language models (LLMs), pretrained on web-scale text corpora, have significantly enhanced the capabilities of these models. For instance, OpenAI's ChatGPT and GPT-4 are now used not only for natural language processing but also as general task solvers, exemplified by their integration into Microsoft's Co-Pilot systems. These models can follow complex human instructions and perform multi-step reasoning when necessary. Consequently, LLMs are becoming fundamental building blocks for the development of general-purpose AI agents or artificial general intelligence (AGI).

LLMs are large-scale, pre-trained, statistical language models based on neural networks. Their recent success is the result of decades of research and development in language modeling, which can be divided into four distinct waves, each with its own starting points and progression: statistical language models, neural language models, pre-trained language models, and LLMs.

Early neural language models (NLMs) [25], [26], [27], [28] address data sparsity by mapping words to low-dimensional continuous vectors, known as embedding vectors, and predicting the next word based on the aggregation of these vectors using neural networks. The embedding vectors learned by NLMs create a hidden space where the semantic similarity between vectors can be easily computed as their distance. This enables the computation of semantic similarity between any two inputs, regardless of their forms (e.g., queries versus documents in web search [29], [30], sentences in different languages in machine translation [31], [32]) or modalities (e.g., image and text in image captioning [32], [33]). However, early NLMs are task-specific models, as they are trained on task-specific data, resulting in a hidden space that is also task-specific.

Pre-trained language models (PLMs), unlike early neural language models (NLMs), are task-agnostic. This generality also extends to the learned hidden embedding space. The training and inference of PLMs follow the pre-training and fine-tuning paradigm, where language models utilizing recurrent neural networks [34] or transformers [5], [35], [36] are pre-trained on web-scale unlabeled text corpora for general tasks such as word prediction. They are then fine-tuned for specific tasks using small amounts of labeled, task-specific data. Recent surveys on PLMs include [37], [38],

[39].

Large language models (LLMs) mainly refer to transformer-based neural language models that contain tens to hundreds of billions of parameters. These models, such as PaLM [40], LLaMA [41], and GPT-4 [42], are pre-trained on massive text datasets. Compared to pre-trained language models (PLMs), LLMs are not only significantly larger in model size but also exhibit stronger language understanding and generation abilities. More importantly, they demonstrate emergent abilities not present in smaller-scale language models. As illustrated in Figure 2.11, these emergent abilities include:

In-context learning: LLMs can learn a new task from a small set of examples presented in the prompt at inference time.

Instruction following: After instruction tuning, LLMs can follow instructions for new types of tasks without explicit examples.

Multi-step reasoning: LLMs can solve complex tasks by breaking them down into intermediate reasoning steps, as demonstrated in the chain-of-thought prompting [43].

LLMs can also be augmented with external knowledge and tools [44], [44], enabling them to effectively interact with users and the environment [44], and continually improve using feedback data collected through interactions, such as via reinforcement learning with human feedback (RLHF).

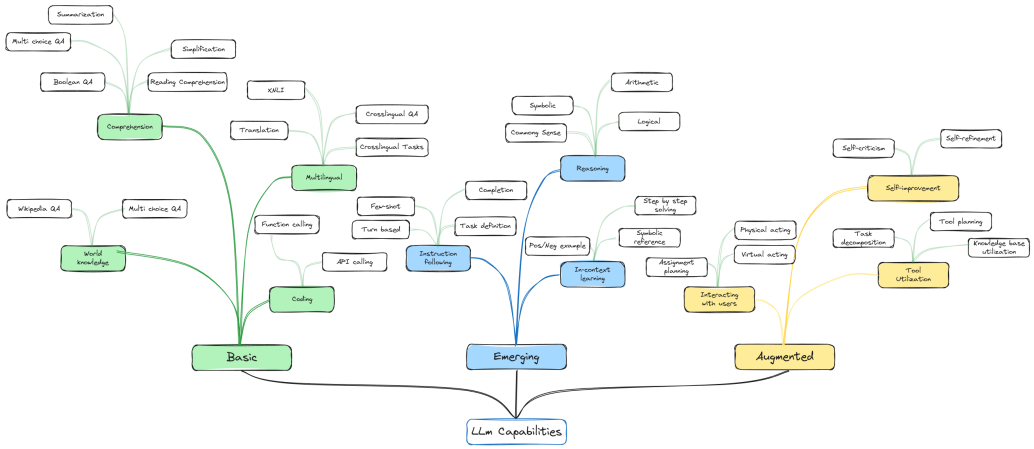


Figure 2.11: *LLM capabilities [45]*

2.6.2 Large Language Model Families

Large language models (LLMs) primarily refer to transformer-based pre-trained language models (PLMs) that encompass tens to hundreds of billions of parameters. Compared to the previously discussed PLMs, LLMs are significantly larger in size and demonstrate enhanced capabilities in language understanding, generation, and

emergent phenomena that are absent in smaller models. This section will examine three prominent LLM families: GPT, LLaMA, and PaLM, as illustrated in Figure 2.12.

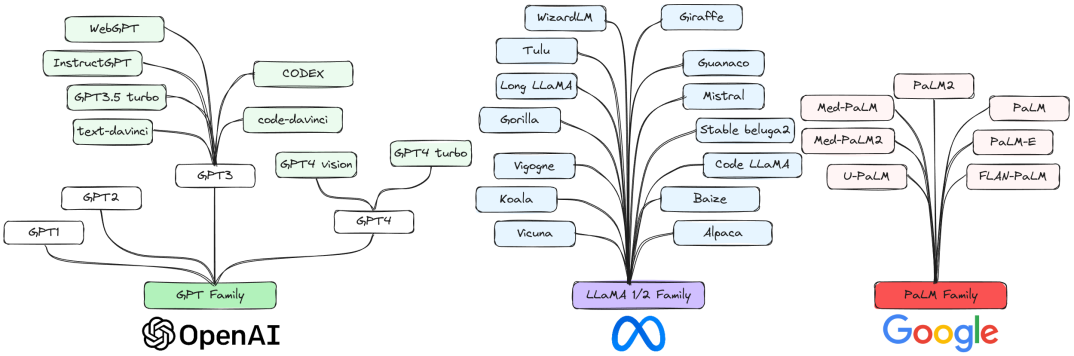


Figure 2.12: *Popular LLM families [45]*

The GPT Family: Generative Pre-trained Transformers (GPT) are a series of decoder-only Transformer-based language models developed by OpenAI. This series includes models such as GPT-1, GPT-2, GPT-3, InstrucGPT, ChatGPT, GPT-4, CODEX, and WebGPT. While the earlier models, such as GPT-1 and GPT-2, are open-source, the more recent models, including GPT-3 and GPT-4, are proprietary and accessible only through APIs.

The LLaMA Family: LLaMA is a series of foundational language models released by Meta. In contrast to GPT models, LLaMA models are open-source, with model weights made available to the research community under a noncommercial license. Consequently, the LLaMA family has expanded rapidly, as these models are widely utilized by numerous research groups to develop improved open-source language models to compete with proprietary models or to create task-specific models for mission-critical applications.

The PaLM Family: The PaLM (Pathways Language Model) family, developed by Google, includes the first PaLM model [40] announced in April 2022 and kept private until March 2023. This transformer-based LLM features 540 billion parameters and is pre-trained on a high-quality text corpus containing 780 billion tokens, covering a broad spectrum of natural language tasks and applications. The model's pre-training utilized 6144 TPU v4 chips within the Pathways system, facilitating highly efficient training across multiple TPU Pods. PaLM showcases the ongoing advantages of scaling, achieving state-of-the-art few-shot learning results on hundreds of language understanding and generation benchmarks. The PaLM-540B model not only surpasses state-of-the-art fine-tuned models on various multi-step reasoning tasks but also performs comparably to humans on the newly introduced BIG-bench benchmark.

2.6.3 Tokenizations

Tokenization refers to the process of converting a sequence of text into smaller units known as tokens. While the simplest tokenization tools split text based on whitespace, most rely on a word dictionary. However, this approach faces the out-of-vocabulary (OOV) problem, as the tokenizer can only recognize words present in its dictionary. To address this, popular tokenizers for LLMs are based on sub-words, which can be combined to form a vast array of words, including those not seen in the training data or from different languages. The following sections describe three popular tokenizers [45].

BytePairEncoding

BytePairEncoding (BPE) is originally a data compression algorithm that leverages frequent patterns at the byte level to compress data. The algorithm primarily aims to retain frequent words in their original form while breaking down less common words. This approach ensures that the vocabulary remains manageable in size while adequately representing common words. Additionally, BPE effectively represents morphological variations of frequent words if their suffixes or prefixes are commonly found in the training data.

WordPieceEncoding

This algorithm is predominantly used in well-known models such as BERT and Electra. At the beginning of training, it incorporates the entire alphabet from the training data to ensure that no characters are left as UNK (unknown). This scenario occurs when the model encounters input that cannot be tokenized, typically involving untokenizable characters. Similar to BytePairEncoding, this approach aims to maximize the likelihood of including all tokens in the vocabulary based on their frequency.

SentencePieceEncoding

Although the previously described tokenizers offer significant advantages over whitespace tokenization, they still assume that words are always separated by white spaces. This assumption does not hold true for all languages, where words can be disrupted by various noisy elements such as unwanted spaces or invented words. SentencePieceEncoding aims to address this issue.

2.6.4 Positional Encoding

Positional encoding is a technique used in transformer models to provide the model with information about the position of tokens in a sequence. This is crucial because transformer models process tokens in parallel, lacking inherent knowledge of token

order. Positional encoding ensures that the model can differentiate between tokens based on their position, which is essential for tasks where word order matters, such as language translation and text generation

Absolute Positional Embeddings

Absolute Positional Embeddings (APE) [19] has been used in the original Transformer model to preserve the sequence order information. Consequently, positional information of words is added to the input embeddings at the base of both the encoder and decoder stacks. Various options exist for positional encodings, including learned or fixed methods. In the vanilla Transformer, sine and cosine functions are utilized for this purpose. However, the main drawback of using APE in Transformers is its limitation to a fixed number of tokens. Moreover, APE does not account for the relative distances between tokens.

Relative Positional Embeddings

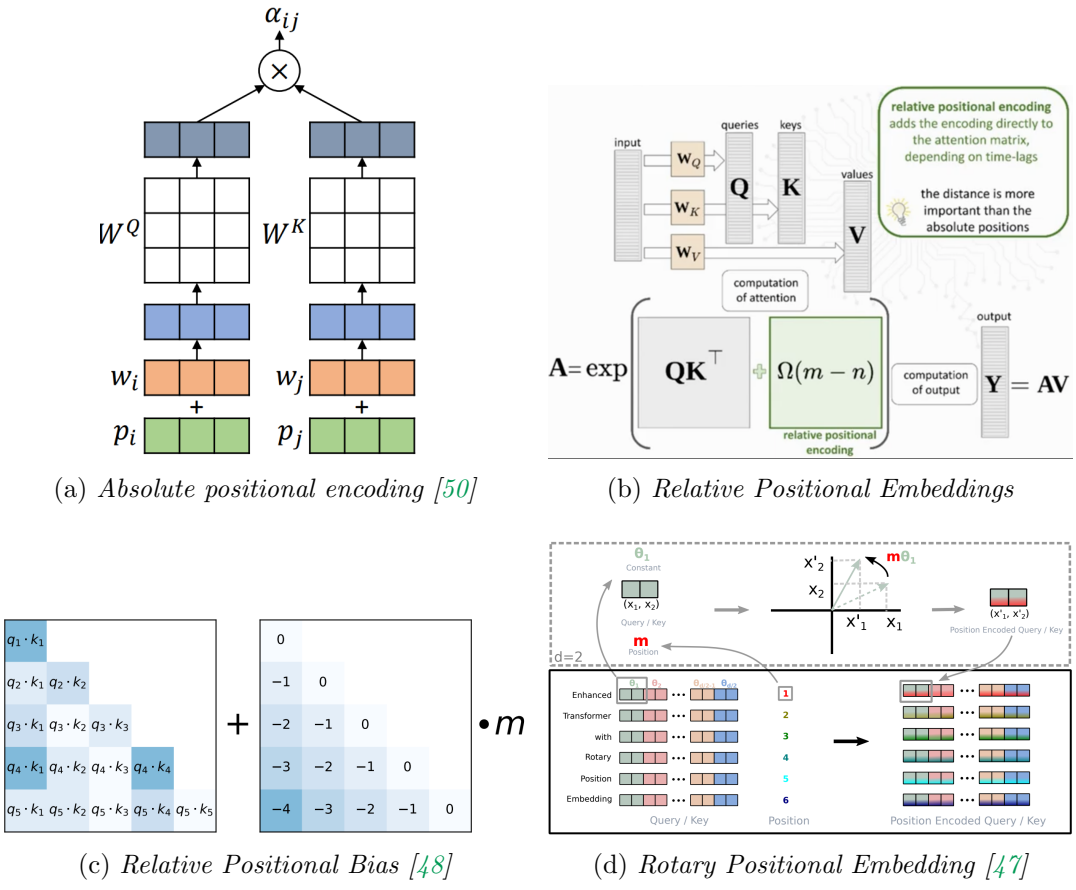
Relative Positional Embeddings(RPE) [46] extend self-attention to consider the pairwise connections between input elements. RPE is integrated into the model at two levels: first, as an additional component to the keys, and subsequently, as a sub-component of the values matrix. This methodology views the input as a fully-connected graph with labels and directed edges. In the context of linear sequences, edges can encode information about the relative positional disparities between input elements. A clipping distance, denoted as $k^2 \leq k \leq n-4$, specifies the maximum limit on relative locations. This mechanism enables the model to make sensible predictions for sequence lengths not present in the training data.

