University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

Security in Machine Learning: Exposing LLM Vulnerabilities through Poisoned Vector Databases in RAG-based System

Supervisor
Prof. Mauro Conti
University of Padova

Co-supervisor
Dr. Stjepan Picek
Radboud University

Master Candidate
Nima Daryabar

Student ID
2049667

Academic Year
2023-2024

To my mother, my sister Negar, my brother-in-law Hamid, and my friends, whose constant support and encouragement have been my greatest strength. Thank you, my dear family, for your endless love, patience, and belief in me, even when the journey was tough. To my best friend Ali, and all my friends, your companionship and motivation have meant the world to me. I could not have achieved any of this without you all by my side.

Abstract

This thesis investigates the vulnerabilities of Large Language Models (LLMs) when integrated with Retrieval-Augmented Generation (RAG) systems. We particularly focus on the risks posed by malicious data poisoning. RAG systems are increasingly used to enhance LLM performance by incorporating external knowledge bases; however, there is a growing concern about their susceptibility to adversarial attacks. In this research, we specifically explore how poisoning vector databases, an essential component of RAG systems, can compromise LLM-generated responses, leading to inaccuracies.

In this study, we employed a methodology that involves injecting misleading data into vector databases and assessing the performance of several baseline and fine-tuned LLMs, including Llama-2-7B-chat-hf, Mistral-7B-Instruct-v0.2, and Llama-13B-chat-hf to explore scalability on a larger version. We fine-tuned these baseline models first on a general dataset and then on a more specialized dataset focused on physics, as the task involves answering questions mostly related to current topics in recent physics papers. The results show significant insights. While the fine-tuned versions of Llama-2-7B-chat-hf and Mistral-7B-Instruct-v0.2 performed better when exposed to benign databases compared to their baseline counterparts, they did not demonstrate increased resilience against misleading information. For instance, the fine-tuned Llama-2-7B-chat-hf demonstrated the best overall performance; however, the fine-tuned Mistral-7B-Instruct-v0.2 model showed the worst performance when exposed to poisoned data, with a 9% drop in accuracy compared to its performance on benign data. Additionally, even larger models like Llama-2-13B-chat-hf struggled with the same issue, showing a similar decrease in accuracy as the smaller models.

These findings suggest that fine-tuning, especially with datasets that do not thoroughly address the model's knowledge gaps, does not uniformly enhance resilience or robustness against misinformation. Moreover, it should be noted that LLMs' tendency to hallucinate and generate incorrect information further complicates their ability to provide accurate responses. This research highlights the need for more advanced strategies to detect and mitigate the impact of misinformation in RAG-based systems, as well as to account for the potential of models to hallucinate, which can lead to the generation of even more inaccurate responses.

Overall, in this thesis, we contribute to the security of AI by identifying critical vulnerabilities in RAG-integrated LLMs and underlining the mandatory need for strong strategies to ensure reliability and safety for AI applications, particularly in sensitive environments where the precision and integrity of information are crucial.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AI ..... Artificial Intelligence

APIRs ..... Altered Papers Incorrect Responses

BERT ..... Bidirectional Encoder Representations from Transformers

CF ..... Catastrophic Forgetting

GPT ..... Generative Pre-trained Transformer

GQA ..... Grouped-Query Attention

KG ..... Knowledge Graph

LLMs ..... Large Language Models

LLaMA ..... Meta's Large Language Model Meta AI

Llama ..... Large Language Model Meta AI

LM ..... Language Models

LSTM ..... Long Short-Term Memory

MLPs ..... Multilayer Perceptrons

MIA ..... Membership Inference Attacks

NLMs ..... Neural Language Models

NLP ..... Natural Language Processing

PDR ..... Performance Drop Rate

PLMs ..... Pre-trained Language Models

RAG ..... Retrieval-Augmented Generation

RAFT ..... Retrieval Augmented Fine Tuning

RLHF ..... Reinforcement Learning with Human Feedback

RNNs . . . . . Recurrent Neural Networks

SFT . . . . . Supervised Fine-Tuning

SLMs . . . . . Statistical Language Models

SMIA . . . . . Semantic Membership Inference Attack

SWA . . . . . Sliding Window Attention

TIRs . . . . . Total Incorrect Responses

1 Introduction

1.1 An Introduction to Security in Large Language Models

The concept of Natural Language Processing (NLP) has been evolving for decades, with foundational ideas dating back to the early 20th century. Swiss linguistics professor Ferdinand de Saussure laid the groundwork for NLP by addressing language as a system where relationships and constructs among words determine meaning or semantics [15]. In 1950, Alan Turing introduced the Turing Test in his paper "Computing Machinery and Intelligence," which assessed whether a computer could imitate human conversation well enough to be indistinguishable from a human, hence demonstrating intelligence [16].