Rotary Position Embeddings

Rotary Positional Embedding (RoPE) [47] addresses limitations inherent in existing approaches. Learned absolute positional encodings may lack generalizability and significance, particularly in the context of short sentences. Additionally, current methods such as T5's positional embedding struggle with constructing a complete attention matrix between positions. RoPE employs a rotation matrix to encode the absolute position of words while concurrently incorporating explicit relative position details in self-attention. This approach offers several advantageous features, including flexibility with varying sentence lengths, reduced word dependency as relative distances increase, and enhancement of linear self-attention through relative position encoding. Notably, models like GPT-NeoX-20B, PaLM, CODEGEN, and LLaMA leverage RoPE within their architectures.

Relative Positional Bias

The rationale behind this form of positional embedding is to enable extrapolation during inference for sequences longer than those encountered during training. In their work [48], Press et al. introduced Attention with Linear Biases (ALiBi). Rather than solely adding positional embeddings to word embeddings, they introduced a bias to the attention scores of query-key pairs, imposing a penalty proportional to their distance. The BLOOM [49] model leverages ALiBi as part of its architecture.



(a) *Absolute positional encoding [50]*

(b) *Relative Positional Embeddings*

(c) *Relative Positional Bias [48]*

(d) *Rotary Positional Embedding [47]*

Figure 2.13: *Various positional encodings employed in LLMs*

2.6.5 Model Pre-training

Pre-training is the very first step in the large language model training pipeline, helping LLMs acquire fundamental language understanding capabilities, which can be useful in a wide range of language-related tasks. During pre-training, the LLM is trained on a massive amount of (usually) unlabeled texts, typically in a self-supervised manner. There are different approaches used for pre-training, such as next sentence prediction [5]. The two most common techniques include next token prediction (au-

toregressive language modeling) and masked language modeling.

In the **Autoregressive Language Modeling** framework, given a sequence of n tokens $x_1, \ldots, x_n$, the model aims to predict the next token x_{n+1} (and occasionally the subsequent sequence of tokens) in an autoregressive manner. A commonly used loss function in this context is the log-likelihood of the predicted tokens, as demonstrated in Equation 2.8.

$$\mathcal{L}_{ALM}(x) = \sum_{i=1}^{N} p(x_{i+n}|x_i, \ldots, x_{i+n-1}) \tag{2.8}$$

Given the autoregressive nature of this framework, decoder-only models are inherently better suited to learn and perform these tasks.

In **Masked Language Modeling**, certain words in a sequence are masked, and the model is trained to predict these masked words based on the surrounding context. This approach is sometimes referred to as denoising autoencoding. If we denote the masked or corrupted samples in the sequence x as $\tilde{x}$, the training objective can be expressed as:

$$\mathcal{L}_{MLM}(x) = \sum_{i=1}^{N} p(\tilde{x}|x\backslash\tilde{x}) \tag{2.9}$$

More recently, **Mixture of Experts (MoE)** [51], [52] have gained significant popularity in the LLM space. MoEs allow models to be pre-trained with substantially less compute, enabling a dramatic increase in model or dataset size within the same compute budget as a dense model. An MoE architecture consists of two main components: Sparse MoE layers, which replace dense feed-forward network (FFN) layers, and contain several "experts" (e.g., 8), each of which is a neural network. Typically, these experts are FFNs, but they can also be more complex networks. A gate network, or router, determines which tokens are sent to which expert. Notably, a token can be routed to multiple experts. Routing tokens to experts is a critical decision in MoE design; the router consists of learned parameters and is trained concurrently with the rest of the network. Figure 29 illustrates a Switch Transformer encoder block, commonly used in MoEs.

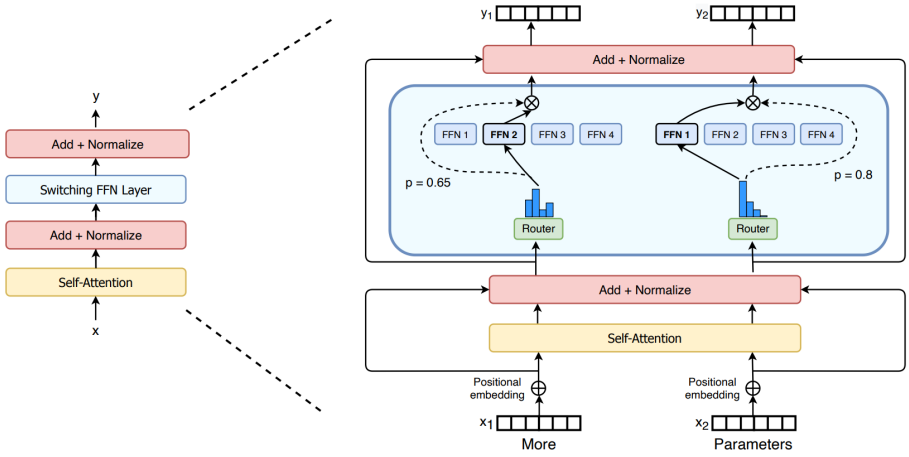


Figure 2.14: *Illustration of a Switch Transformer encoder block. They replaced the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue) [52].*

2.6.6 Fine-tuning and Instruction Tuning

Early language models, such as BERT, trained using self-supervision, were not able to perform specific tasks directly. To make the foundation model useful, it needed to be fine-tuned for a specific task using labeled data, a process known as supervised fine-tuning (SFT). For instance, in the original BERT paper [5], the model was fine-tuned for 11 different tasks. While more recent LLMs no longer require fine-tuning to be functional, they can still benefit from task-specific or data-specific fine-tuning. For example, OpenAI reports that the much smaller GPT-3.5 Turbo model can outperform GPT-4 when fine-tuned with task-specific data[3].

Fine-tuning does not have to be limited to a single task; various approaches to multi-task fine-tuning exist (see, e.g., Mahabi et al. [53]). Fine-tuning on one or more tasks is known to improve results and reduce the complexity of prompt engineering, offering an alternative to retrieval-augmented generation. Additionally, there are other compelling reasons to fine-tune a model. For instance, fine-tuning can expose the model to new or proprietary data that it did not encounter during pre-training.

An essential motivation for fine-tuning LLMs is to ensure that their responses align with the expectations humans have when providing instructions through prompts. This process is referred to as instruction tuning [54]. While we delve into the specifics of designing and engineering prompts in later sections, it's crucial to grasp that in the context of instruction tuning, the instruction acts as a prompt that outlines the task the LLM is expected to perform. Instruction tuning datasets, such as Natural Instructions [55], encompass not only the task definition but also include other components such as positive and negative examples or instructions on what to avoid.

[3] https://platform.openai.com/docs/guides/fine-tuning

The specific methodology and instruction datasets utilized for instruction tuning an LLM may vary; however, as a general trend, instruction-tuned models tend to surpass the performance of their original foundation models upon which they are built. For instance, InstructGPT [56] exhibits superior performance to GPT-3 across most benchmarks. Similarly, Alpaca [57] outperforms LLaMA in comparative evaluations.

Self-Instruct [58], introduced by Wang et al., is another prominent method in this domain. They proposed a framework aimed at enhancing the instruction-following abilities of pre-trained language models by leveraging their own generated outputs. Their pipeline involves generating instructions, input, and output samples from a language model, followed by filtering out invalid or redundant ones before employing them to fine-tune the original model.

2.6.7 Reinforcement Learning from Human Feedback (RLHF)

AI Alignment is the process of steering AI systems towards human goals, preferences, and principles. LLMs, pre-trained for word prediction, often exhibit unintended behaviors. For example, they might generate contents that are toxic, harmful, misleading and biased.

Instruction tuning, discussed above, gets LLMs a step closer to being aligned. However, in many cases, it is important to include further steps to improve the alignment of the model and avoid unintended behaviors.

Reinforcement Learning from Human Feedback (RLHF) is a powerful technique used to fine-tune large language models (LLMs) by leveraging human-generated feedback. This approach helps address limitations of supervised fine-tuning, such as overfitting to the training data, and enables LLMs to better align with human preferences and values. This section explores the key components and process of RLHF [59].

Collecting Human Feedback

The first step in RLHF involves collecting human feedback on model-generated outputs. This can be achieved through several methods, such as:

- Providing demonstrations of correct behavior.
- Comparing different model-generated outputs and ranking them based on quality.
- Assigning explicit rewards or scores to model outputs.

The collected feedback serves as a valuable source of information for the model to learn from, helping it better understand human preferences and expectations.

Reward Modeling

Once human feedback is collected, a reward model is trained to predict the quality of the model-generated outputs. This reward model acts as a proxy for human judgment and is used to guide the reinforcement learning process. By training the reward model on human-generated feedback, it learns to approximate human preferences and assign rewards accordingly.

Reinforcement Learning

With a reward model in place, the LLM is fine-tuned using reinforcement learning algorithms such as Proximal Policy Optimization (PPO) [60] or REINFORCE [61]. The goal is to generate outputs that maximize the predicted rewards, which are a proxy for human preferences. By optimizing the model's behavior to align with the reward model, LLMs can learn to generate outputs that are more desirable and useful to human users.

Iterative Improvement

The RLHF process is iterative. As the model improves, additional human feedback can be collected to further refine the reward model. This updated reward model is then used to guide another round of reinforcement learning, resulting in continued performance improvements.

Through these iterations, LLMs can learn to better align with human values and preferences, which can lead to higher-quality outputs and improved performance on specific tasks. RLHF has been successfully applied to fine-tune models like OpenAI's ChatGPT [56], demonstrating its effectiveness in enhancing the capabilities of LLMs.

2.6.8 How LLMs are used and augmented

Once LLMs are trained, they become capable of generating desired outputs for various tasks. While LLMs can be directly utilized through basic prompting, fully exploiting their potential or overcoming some of their limitations often requires augmentation through external means. In this section, we initially provide a concise overview of the primary shortcomings of LLMs, with a particular focus on the issue of hallucination. We then discuss how prompting and certain augmentation strategies can not only mitigate these limitations but also enhance the capabilities of LLMs, potentially transforming them into comprehensive AI agents with the capacity to interact with the external world.

LLM limitations

It's crucial to acknowledge that LLMs are trained to predict tokens. Despite the enhancements brought about by fine-tuning and alignment, they still exhibit significant limitations, especially if used in a simplistic manner. Some notable constraints include:

- Lack of state/memory: LLMs inherently lack the ability to retain information from previous prompts, which poses a challenge for tasks requiring some form of state management.
- Stochastic/probabilistic nature: Repeatedly sending the same prompt to an LLM may result in different responses due to its probabilistic nature. Although parameters like temperature can mitigate variability, this trait stems from their training process and can lead to inconsistent outputs.
- Limited access to external data: LLMs operate in isolation and lack real-time information or access to data beyond their training set.
- Large resource requirements: Given their substantial size, training and serving LLMs often necessitate expensive GPU resources. Larger models may also exhibit poor service level agreements (SLAs), particularly concerning latency.
- Hallucinatory tendencies: LLMs lack a definitive concept of "truth" and have been trained on diverse content, leading to the generation of plausible yet inaccurate responses.

While all these limitations are pertinent to various applications, it's particularly insightful to delve deeper into the issue of hallucinations. This phenomenon has garnered significant attention in recent months and has catalyzed the development of numerous prompting approaches and LLM augmentation techniques, as we'll explore further.

In the domain of Large Language Models (LLMs), the concept of **"hallucinations"** has become a focal point of discussion. Defined in the literature, particularly in the paper "Survey of Hallucination in Natural Language Generation" [62], hallucination within an LLM refers to "the generation of content that is nonsensical or deviates from the provided source." Although the term originates from psychological discourse, it has been adopted and adapted within the realm of artificial intelligence.

Hallucinations in LLMs can be broadly categorized into two types:

1. **Intrinsic Hallucinations:** These directly conflict with the source material, introducing factual inaccuracies or logical inconsistencies.

2. **Extrinsic Hallucinations:** These, while not contradicting, are unverifiable against the source, encompassing speculative or unconfirmable elements.

In LLM contexts, the definition of 'source' varies depending on the task at hand. In dialogue-based tasks, 'source' typically refers to 'world knowledge,' encompassing a broad range of factual and contextual information. Conversely, in tasks like text summarization, the 'source' is more narrowly defined as the input text itself. This distinction holds significance in assessing and understanding hallucinations within LLM-generated content. Furthermore, the implications of hallucinations are highly context-dependent. For instance, in creative endeavors such as poem writing, hallucinations may be considered acceptable or even advantageous, as they can contribute to artistic expression and innovation.

LLMs, trained on diverse datasets sourced from the internet, books, and Wikipedia, generate text based on probabilistic models, lacking an intrinsic understanding of truth or falsehood. Despite recent advancements such as instruct tuning and Reinforcement Learning from Human Feedback (RLHF), which aim to guide LLMs toward producing more factually accurate outputs, the fundamental probabilistic nature and associated limitations persist. A recent study titled "Sources of Hallucination by Large Language Models on Inference Tasks" [63] sheds light on two critical factors contributing to hallucinations in LLMs: the veracity prior and the relative frequency heuristic. This underscores the intricate challenges involved in LLM training and output generation.

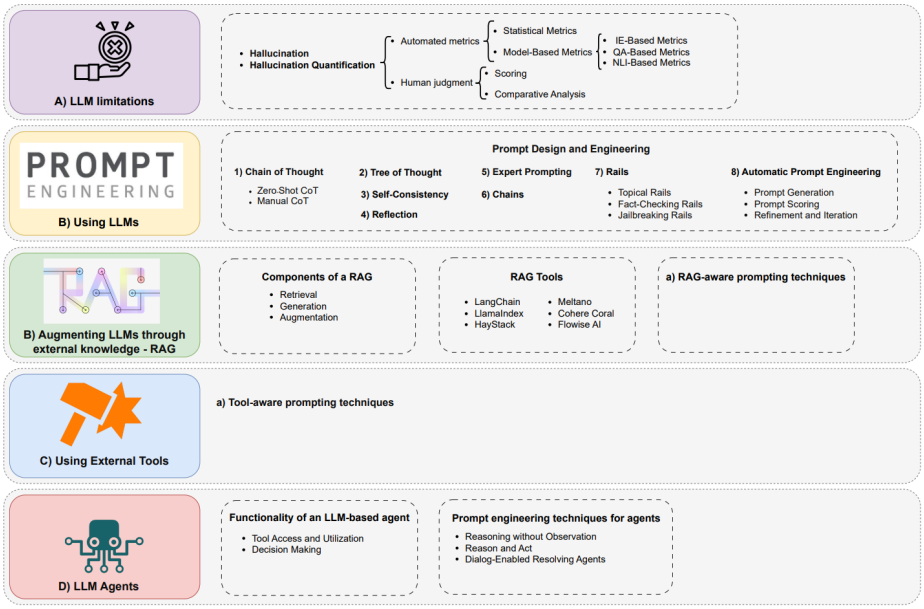


Figure 2.15: *How LLMs Are Used and Augmented [45].*

Prompt Design and Engineering

In generative AI models, a prompt refers to the textual input provided by users to steer the model's output. This input can vary from straightforward questions to detailed descriptions or specific tasks. Typically, prompts encompass instructions, questions, input data, and examples. In practice, to prompt a desired response from an AI model, a prompt must include either instructions or questions, although other elements are optional. Advanced prompts may involve more intricate structures, such as "chain of thought" prompting, where the model is directed to follow a logical reasoning process to generate an answer.

Prompt engineering is a rapidly evolving discipline that shapes the interactions and outputs of LLMs and other generative AI models. At its core, prompt engineering involves crafting optimal prompts to achieve specific goals with a generative model. This process goes beyond merely instructing the model; it also requires a deep understanding of the model's capabilities and limitations, as well as the context in which it operates.

Researchers employ prompt engineering to enhance the performance of LLMs across a broad spectrum of tasks, including question answering and arithmetic reasoning. Developers utilize prompt engineering to create robust and effective prompting techniques that interface seamlessly with LLMs and other tools.

Prompt engineering extends beyond the design and development of prompts. It encompasses a wide array of skills and techniques essential for interacting with and developing LLMs. This discipline is crucial for interfacing with LLMs, building applications with them, and understanding their capabilities. Through prompt engineering, one can enhance the safety of LLMs and develop new functionalities, such as augmenting LLMs with domain knowledge and integrating external tools.

In the following paragraphs we detail some of the most interesting and popular prompt engineering approaches.

1) Chain of Thought (CoT): The Chain of Thought (CoT) technique, first described in the paper "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models" [43] by Google researchers, marks a significant advancement in prompt engineering for Large Language Models (LLMs). This method is based on the understanding that although LLMs excel in token prediction, they are not inherently designed for explicit reasoning. CoT mitigates this limitation by guiding the model through essential reasoning steps.

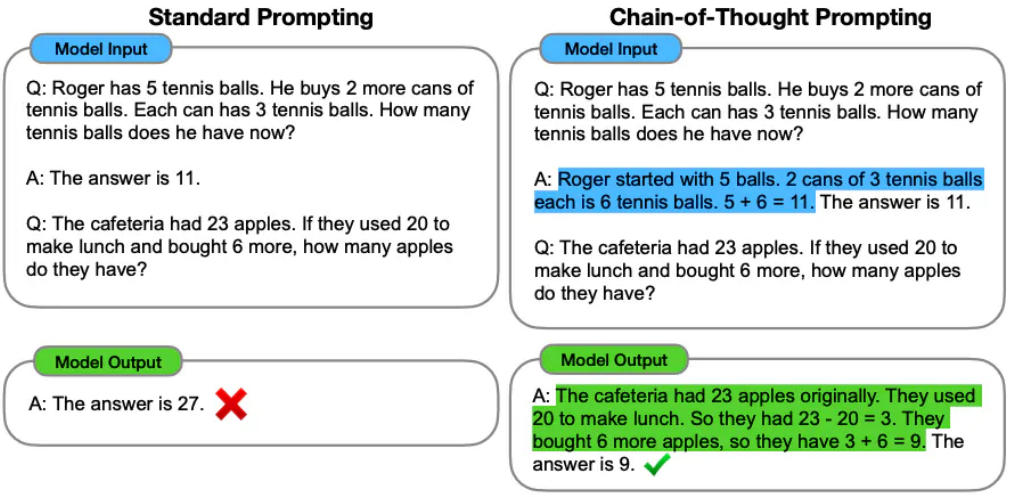


Figure 2.16: *Chain-of-Thought (CoT) Prompting [43]*

CoT is designed to make the implicit reasoning process of LLMs explicit. By outlining the necessary reasoning steps, this technique guides the model towards producing more logical and reasoned outputs, particularly in scenarios that demand more than simple information retrieval or pattern recognition.

2) Tree of Thought (ToT): The Tree of Thought (ToT) prompting technique [64] draws inspiration from the concept of exploring various alternative solutions or thought processes before settling on the most plausible one. ToT involves branching out into multiple "thought trees," with each branch representing a different line of reasoning. This method enables the LLM to investigate various possibilities and hypotheses, mirroring human cognitive processes where multiple scenarios are considered before determining the most likely outcome.

A critical aspect of ToT is the evaluation of these reasoning paths. As the LLM generates different branches of thought, each branch is assessed for its validity and relevance to the query. This process involves real-time analysis and comparison of the branches, ultimately leading to the selection of the most coherent and logical outcome.

ToT is particularly useful in complex problem-solving scenarios where a single line of reasoning might not suffice. It enables LLMs to mimic a more human-like problem-solving approach by considering a range of possibilities before arriving at a conclusion. This technique enhances the model's ability to handle ambiguity, complexity, and nuanced tasks, making it a valuable tool in advanced AI applications.

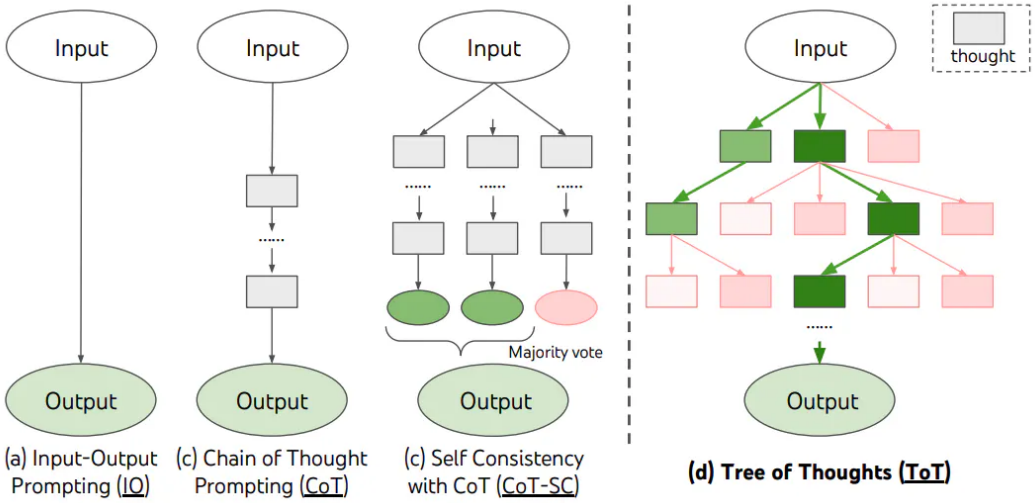


Figure 2.17: *Tree-of-Thought (ToT) Framework [64]*

3) Self-Consistency: Self-Consistency [65] employs an ensemble-based method, prompting the LLM to generate multiple responses to the same query. The consistency among these responses acts as an indicator of their accuracy and reliability.

The Self-Consistency approach is based on the principle that if an LLM generates multiple similar responses to the same prompt, the response is more likely to be accurate. This method involves prompting the LLM to address a query multiple times and then analyzing the responses for consistency. This technique is particularly useful in scenarios where factual accuracy and precision are critical.

The consistency of responses can be measured using various methods. One common approach is to analyze the overlap in the content of the responses. Other methods include comparing the semantic similarity of responses or employing more sophisticated techniques like BERT-scores or n-gram overlaps. These measures help quantify the level of agreement among the responses generated by the LLM.

4) Reflexion: Reflexion [66] involves prompting LLMs to assess and potentially revise their own outputs by reasoning about the correctness and coherence of their responses. The concept of Reflexion centers on the ability of LLMs to engage in a form of self-evaluation. After generating an initial response, the model is prompted to reflect on its own output, considering factors such as factual accuracy, logical consistency, and relevance. This introspective process can lead to the generation of revised or improved responses.

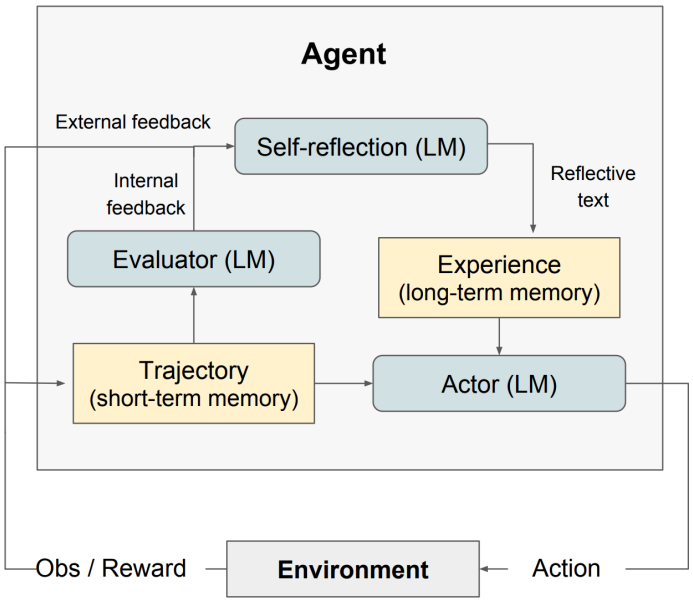


Figure 2.18: *Diagram of Reflection [66].*

A crucial component of Reflexion is the LLM's ability to self-edit. By assessing its initial response, the model can detect potential errors or areas that need improvement. This iterative process of generating, reflecting, and revising enables the LLM to enhance its output, improving the overall quality and reliability of its responses.

5) Chains: Chains refer to the method of linking multiple components in a sequence to manage complex tasks with Large Language Models (LLMs). This approach involves creating a series of interconnected steps or processes, each contributing to the final outcome. The concept of Chains is based on constructing a workflow where different stages or components are sequentially arranged. Each component in a Chain performs a specific function, with the output of one serving as the input for the next. This end-to-end arrangement allows for more intricate and nuanced processing, as each stage can be customized to handle a specific aspect of the task. The complexity and structure of Chains can vary based on the requirements. In "PromptChainer: Chaining Large Language Model Prompts through Visual Programming" [67], the authors discuss the main challenges in designing chains and introduce a visual tool to support these tasks.