NLP has significantly advanced over the decades, particularly in recent years, due to progress in Artificial Intelligence (AI) and neural network architectures. Today, we have entered the era of Large Language Models (LLMs), which began with the introduction of the first chatbot, ELIZA, in the 1960s [17]. ELIZA, created by Joseph Weizenbaum, simulated human conversation using pattern recognition. This innovation has continued with the development of deep learning models such as Long Short-Term Memory (LSTM) networks [18], and one of the significant advancements in NLP, the Transformer architecture [10].

These advancements have resulted in one of the most significant breakthroughs in AI: LLMs. LLMs represent a substantial advancement in NLP capabilities, able to mimic human com-

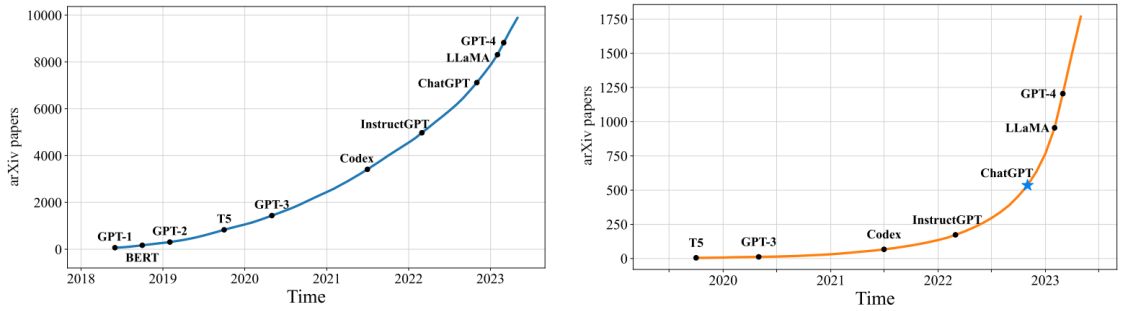
munication, and understand, and interact with human language. OpenAI, a leading AI company, recently announced its newest flagship model capable of reasoning over audio, vision, and text in real-time [19]. These models excel in various tasks, including generating coherent and relevant sentences, translating text between languages, answering questions, summarizing large texts, creating images and visual content from textual descriptions, and image captioning. Prominent examples of these models include OpenAI's Generative Pre-trained Transformer (GPT) models [20], Google's Bidirectional Encoder Representations from Transformers (BERT) [21], and Meta's Large Language Model Meta AI (LLaMA) [22].

LLMs stand out due to their complex architectures, the size of the datasets they are trained on, and the number of parameters they contain. LLMs are trained on large data corpora, such as books, articles, and internet content, involving billions of parameters. These parameters serve as the internal weights learned during training, which play a crucial role in the model's ability to comprehend and generate language. These models are typically based on transformer architectures and utilize unsupervised learning, which allows them to generate text by predicting the likelihood of word sequences based on user-provided context [10]. For example, GPT-3, one of the most well-known LLMs, has 175 billion parameters [23]. Similarly, Meta's newest model, LLaMA 3 (70B), demonstrates the massive scale of modern LLMs with its 70 billion parameters [24].

Nowadays, many companies in various fields, from small to large enterprises, as well as research institutes and laboratories, attempt to utilize these language models for specific tasks related to their purposes, ranging from scientific research to medical applications, as shown in Figure 1.1 we can see the increase of the numbers of papers published over the past years concerning Language Models (LM) and LLMs. Despite their impressive capabilities, one of the major drawbacks of these models is their inability to respond accurately to questions about recent events that are not included in their training data. This deficiency in access to the latest information can result in incorrect answers, especially in specialized fields where new terminology may be used [25].

One effective method to mitigate this problem is through fine-tuning. Fine-tuning an LLM involves training a pre-trained model on smaller, more specific datasets to adapt the model to a particular task or domain. We can easily use cloud services like Amazon AWS [26], Microsoft Azure AI Studio [27], or On-premises infrastructure to fine-tune these models, creating highly accurate and specialized language models for various applications and business use cases.