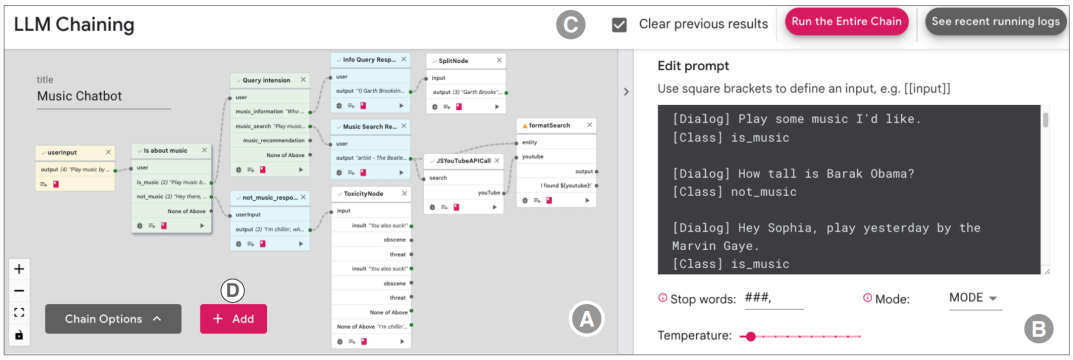


Figure 2.19: *The PromptChainer interface. (A) The Chain View visualizes the chain structure with node-edge diagrams (enlarged in Figure 2), and allows users to edit the chain by adding, removing, or reconnecting nodes. (B) The Node View supports implementing, improving, and testing each individual node, e.g., editing prompts for LLM nodes. PromptChainer also supports running the chain end-to-end (C) [67].*

6) Automatic Prompt Engineering (APE): Automatic Prompt Engineering (APE) [68] aims to automate the creation of prompts for Large Language Models (LLMs). APE seeks to streamline and optimize the prompt design process by leveraging the capabilities of LLMs themselves to generate and evaluate prompts. This approach involves using LLMs in a self-referential manner, where the model generates, scores, and refines prompts. By recursively utilizing LLMs in this way, APE facilitates the creation of high-quality prompts that are more likely to elicit the desired response or outcome.

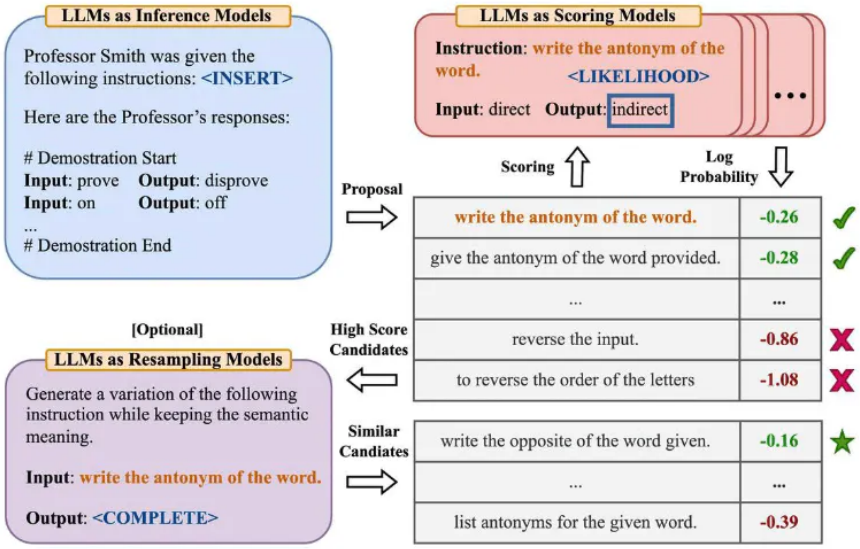


Figure 2.20: *Automatic Prompt Engineer framework [68].*

Augmenting LLMs through external knowledge - RAG

One of the primary limitations of pre-trained Large Language Models (LLMs) is their lack of up-to-date knowledge or access to private or use case-specific information. This is where Retrieval Augmented Generation (RAG) comes into the picture [69]. As illustrated in Figure 2.21, RAG involves extracting a query from the input prompt and using that query to retrieve relevant information from an external knowledge source (e.g., a search engine or a knowledge graph.). The relevant information is then added to the original prompt and fed to the LLM to generate the final response. A RAG system includes three important components: Retrieval, Generation, and Augmentation [2].

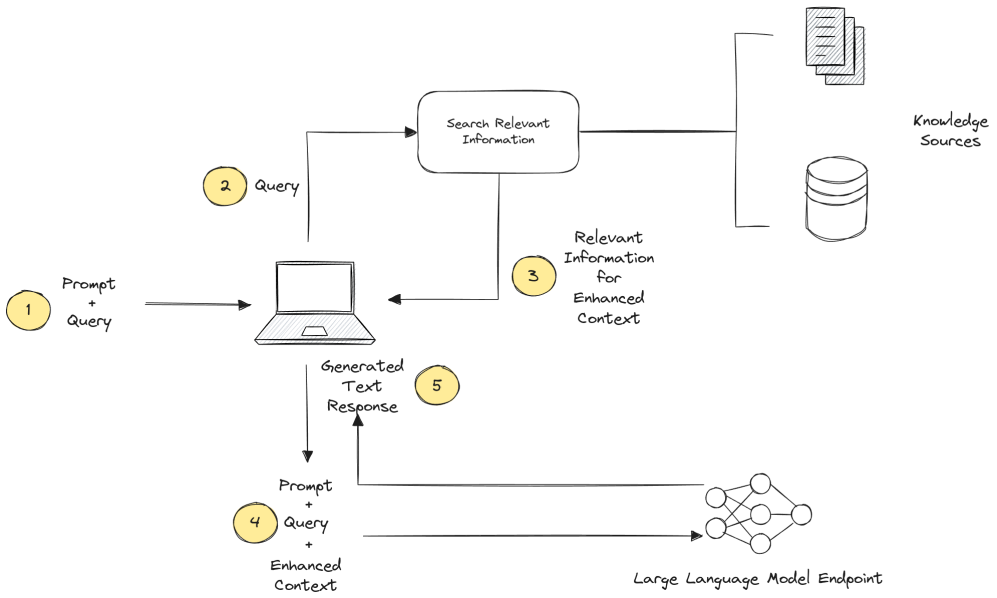


Figure 2.21: *An example of synthesizing RAG with LLMs for question answering application.*

Due to the significance of RAG in the development of advanced LLM systems, several RAG-aware prompting techniques have been developed recently. One such technique is Forward-looking Active Retrieval Augmented Generation (FLARE).

Forward-looking Active Retrieval Augmented Generation (FLARE) [70] enhances the capabilities of Large Language Models (LLMs) by iteratively integrating prediction and information retrieval. FLARE represents an advancement in retrieval-augmented generation, aimed at improving the accuracy and relevance of LLM responses.

FLARE involves an iterative process where the LLM actively predicts upcoming content and uses these predictions as queries to retrieve relevant information. This method differs from traditional retrieval-augmented models, which typically re-

trieve information once before proceeding with generation. In FLARE, the process is dynamic and ongoing throughout the generation phase. Each sentence or segment generated by the LLM is evaluated for confidence. If the confidence level falls below a certain threshold, the model uses the generated content as a query to retrieve relevant information, which is then used to regenerate or refine the sentence. This iterative process ensures that each part of the response is informed by the most relevant and current information available.

Augmenting LLMs through external tools

Retrieving information from an external knowledge source, as previously described, is just one way to enhance an LLM. More broadly, an LLM can interface with various external tools (e.g., an API to a service) to extend its capabilities. In this context, RAG (Retrieval-Augmented Generation) can be viewed as a specific example within the broader category of these "tools."

In this context, tools refer to external functions or services that LLMs can utilize. These tools enhance the capabilities of LLMs, enabling them to perform a broader range of tasks, from basic information retrieval to complex interactions with external databases or APIs.

In the paper "Toolformer: Language Models Can Teach Themselves to Use Tools" [71], the authors advance beyond basic tool usage by training an LLM to determine which tool to use and what parameters the API requires. These tools include two different search engines and a calculator. In the examples provided, the LLM autonomously decides to use an external Q&A tool, a calculator, and a Wikipedia search engine. More recently, researchers at Berkeley developed a new LLM called Gorilla [72], which surpasses GPT-4 in API utilization, a specific but broadly applicable tool.

2.7 Retrieval Augmented Generation (RAG)

2.7.1 Introduction

Large language models (LLMs) have demonstrated significant success, yet they continue to encounter substantial limitations, particularly in tasks that are domain-specific or knowledge-intensive [73]. A prominent issue is the generation of "hallucinations" [1], where LLMs produce responses to queries that exceed their training data or necessitate up-to-date information. To mitigate these challenges, Retrieval-Augmented Generation (RAG) improves LLM performance by retrieving pertinent document chunks from an external knowledge base through semantic similarity calculations. By incorporating external knowledge, RAG substantially reduces the likelihood of generating factually incorrect content. The integration of RAG into LLMs has become widespread, establishing it as a crucial technology for advancing chatbots and enhancing the practical applicability of LLMs in real-world scenarios.

To address the limitations of generative AI, researchers and engineers have developed innovative methods, including the Retrieval-Augmented Generation (RAG) approach. RAG gained significant attention among generative AI developers following the release of the seminal paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" by Lewis et al. (2020) at Facebook AI Research [69]. RAG enhances the quality and relevance of generated text by combining the strengths of generative AI with retrieval techniques. Unlike traditional generative models that depend solely on their internal knowledge, RAG incorporates an additional step of retrieving information from external sources, such as databases, documents, or the web, before generating a response. This integration enables RAG to access up-to-date information and context, making it especially valuable for applications requiring accurate and current information.

RAG technology has seen rapid development in recent years, as illustrated by the technology tree summarizing related research in Figure 2.22. The evolution of RAG within the era of large models can be divided into several distinct stages. Initially, the inception of RAG coincided with the rise of the Transformer architecture, focusing on improving language models by integrating additional knowledge through Pre-Training Models (PTM). This early phase was marked by foundational efforts to refine pre-training techniques [74]-[75]. The subsequent emergence of ChatGPT [56] represented a pivotal moment, showcasing powerful in-context learning (ICL) capabilities. Following this, RAG research shifted towards enhancing LLMs' ability to address more complex and knowledge-intensive tasks during the inference stage, spurring rapid advancements in RAG studies. Over time, the focus expanded beyond the inference stage to include fine-tuning techniques for LLMs, further enhancing RAG's capabilities.

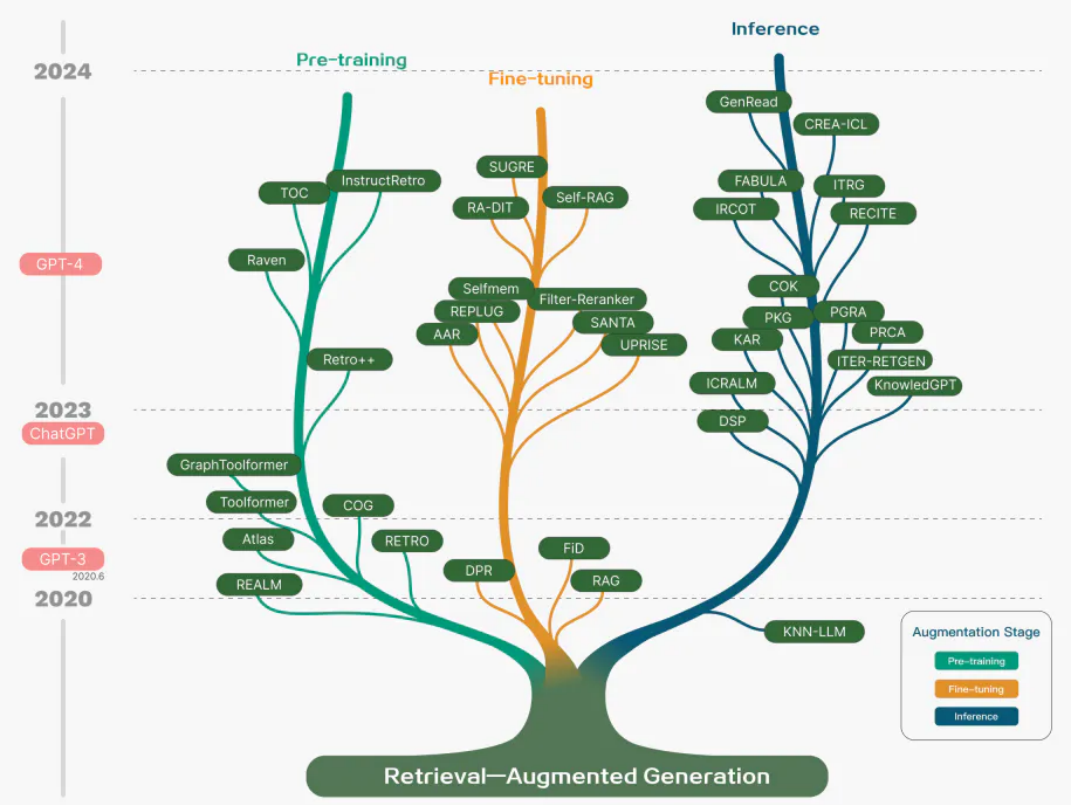


Figure 2.22: *Technology tree of RAG research [2]*

2.7.2 Retrieval Augmented Generation (RAG) Categories

The RAG research paradigm is continuously evolving and can be categorized into three stages: Naive RAG, Advanced RAG, and Modular RAG, as shown in Figure 2.23. Although RAG methods are cost-effective and outperform native LLMs, they also present several limitations. The development of Advanced RAG and Modular RAG addresses these specific shortcomings found in Naive RAG.

Naive RAG

The Naive RAG research paradigm represents the earliest methodology, gaining prominence soon after the widespread adoption of ChatGPT. Naive RAG follows a traditional process encompassing indexing, retrieval, and generation, often described as a "Retrieve-Read" framework [76].

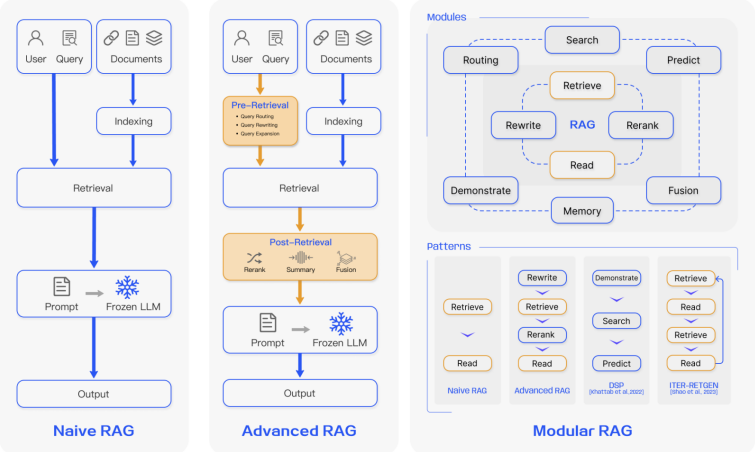


Figure 2.23: *Comparison between the three paradigms of RAG [2].*

Indexing starts with cleaning and extracting raw data from diverse formats like PDF, HTML, Word, and Markdown, converting it into a uniform plain text format. To address the context limitations of language models, the text is broken down into smaller, manageable chunks. These chunks are then encoded into vector representations using an embedding model and stored in a vector database. This step is critical for enabling efficient similarity searches in the next retrieval phase.

Upon receiving a user query, the RAG system employs the same encoding model used during the indexing phase to convert the query into a vector representation. It then calculates similarity scores between the query vector and the vectors of the chunks in the indexed corpus. The system prioritizes and retrieves the top K chunks that show the highest similarity to the query. These chunks are then used as the expanded context in the prompt for the generation phase.

During the generation phase, the posed query and the selected documents are combined into a coherent prompt. A large language model then formulates a response based on this prompt. The model's approach to answering can vary depending on task-specific criteria, allowing it to either utilize its inherent parametric knowledge or restrict its responses to the information within the provided documents. In cases of ongoing dialogues, any existing conversational history can be incorporated into the prompt, enabling the model to effectively engage in multi-turn interactions.

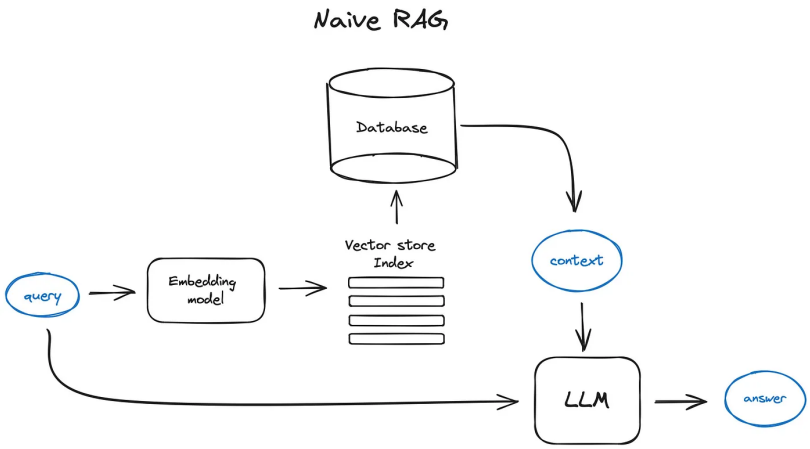


Figure 2.24: *Naive RAG*

Advanced RAG

Advanced RAG introduces specific enhancements to address the limitations of Naive RAG. It focuses on improving retrieval quality through both pre-retrieval and post-retrieval strategies. To resolve indexing issues, Advanced RAG employs refined indexing techniques such as a sliding window approach, fine-grained segmentation, and the integration of metadata. Additionally, it incorporates various optimization methods to streamline the retrieval process.

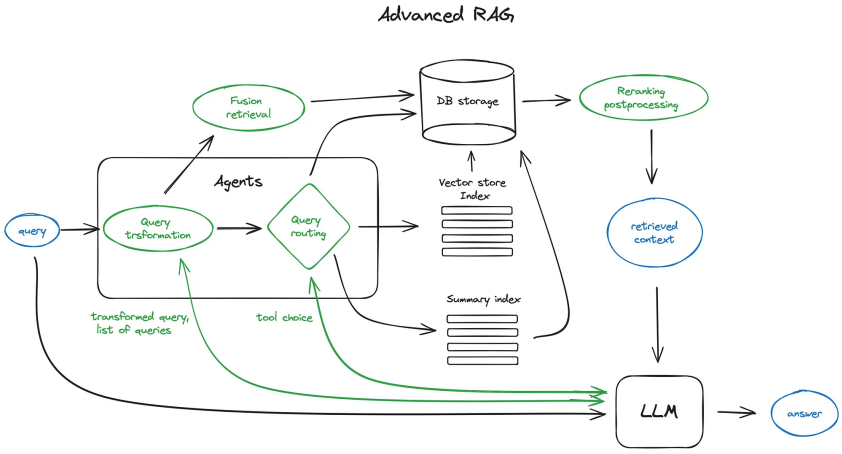


Figure 2.25: *Advanced RAG*

Pre-retrieval process. In this stage, the primary focus is on optimizing both the indexing structure and the original query. The objective of indexing optimization is to enhance the quality of the content being indexed. This involves several strategies: increasing data granularity, optimizing index structures, incorporating metadata, alignment optimization, and mixed retrieval. The goal of query optimization is

to make the user's original question clearer and more suitable for retrieval. Common methods include query rewriting, query transformation, query expansion, and other techniques [76], [77] [78].

Post-Retrieval Process. Once relevant context is retrieved, it is crucial to integrate it effectively with the query. The main methods in the post-retrieval process include reranking chunks and compressing the context. Re-ranking involves adjusting the order of the retrieved information to ensure the most relevant content is positioned at the forefront of the prompt. This strategy is implemented in frameworks such as LlamaIndex[4], LangChain[5], and HayStack[6]. Directly feeding all relevant documents into LLMs can cause information overload, diluting the focus on key details with irrelevant content. To mitigate this, post-retrieval efforts concentrate on selecting essential information, highlighting critical sections, and reducing the context to a manageable size for processing.

Modular RAG

The modular RAG architecture advances beyond the earlier RAG paradigms, providing enhanced adaptability and versatility. It incorporates various strategies to improve its components, such as adding a search module for similarity searches and fine-tuning the retriever. Innovations like restructured RAG modules [79] and rearranged RAG pipelines [80] have been developed to address specific challenges. The trend towards a modular RAG approach is becoming increasingly common, supporting both sequential processing and integrated end-to-end training across its components. Despite its distinctiveness, Modular RAG builds on the foundational principles of Advanced and Naive RAG, demonstrating a progression and refinement within the RAG family.

2.7.3 Retrieval Models for Document Search

Selecting the appropriate retrieval model is critical for ensuring efficient and accurate document search and ranking in response to user queries. Here are some key retrieval models and considerations:

TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF is a classical retrieval model that calculates the importance of terms within a document relative to a corpus. It is straightforward to implement and particularly effective for certain tasks, especially within smaller or less complex datasets. This model is suitable for straightforward keyword-based search tasks, offering a simple yet effective approach for basic retrieval needs.

[4] https://www.llamaindex.ai/
[5] https://www.langchain.com/
[6] https://haystack.deepset.ai/

BM25: BM25 builds upon TF-IDF by accounting for document length and term saturation, making it an improved version of its predecessor. It handles term frequency variations and document length normalization better, often resulting in more effective retrieval for modern tasks. BM25 is ideal for general-purpose search engines and more sophisticated retrieval needs, providing a robust solution for a variety of applications.

Vector Space Models: These models represent documents and queries as vectors in a high-dimensional space, using metrics like cosine similarity for ranking. Implementations such as Latent Semantic Analysis (LSA) and Word Embeddings (e.g., Word2Vec) can capture semantic similarities and relationships between terms. This makes vector space models effective for tasks that require understanding nuanced meanings and context within documents.

Neural Ranking Models: Modern neural models, such as those based on BERT, capture complex semantic relationships and provide deep understanding of context and nuances. These models can be fine-tuned for specific tasks and domains, making them highly effective for advanced retrieval needs. Neural ranking models are particularly suitable for tasks requiring sophisticated semantic comprehension and detailed query resolution.

Hybrid Models: Hybrid models combine multiple retrieval approaches, such as TF-IDF with neural models, leveraging the strengths of each method. By balancing simplicity and effectiveness, hybrid models offer robustness across various scenarios, making them useful for comprehensive retrieval systems that need to perform well in a range of conditions.

2.7.4 Embeddings and Vector Databases for Retrieval in RAG

In addition to selecting an appropriate retrieval model, leveraging embeddings and vector databases can significantly enhance the performance and efficiency of the retrieval component. Vector embeddings are a fundamental concept in modern information retrieval and natural language processing. They transform textual data into numerical vectors, enabling computers to understand and manipulate text data within a mathematical, geometric space. These embeddings capture semantic and contextual relationships between words, documents, and other textual entities, making them highly valuable in various applications, including the retrieval component of Retrieval-Augmented Generation (RAG). By employing embeddings and vector databases, the retrieval process becomes more efficient and effective, providing more accurate and contextually relevant results.

Vector Embeddings

Vector embeddings represent words, phrases, sentences, or even entire documents as points in a high-dimensional vector space. The key idea is to map each textual element into a vector in such a way that semantically similar elements are located close to each other in this space, while dissimilar elements are further apart. This geometric representation facilitates similarity calculations, clustering, and other operations [81].

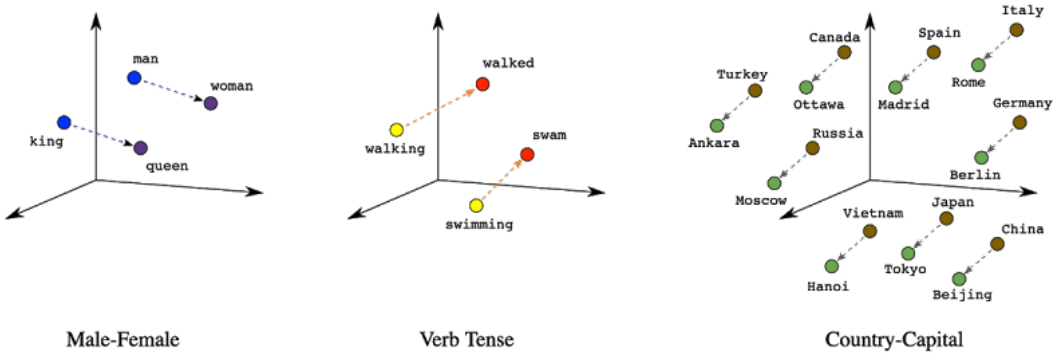


Figure 2.26: *Words plotted in 3-dimensional space. Embeddings can have hundreds or thousands of dimensions-too many for humans to visualize [82].*

Examples of Vector Embeddings:

- **Word Embeddings (Word2Vec, GloVe):** Word embeddings represent individual words as vectors. For example, "king" and "queen" may be represented as vectors that are close together in the vector space because they share similar semantic properties.
- **Document Embeddings (Doc2Vec, BERT):** Document embeddings map entire documents (such as PDFs) into vectors. Two documents discussing similar topics will have embeddings that are close in the vector space.

Vector Databases and Their Role in Enhancing Retrieval

Vector databases, also known as similarity search engines or vector index databases, play a crucial role in the retrieval component of RAG by efficiently storing and retrieving these vector embeddings. They are specialized databases designed for retrieving vectors based on similarity, making them well-suited for scenarios where similarity between data points needs to be calculated quickly and accurately.

How Vector Databases Enhance Retrieval in RAG:

Fast Retrieval: Vector databases utilize indexing structures specifically optimized for similarity searches. They employ algorithms such as approximate nearest neighbor (ANN) search to quickly identify the most similar vectors, even within large datasets containing numerous documents.

Scalability: Vector databases are designed to efficiently scale as the document corpus expands. This ensures that retrieval performance remains consistent, regardless of the dataset's size, maintaining high efficiency and speed.

Advanced Similarity Scoring: These databases provide a variety of similarity metrics, including cosine similarity and the Jaccard index. This allows for fine-tuning the relevance ranking of retrieved documents based on specific requirements, enhancing the precision of search results.

Integration with Retrieval Models: Vector databases can be seamlessly integrated into existing retrieval systems. They complement retrieval models like TF-IDF, BM25, and neural ranking models by offering an efficient method for selecting candidate documents based on vector similarity, thereby improving the overall retrieval process.

These factors have led to the emergence of numerous new vector databases. Choosing one of these databases can create long-term dependencies and have significant impacts on your system. Ideally, a vector database should exhibit strong scalability while maintaining cost-efficiency and minimizing latency. Some of these vector databases include Qdrant[7], Weaviate[8], Pinecone[9], pgvector[10], Milvus[11], and Chroma[12].

2.7.5 Challenges of Retrieval-Augmented Generation

The adoption of Retrieval-Augmented Generation (RAG) marks a significant advancement in natural language processing and information retrieval. However, like any complex AI system, RAG presents a set of challenges that must be addressed to fully harness its potential. This section explores some of the key challenges associated with RAG.

Data Quality and Relevance

RAG heavily relies on the availability of high-quality and relevant data for both retrieval and generation tasks. Challenges in this area include:

[7] https://qdrant.tech/
[8] https://weaviate.io/
[9] https://www.pinecone.io/
[10] https://github.com/pgvector/pgvector
[11] https://milvus.io/
[12] https://www.trychroma.com/

- **Noisy Data:** Incomplete, outdated, or inaccurate data sources can lead to the retrieval of irrelevant information, negatively impacting the quality of generated responses.
- **Bias and Fairness:** Biases present in training data may result in biased retrieval and generation, perpetuating stereotypes or misinformation.

Integration Complexity

Integrating retrieval and generation components seamlessly is a complex task, as it involves bridging different architectures and models. Challenges include:

- **Model Compatibility**: Ensuring that the retrieval and generation models work harmoniously, particularly when combining traditional methods (e.g., TF-IDF) with neural models (e.g., GPT-3).
- **Latency and Efficiency:** Balancing the need for real-time responsiveness with the computational resources required for both retrieval and generation.

Scalability

Scaling RAG systems to handle large volumes of data and user requests presents several challenges:

- **Indexing Efficiency:** As the document corpus grows, maintaining an efficient and up-to-date index becomes crucial for ensuring retrieval speed.
- **Model Scaling:** Deploying large-scale neural models for both retrieval and generation demands substantial computational resources.

Domain Adaptation

Adapting RAG systems to specific domains or industries can be complex:

- **Domain-Specific Knowledge:** Incorporating industry-specific knowledge and terminology into retrieval and generation.
- **Training Data Availability:** Ensuring the availability of domain-specific training data for fine-tuning models.

Chapter 3

Proposed Solution

In this chapter, we delve into the implementation details of our proposed solution, designed to enhance the efficiency and accuracy of information retrieval through a Local Retrieval-Augmented Generation (RAG) agent. Our solution leverages a variety of advanced technologies, including FastAPI[1], Langchain[2], Langraph[3], Ollama[4], Docker, Docker Compose[5], and LangSmith[6]. By integrating these tools, our aim is to create a robust, scalable, and efficient system capable of serving both local and web-based information needs.

Our implementation focuses exclusively on using LLaMA3 for the language model component. This approach ensures that we can thoroughly evaluate and optimize the performance of LLaMA3 in the context of our application.

We will begin by discussing the high-level architecture of our solution, highlighting the key components and their interactions. This will be followed by an overview of the backend implementation, which leverages FastAPI, Langchain, and Langraph for efficient document retrieval and model serving, complemented by Ollama for local model serving. We will also cover the use of Docker and Docker Compose for containerization and deployment, ensuring the scalability and reliability of our backend services. Additionally, we will discuss the integration of LangSmith for tracing and logging LLM outputs to monitor and improve system performance.

Throughout this chapter, we will provide a comprehensive overview of the data flow, from the initial user query to the final generated answer, elucidating the intricacies of our system's workflow. Finally, we will address the challenges encountered and the solutions devised to overcome them.

3.1 Problem Definition

In the era of rapid information growth and increasing complexity of queries, traditional question-answering systems often struggle to provide comprehensive and accurate responses. These systems frequently rely on a single approach, such as keyword-

[1] https://fastapi.tiangolo.com/
[2] https://python.langchain.com/v0.1/docs/get_started/introduction
[3] https://python.langchain.com/v0.1/docs/langgraph/
[4] https://www.ollama.com/
[5] https://www.docker.com/
[6] https://www.langchain.com/langsmith

based retrieval or language model generation, which can lead to suboptimal results. Furthermore, many existing systems are limited to a predefined knowledge base or corpus, restricting their ability to adapt and incorporate the latest information from the vast expanse of the internet.

Traditional question-answering approaches may also suffer from issues such as hallucinations [1] or failing to address the core aspects of a query. There is a need for a more advanced and adaptive question-answering system that can leverage the strengths of various retrieval and generation techniques, while also accounting for potential limitations or errors.

The goal of our project is to develop a Local RAG (Retrieval-Augmented Generation) agent with LLaMA3 and other open source LLMs that combines ideas from several state-of-the-art RAG papers. This agent aims to provide more accurate and comprehensive answers to a wide range of queries by intelligently routing queries, utilizing fallback mechanisms, and incorporating self-correction capabilities. The Local RAG agent will leverage the power of large language models like LLaMA3, while also incorporating retrieval techniques to ground the responses in relevant information sources. Additionally, the agent will have the ability to fall back to web search when the available information is insufficient, and self-correct its responses to address potential hallucinations or incomplete answers.

By combining these different approaches, the Local RAG agent aims to overcome the limitations of traditional question-answering systems, providing users with more reliable, up-to-date, and context-aware responses.

3.2 Solution Overview

In this section, we will thoroughly examine the overall architecture of the solution, detailing its various components and fundamental elements, and emphasizing the function of each. Figure 3.1 presents an illustration of the solution's architecture.

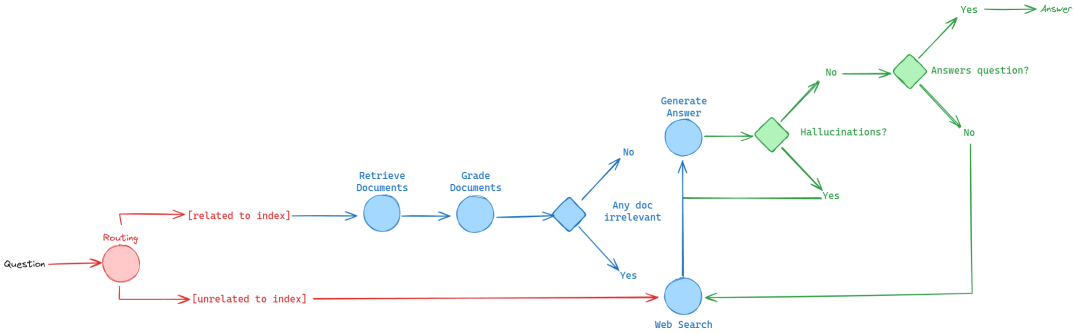


Figure 3.1: *Proposed solution architecture.*

3.2.1 High-Level Architecture

Our solution aims to develop a Local Retrieval-Augmented Generation (RAG) agent using LLaMA3. The architecture is designed to efficiently manage the entire process of information retrieval and answer generation. Below, we outline the primary components and their interactions:

1. **User Query Input:** Users submit queries to the system through an API endpoint.
2. **Routing:** The system routes the incoming queries to appropriate retrieval approaches using a routing mechanism. Queries that are relevant to the indexed documents are processed through the local retrieval system. Queries that are not relevant to the indexed documents are routed to a fallback mechanism for web search.
3. **Document Retrieval and Grading:** The retrieval system fetches relevant documents from the index. Retrieved documents are graded based on relevance and quality using Langchain and Langraph.
4. **Model Serving with Ollama:** LLaMA3 models are served locally using Ollama. The system uses these models to generate answers based on the retrieved and graded documents.
5. **Answer Generation and Validation:** Generated answers are validated for relevance and accuracy. The system checks for hallucinations and ensures the answer addresses the user's query.
6. **Fallback to Web Search:** If the retrieved documents are not relevant, the system falls back to a web search. This ensures that users receive comprehensive answers even if the local index does not have relevant information.
7. **Self-Correction and Final Answer:** The system self-corrects any answers with hallucinations or that do not address the question properly. The final validated answer is returned to the user.
8. **Logging and Monitoring with LangSmith:** LangSmith is integrated to trace and log the outputs of the LLM. This allows monitoring and improving the performance of the system.

3.2.2 Key Technologies and Frameworks

- **FastAPI:** Chosen for its high performance and ease of use in building robust APIs. Facilitates the development of a scalable and efficient backend system.

- **Langchain and Langraph:** Used for document retrieval and grading. These frameworks help in efficiently managing and processing large volumes of text data.
- **ChromaDB:** Utilized as the vector database for efficient document retrieval. Enhances the retrieval process by providing fast and accurate vector search capabilities.
- **Ollama:** Responsible for serving LLaMA3 models locally. Ensures efficient model management and minimizes latency.
- **Docker and Docker Compose:** Used for containerizing backend services. Ensures consistency across different environments and simplifies deployment.
- **LangSmith:** Integrated for tracing and logging LLM outputs. Enhances monitoring capabilities and aids in performance optimization.

3.2.3 Data Flow

1. **Query Reception:** The user submits a query to the FastAPI endpoint. The query is routed based on its relevance to the indexed documents.
2. **Document Processing:** Relevant documents are retrieved from ChromaDB and graded. Irrelevant queries are redirected to a fallback web search.
3. **Model Interaction:** The system interacts with LLaMA3 models through Ollama to generate answers. LangSmith monitors and logs these interactions for performance tracking.
4. **Answer Validation:** Generated answers are validated and corrected if necessary. The final answer is then delivered to the user.

3.3 Detailed Workflow

The workflow of our Local Retrieval-Augmented Generation (RAG) agent involves several key steps, from the initial query submission to the final generation and validation of answers. Here, we delve into each step to provide a comprehensive understanding of the process.

3.3.1 Question Routing

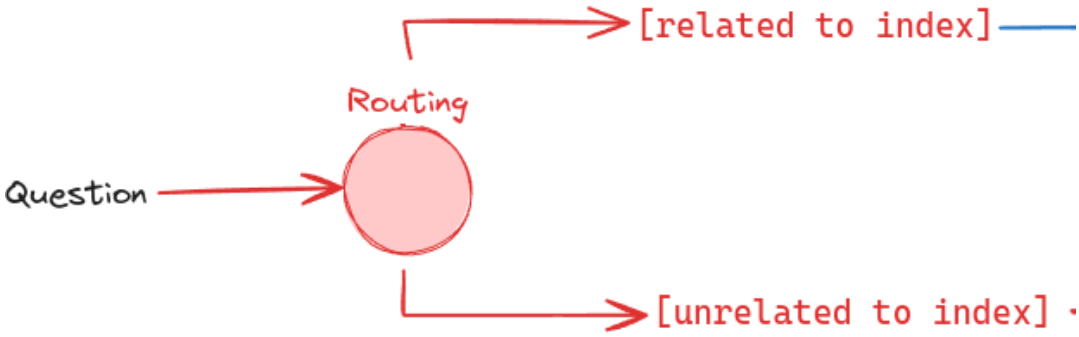


Figure 3.2: *Question Routing*

The workflow begins with the user submitting a query via a FastAPI endpoint. This endpoint is designed to handle incoming requests efficiently and forward them to the appropriate processing unit. Upon receiving a query, the system evaluates its relevance to the indexed documents. This evaluation determines whether the query can be answered using the locally stored documents or if it needs to be redirected to a fallback mechanism that utilizes web search.

For queries that are relevant to the indexed content, the system routes them to the local retrieval system. If the query is deemed unrelated to the indexed content, the system redirects it to the fallback mechanism, ensuring that users receive comprehensive answers even if the local index does not contain relevant information.

3.3.2 Document Retrieval and Grading

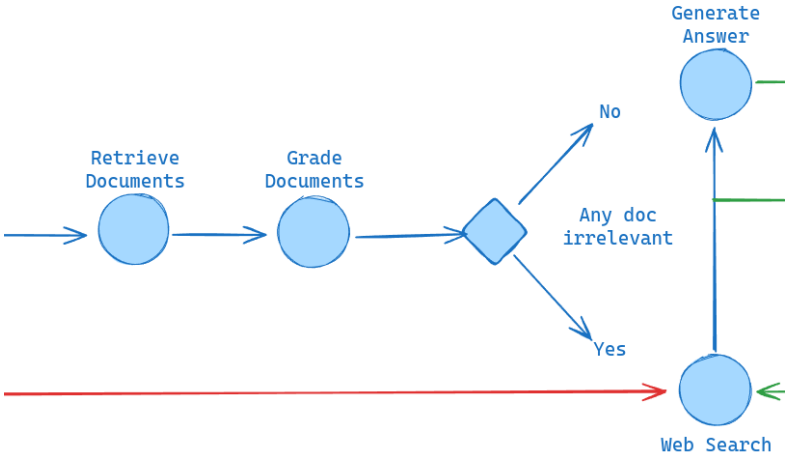


Figure 3.3: *Document Retrieval and Grading*

For queries routed to the local retrieval system, the next step is to fetch relevant documents from the ChromaDB vector database. ChromaDB employs vector search techniques to find the most relevant documents based on the query vector. This process involves converting the query into a vector representation and searching the database for documents with similar vector representations.

Once the relevant documents are retrieved, they undergo a grading process using the LLaMA3 model. The model evaluates the relevance and quality of the retrieved documents, ensuring that only the most pertinent documents are used for answer generation. This grading process is essential to maintain the quality and relevance of the information provided to the user.

If any document is found irrelevant during the grading process, the system flags it and prepares to use the fallback web search mechanism. This step is crucial to maintain the quality and relevance of the information provided to the user.

3.3.3 Answer Validation and Correction

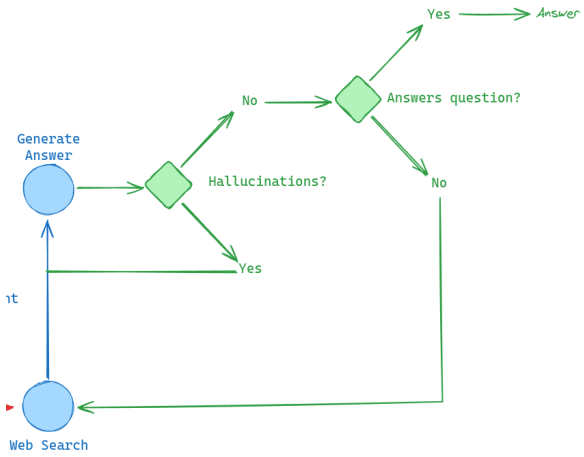


Figure 3.4: *Answer validation and correction.*

Once an answer is generated, it undergoes an initial validation process to check for relevance and accuracy. The system evaluates whether the answer sufficiently addresses the user's query and ensures it is free from hallucinations—instances where the model generates incorrect or nonsensical information.

If hallucinations or other issues are detected, the answer is flagged for further correction. The system includes a self-correction mechanism that refines answers with detected issues. This mechanism leverages additional passes through the model and grading processes to improve the answer quality. If the initial validation fails or if the retrieved documents were not relevant, the system employs the fallback mechanism to perform a web search. The results from the web search are then processed and used to generate a new answer.

After any necessary corrections, the answer undergoes a final validation step to ensure it is accurate, relevant, and free from hallucinations. This step is crucial to maintain the integrity and reliability of the system's responses.

3.3.4 Final Answer Delivery

Once the answer has been validated and corrected, it is ready for delivery. The final validated answer is sent back to the user through the FastAPI endpoint. This endpoint handles the response and ensures that the user receives a comprehensive and accurate answer to their query.

The user interaction ends here, but the system continues to monitor and log the entire process to improve future responses. By maintaining a high standard of validation and correction, the system ensures that users consistently receive high-quality information.

3.3.5 Logging and Monitoring with LangSmith

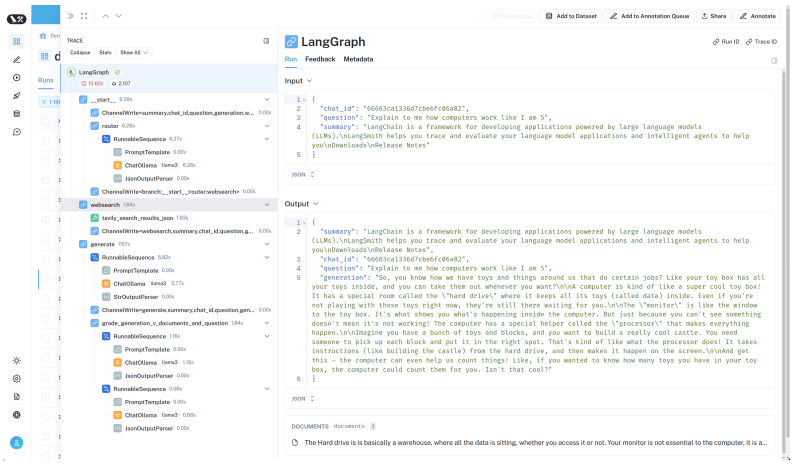


Figure 3.5: *Logging and Monitoring with LangSmith.*

Throughout the entire workflow, LangSmith is used to trace and log the outputs of the LLM. This includes logging the initial query, the retrieved documents, the generated answers, and any corrections made. LangSmith provides tools to monitor the performance of the system, identifying bottlenecks and areas for improvement.

This continuous monitoring and logging are crucial for maintaining the system's performance and reliability. By analyzing the logs, we can pinpoint issues and optimize the system to handle future queries more efficiently. LangSmith's integration enhances the system's ability to provide accurate and relevant responses while also allowing for continuous improvement.

3.3.6 Model Interaction with Ollama

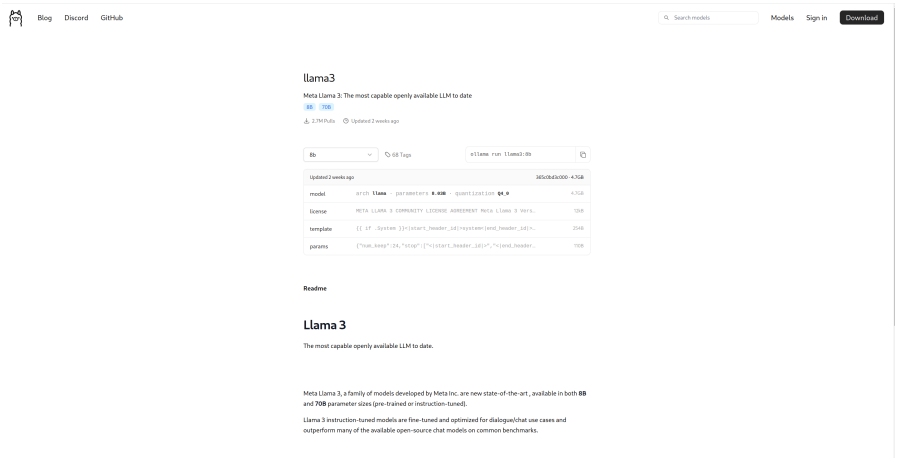


Figure 3.6: *Llama3's page on ollama website.*

With the relevant and graded documents in hand, the system then interacts with the LLaMA3 models served locally using Ollama. Ollama ensures low-latency and efficient model interaction by managing the lifecycle of the models and ensuring they are readily available for processing incoming queries.

The LLaMA3 model takes the query and the graded documents as input to generate a coherent and relevant answer. This model interaction is critical as it synthesizes the information from the documents and produces a response tailored to the user's query. By using locally served models, the system minimizes latency and enhances performance, providing users with quick and accurate responses.

3.4 Challenges and Solutions

Developing and implementing a Local Retrieval-Augmented Generation (RAG) agent using LLaMA3 involves several challenges, from integrating various technologies to ensuring the system's performance and reliability. Below, we discuss some of the key challenges encountered during the development process and the solutions implemented to address them.

3.4.1 Integration of Diverse Technologies

Challenge: Integrating multiple technologies such as FastAPI, Langchain, ChromaDB, Ollama, and LangSmith can be complex and time-consuming. Ensuring that these components work seamlessly together is critical for the overall performance of the system.

Solution: To address this challenge, a modular architecture was adopted, where each component is designed as an independent module with well-defined interfaces. Docker and Docker Compose were used to containerize each service, ensuring consistency across different environments. This modular approach simplifies the integration process and makes it easier to isolate and troubleshoot issues. Comprehensive integration testing was conducted to ensure that all components work together as expected.

3.4.2 Efficient Document Retrieval

Challenge: Efficiently retrieving relevant documents from a large dataset using vector search can be computationally intensive and time-consuming, potentially leading to latency issues.

Solution: ChromaDB was chosen as the vector database due to its optimized vector search capabilities. The system leverages advanced indexing and search algorithms to quickly retrieve the most relevant documents.

3.4.3 Document Grading Accuracy

Challenge: Ensuring the accuracy of document grading is crucial for generating high-quality answers. Incorrectly graded documents can lead to irrelevant or inaccurate answers.

Solution: The LLaMA3 model was utilized for both document grading and answer generation. By employing prompting techniques, the LLaMA3 model evaluates the relevance and quality of the retrieved documents, ensuring that only the most pertinent documents are used for answer generation. This approach maintains consistency and leverages the model's inherent capabilities without the need for fine-tuning.

3.4.4 Monitoring and Logging

Challenge: Monitoring the performance and behavior of the system in real-time, and logging all interactions to identify and troubleshoot issues, is crucial for maintaining reliability.

Solution: LangSmith was integrated into the system to provide comprehensive logging and monitoring capabilities. LangSmith traces and logs every interaction, from the initial query to the final answer delivery. These logs are analyzed to identify performance bottlenecks and areas for improvement. Real-time monitoring dashboards were set up to track key performance metrics, allowing for proactive maintenance and optimization of the system.

3.5 Conclusion

In conclusion, the implementation of this Local Retrieval-Augmented Generation agent demonstrates a comprehensive approach to leveraging state-of-the-art technologies for efficient and accurate information retrieval and generation. By addressing key challenges with targeted solutions, we created a scalable and robust system capable of providing high-quality responses to user queries. The continuous monitoring and logging with LangSmith ensure that the system can be maintained and optimized over time, ensuring sustained performance and reliability.

Chapter 4

General Conclusion

Throughout this project, we have embarked on a journey to develop and deploy a Local Retrieval-Augmented Generation (RAG) system, amalgamating a spectrum of cutting-edge technologies including LLaMA3, FastAPI, Langchain, ChromaDB, Ollama, Docker, and LangSmith. Our overarching aim has been to construct a versatile and efficient platform capable of retrieving relevant documents and generating precise answers to user inquiries.

The inception of our exploration involved an in-depth examination of the system's architecture, delineating the pivotal roles of each constituent component. Through meticulous scrutiny, we elucidated the sequential procedures underlying the system's operation, underscoring the significance of each stage in ensuring the fidelity and pertinence of the generated responses.

The integration of disparate technologies presented formidable challenges, encompassing facets such as streamlined document retrieval, precise document evaluation, model deployment, and system performance optimization. Through the adoption of a modular design ethos, harnessing advanced retrieval algorithms, strategic application of prompting techniques with LLaMA3, and establishment of robust monitoring and logging mechanisms, we navigated these challenges with finesse.

By incorporating prompting techniques with LLaMA3, we ensured the precision and relevance of document evaluation and answer generation, without necessitating fine-tuning of the model. This approach not only bolstered performance but also streamlined implementation efforts.

The iterative validation and refinement of answers by the LLaMA3 model culminated in the elimination of hallucinations and the accurate addressing of user queries, thereby augmenting the overall reliability and quality of the system's outputs.

In summation, the development and deployment of this Local Retrieval-Augmented Generation agent epitomize a holistic approach to leveraging state-of-the-art technologies for efficient and accurate information retrieval and generation. By adeptly addressing key challenges with tailored solutions, we have birthed a scalable and resilient platform capable of delivering high-quality responses to user inquiries. The continuous monitoring and logging facilitated by LangSmith ensure that the system remains maintainable and optimizable over time, thereby guaranteeing sustained performance and reliability.

Bibliography

[1] Yue Zhang et al. *Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models*. 2023. arXiv: 2309.01219 [cs.CL].

[2] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL].

[3] Sebastian Riedel et al. *Retrieval Augmented Generation: Streamlining the creation of intelligent natural language processing models*. https://ai.meta.com/blog/retrieval-augmented-generation-streamlining-the-creation-of-intelligent-natural-language-processing-models/. 2020.

[4] Alec Radford. *Improving language understanding with unsupervised learning*. https://openai.com/index/language-unsupervised. 2018.

[5] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[6] Matt Crabtree. *What is Machine Learning? Definition, Types, Tools & More*. https://www.datacamp.com/blog/what-is-machine-learning. 2023.

[7] Moez Ali. *Supervised Machine Learning*. https://www.datacamp.com/blog/supervised-machine-learning. 2022.

[8] IBM. *Supervised Machine Learning*. URL: https://www.ibm.com/topics/unsupervised-learning.

[9] AWS. *What is Reinforcement Learning?* URL: https://aws.amazon.com/what-is/reinforcement-learning/..

[10] Abid Ali Awan. *What is Deep Learning? A Tutorial for Beginners*. https://www.datacamp.com/tutorial/tutorial-deep-learning-tutorial. 2023.

[11] Amirsina Torfi et al. *Natural Language Processing Advancements By Deep Learning: A Survey*. 2021. arXiv: 2003.01200 [cs.CL].

[12] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[13] Zachary C. Lipton, John Berkowitz, and Charles Elkan. *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 2015. arXiv: 1506.00019 [cs.LG].

[14] Michael Nielsen. *Neural Networks and Deep Learning*. 2015. URL: https://www.ise.ncsu.edu/fuzzy-neural/wp-content/uploads/sites/9/2022/08/neuralnetworksanddeeplearning.pdf.

[15] S. A. BILLINGS S. CHEN and P. M. GRANT. "Non-linear system identification using neural networks". In: *International Journal of Control* (). URL: https://doi.org/10.1080/00207179008934126.

[16] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. 2019. arXiv: 1912.05911 [cs.LG].

[17] P.J. Werbos. "Backpropagation through time: what it does and how to do it". In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560. DOI: 10.1109/5.58337.

[18] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[19] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[20] Jordan Hoffmann et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv: 2203.15556 [cs.CL].

[21] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG].

[22] C. E. Shannon. "Prediction and entropy of printed English". In: *The Bell System Technical Journal* 30.1 (1951), pp. 50–64. DOI: 10.1002/j.1538-7305.1951.tb01366.x.

[23] Frederick Jelinek. "Statistical methods for speech recognition". In: 1997. URL: %5Curl%7Bhttps://api.semanticscholar.org/CorpusID:12495425%7D.

[24] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[25] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. "A Neural Probabilistic Language Model". In: *Advances in Neural Information Processing Systems*. Ed. by T. Leen, T. Dietterich, and V. Tresp. Vol. 13. MIT Press, 2000. URL: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf.

[26] Holger Schwenk, Daniel Dechelotte, and Jean-Luc Gauvain. "Continuous Space Language Models for Statistical Machine Translation". In: *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*. Sydney, Australia: Association for Computational Linguistics, July 2006, pp. 723–730. URL: https://aclanthology.org/P06-2093.

[27] Tomáš Mikolov et al. "Recurrent neural network based language model". In: *Proc. Interspeech 2010*. 2010, pp. 1045–1048. DOI: 10.21437/Interspeech.2010-343.

[28] Alex Graves. *Generating Sequences With Recurrent Neural Networks*. 2014. arXiv: 1308.0850 [cs.NE].

[29] Po-Sen Huang et al. "Learning deep structured semantic models for web search using clickthrough data". In: *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*. CIKM '13. San Francisco, California, USA: Association for Computing Machinery, 2013, pp. 2333–2338. ISBN: 9781450322638. DOI: 10.1145/2505515.2505665. URL: %5Curl%7Bhttps://doi.org/10.1145/2505515.2505665%7D.

[30] Jianfeng Gao et al. *Neural Approaches to Conversational Information Retrieval*. 2022. arXiv: 2201.05176 [cs.IR].

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: 1409.3215 [cs.CL].

[32] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. arXiv: 1409.1259 [cs.CL].

[33] Oriol Vinyals et al. *Show and Tell: A Neural Image Caption Generator*. 2015. arXiv: 1411.4555 [cs.CV].

[34] Matthew E. Peters et al. *Deep contextualized word representations*. 2018. arXiv: 1802.05365 [cs.CL].

[35] Yinhan Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].

[36] Pengcheng He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2021. arXiv: 2006.03654 [cs.CL].

[37] Ce Zhou et al. *A Comprehensive Survey on Pretrained Foundation Models: A History from BERT to ChatGPT*. 2023. arXiv: 2302.09419 [cs.AI].

[38] Xu Han et al. *Pre-Trained Models: Past, Present and Future*. 2021. arXiv: 2106.07139 [cs.AI].

[39] XiPeng Qiu et al. "Pre-trained models for natural language processing: A survey". In: *Science China Technological Sciences* 63.10 (Sept. 2020), pp. 1872–1897. ISSN: 1869-1900. DOI: 10.1007/s11431-020-1647-3. URL: http://dx.doi.org/10.1007/s11431-020-1647-3.

[40] Aakanksha Chowdhery et al. *PaLM: Scaling Language Modeling with Pathways*. 2022. arXiv: 2204.02311 [cs.CL].

[41] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[42] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL].

[43] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].

[44] Grégoire Mialon et al. *Augmented Language Models: a Survey*. 2023. arXiv: 2302.07842 [cs.CL].

[45] Shervin Minaee et al. *Large Language Models: A Survey*. 2024. arXiv: 2402.06196 [cs.CL].

[46] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. *Self-Attention with Relative Position Representations*. 2018. arXiv: 1803.02155 [cs.CL].

[47] Jianlin Su et al. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2023. arXiv: 2104.09864 [cs.CL].

[48] Ofir Press, Noah A. Smith, and Mike Lewis. *Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation*. 2022. arXiv: 2108.12409 [cs.CL].

[49] BigScience Workshop et al. *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model*. 2023. arXiv: 2211.05100 [cs.CL].

[50] Guolin Ke, Di He, and Tie-Yan Liu. *Rethinking Positional Encoding in Language Pre-training*. 2021. arXiv: 2006.15595 [cs.CL].

[51] Noam Shazeer et al. *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. 2017. arXiv: 1701.06538 [cs.LG].

[52] William Fedus, Barret Zoph, and Noam Shazeer. *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. 2022. arXiv: 2101.03961 [cs.LG].

[53] Rabeeh Karimi Mahabadi et al. *Parameter-efficient Multi-task Fine-tuning for Transformers via Shared Hypernetworks*. 2021. arXiv: 2106.04489 [cs.CL].

[54] Shengyu Zhang et al. *Instruction Tuning for Large Language Models: A Survey*. 2024. arXiv: 2308.10792 [cs.CL].

[55] Swaroop Mishra et al. *Cross-Task Generalization via Natural Language Crowdsourcing Instructions*. 2022. arXiv: 2104.08773 [cs.CL].

[56] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[57] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[58] Yizhong Wang et al. *Self-Instruct: Aligning Language Models with Self-Generated Instructions*. 2023. arXiv: 2212.10560 [cs.CL].

[59] Paul Christiano et al. *Deep reinforcement learning from human preferences*. 2023. arXiv: 1706.03741 [stat.ML].

[60] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[61] Arash Ahmadian et al. *Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs*. 2024. arXiv: 2402.14740 [cs.LG].

[62] Ziwei Ji et al. "Survey of Hallucination in Natural Language Generation". In: *ACM Computing Surveys* 55.12 (Mar. 2023), pp. 1–38. ISSN: 1557-7341. DOI: 10.1145/3571730. URL: http://dx.doi.org/10.1145/3571730.

[63] Nick McKenna et al. *Sources of Hallucination by Large Language Models on Inference Tasks*. 2023. arXiv: 2305.14552 [cs.CL].

[64] Shunyu Yao et al. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. 2023. arXiv: 2305.10601 [cs.CL].

[65] Potsawee Manakul, Adian Liusie, and Mark J. F. Gales. *SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models*. 2023. arXiv: 2303.08896 [cs.CL].

[66] Noah Shinn et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023. arXiv: 2303.11366 [cs.AI].

[67] Tongshuang Wu et al. *PromptChainer: Chaining Large Language Model Prompts through Visual Programming*. 2022. arXiv: 2203.06566 [cs.HC].

[68] Yongchao Zhou et al. *Large Language Models Are Human-Level Prompt Engineers*. 2023. arXiv: 2211.01910 [cs.LG].

[69] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL].

[70] Zhengbao Jiang et al. *Active Retrieval Augmented Generation*. 2023. arXiv: 2305.06983 [cs.CL].

[71] Timo Schick et al. *Toolformer: Language Models Can Teach Themselves to Use Tools*. 2023. arXiv: 2302.04761 [cs.CL].

[72] Shishir G. Patil et al. *Gorilla: Large Language Model Connected with Massive APIs*. 2023. arXiv: 2305.15334 [cs.CL].

[73] Nikhil Kandpal et al. *Large Language Models Struggle to Learn Long-Tail Knowledge*. 2023. arXiv: 2211.08411 [cs.CL].

[74] Daman Arora et al. *GAR-meets-RAG Paradigm for Zero-Shot Information Retrieval*. 2023. arXiv: 2310.20158 [cs.CL].

[75] Sebastian Borgeaud et al. *Improving language models by retrieving from trillions of tokens*. 2022. arXiv: 2112.04426 [cs.CL].

[76] Xinbei Ma et al. *Query Rewriting for Retrieval-Augmented Large Language Models*. 2023. arXiv: 2305.14283 [cs.CL].

[77] Wenjun Peng et al. *Large Language Model based Long-tail Query Rewriting in Taobao Search*. 2024. arXiv: 2311.03758 [cs.IR].

[78] Luyu Gao et al. *Precise Zero-Shot Dense Retrieval without Relevance Labels*. 2022. arXiv: 2212.10496 [cs.IR].

[79] Wenhao Yu et al. *Generate rather than Retrieve: Large Language Models are Strong Context Generators*. 2023. arXiv: 2209.10063 [cs.CL].

[80] Zhihong Shao et al. *Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy*. 2023. arXiv: 2305.15294 [cs.CL].

[81] Angelina Yang Mehdi Allahyari. *A Practical Approach to Retrieval Augmented Generation Systems*. https://mallahyari.github.io/rag-ebook/. 2023.

[82] Dale Markowitz. *Meet AI's multitool: Vector embeddings*. https://cloud.google.com/blog/topics/developers-practitioners/meet-ais-multitool-vector-embeddings. 2022.

Chapter 5

Apendix

5.1 Prompts used

5.1.1 Retrieval Grader

```
prompt = PromptTemplate(
    template="""<|begin_of_text|><|start_header_id|>system<|end_header_id|> You are a grader assessing relevance
    of a retrieved document to a user question. If the document contains keywords related to the user question,
    grade it as relevant. It does not need to be a stringent test. The goal is to filter out erroneous retrievals. \n
    Give a binary score 'yes' or 'no' score to indicate whether the document is relevant to the question. \n
    Provide the binary score as a JSON with a single key 'score' and no premable or explanation.
    <|eot_id|><|start_header_id|>user<|end_header_id|>
    Here is the retrieved document: \n\n {document} \n\n
    Here is the user question: {question} \n <|eot_id|><|start_header_id|>assistant<|end_header_id|>""",
    input_variables=["question", "document"],
)
```

Listing 5.1: *Prompt used for the retrieval grader.*

5.1.2 Answer Generation

```
prompt = PromptTemplate(
    template="""<|begin_of_text|><|start_header_id|>system<|end_header_id|> You are an assistant for question-answering tasks.
    Use the following pieces of retrieved context to answer the question. If you don't know the answer, just say that you don't know.
    <|eot_id|><|start_header_id|>user<|end_header_id|>
    Question: {question}
    Context: {context}
    Answer: <|eot_id|><|start_header_id|>assistant<|end_header_id|>""",
    input_variables=["question", "document"],
)
```

Listing 5.2: *Prompt used for answer generation.*

5.1.3 Hallucination Grader

```
prompt = PromptTemplate(
    template=""" <|begin_of_text|><|start_header_id|>system<|end_header_id|> You are a grader assessing whether an answer is grounded in / supported by a set of facts. Give a binary 'yes' or 'no' score to indicate whether the answer is grounded in / supported by a set of facts. Provide the binary score as a JSON with a single key 'score' and no preamble or explanation. Make sure to exactly output a json with one key 'score' <|eot_id|><|start_header_id|>user<|end_header_id|>
    Here are the facts:
    \n ------- \n
    {documents}
    \n ------- \n
    Here is the answer you need to grade and don't forget, the output needs to be in json format with one and only one key called score (yes if the answer is grouned to the facts else no): {generation} <|eot_id|><|start_header_id|>assistant<|end_header_id|>""",
    input_variables=["generation", "documents"],
)
```

Listing 5.3: *Prompt used for the hallucination grader.*

5.1.4 Answer Grader

```
prompt = PromptTemplate(
    template="""<|begin_of_text|><|start_header_id|>system<|end_header_id|> You are a grader assessing whether an answer is useful to resolve a question. Give a binary score 'yes' or 'no' to indicate whether the answer is useful to resolve a question. Provide the binary score as a JSON with a single key 'score' and no preamble or explanation. <|eot_id|><|start_header_id|>user<|end_header_id|> Here is the answer:
    \n ------- \n
    {generation}
    \n ------- \n
    Here is the question: {question} <|eot_id|><|start_header_id|>assistant<|end_header_id|>""",
    input_variables=["generation", "question"],
)
```

Listing 5.4: *Prompt used for the hallucination grader.*

5.1.5 Router

```
prompt = PromptTemplate(
```

```
        template="""<|begin_of_text|><|start_header_id|>system<|
    end_header_id|> You are an expert at routing a
        user question to a vectorstore or web search. Use the vectorstore
    for questions on LLM  agents,
        {summary}. You do not need to be stringent with the keywords
        in the question related to these topics. Otherwise, use web-search
    . Give a binary choice 'web_search'
        or 'vectorstore' based on the question. Return the a JSON with a
    single key 'datasource' and
        no premable or explanation. Question to route: {question} <|eot_id
    |><|start_header_id|>assistant<|end_header_id|>""",
        input_variables=["summary", "question"],
    )
```

Listing 5.5: *Prompt used for the hallucination grader.*