The fine-tuning process involves several steps: identifying the task and domain of the use case, gathering, and preprocessing a relevant dataset, initializing the LLM with pre-trained



(a) Query="Language Model"

(b) Query="Large Language Model"

Figure 1.1: Zhao et.al in "A Survey of Large Language Models" represent this diagram showing the numbers of arXiv papers containing the keyphrases "language model" and "large language model" since June 2018 and October 2019, respectively, reflecting current trends. These statistics are determined by exact match searches of the keyphrases in titles or abstracts monthly. Different x-axis ranges are set for the two key phrases due to the earlier exploration of "language models." Significant milestones in the research progress of LLMs are set by labeled points. figure 1(b) demonstrates that following the release of ChatGPT, there has been a notable surge in the average daily publication of arXiv papers containing the keyphrase "large language model" in titles or abstracts, increasing from 0.40 to 8.58 per day [1].

weights, fine-tuning the LLM, and finally evaluating and iterating on the results [28].

However, LLMs face some challenges, such as becoming outdated or failing to provide specific sources to validate their responses. To address these issues, we can use a method called Retrieval-Augmented Generation (RAG) [29]. This method helps keep the models up-to-date and prevents them from hallucinating—producing plausible but made-up responses (Huang et al., 2023) [30]. By providing external sources, LLMs can leverage this information to generate more accurate answers. After prompting the language model, it first retrieves relevant content, combines it with the user's question, and then generates the response. Additionally, it can provide evidence to validate the response's accuracy.

This approach addresses the problematic challenge of retraining the model with new information, especially in fast-evolving scientific fields. By augmenting the data store with new information, LLMs can access up-to-date and relevant information to handle domain-specific tasks more effectively. Nevertheless, the quality of the retriever, which provides the LLM with highly relevant grounding information, is crucial [31]. The choice of retriever directly impacts the quality and relevance of the information provided to the LLM, influencing the overall accuracy and reliability of the generated responses [32].

Thus, LLMs must pay attention to source data before generating a response, and a well-chosen retriever ensures that the most contextually relevant information is retrieved from external resources. This reduces the model's reliance on the information it was originally trained

on, leading to more accurate outputs, and decreasing the likelihood of data leakage or hallucinations [33] [34].

Despite the significant capabilities of LLMs, their reliance on vast amounts of data and complex architectures makes them vulnerable to various threats. These models raise concerns regarding their tendency to propagate biases, misinformation, and manipulation [35]. A common security issue is the use of jailbreaking to mislead the models and induce them to generate unethical responses [36]. Another problem is the presence of biases in the training data, which can emerge in the generated responses [37]. Since these models heavily rely on the data they are exposed to, malicious actors can exploit them by injecting malicious and poisonous data, leading to harmful usage, and posing serious threats.

Adversarial attacks on AI models, including LLMs, are an active area of research. These vulnerabilities such as the propagation of biases present in training data, sensitivity to adversarial inputs, and vulnerability to poisoned data and external resources. underscore the need for robust security measures to protect LLMs from being exploited in harmful ways [38].

This thesis investigates the vulnerability of LLMs to malicious information through Retrieval-Augmented Generation systems. RAG systems enhance the capabilities of LLMs by integrating external knowledge bases, typically stored in vector databases, to improve the relevance and accuracy of generated responses [39]. However, this integration also introduces new attack vectors. By poisoning these vector databases with misleading data, we aim to analyze the potential weaknesses in RAG-based systems and explore the risks of misinformation and manipulation by attackers. Understanding these vulnerabilities is crucial for developing more secure and reliable AI systems.

1.2 Research Questions and Objectives

Understanding the vulnerabilities of LLMs is crucial for developing more secure and reliable AI systems [38]. As LLMs, especially when integrated with RAG systems, are increasingly used in various applications, from customer service to content creation and beyond, ensuring their integrity and trustworthiness becomes paramount. The combination of LLMs with RAG systems introduces additional complexity, as they rely on external knowledge bases, further emphasizing the need for robust security measures.

In automated customer service, RAG systems enhance the quality and efficiency of interactions by combining advanced retrieval mechanisms with generative AI models. They dynamically retrieve relevant information, ensuring accurate and up-to-date responses. This technol-

ogy enables personalized replies, reduces response times, and lowers operational costs, while also handling complex queries by integrating knowledge from various sources. Xu et al. [40] discuss a system that combines RAG with a Knowledge Graph (KG), improving retrieval accuracy and answer quality, with significant performance gains in BLEU and ROUGE metrics

Similarly, In the educational sector, RAG systems improve content generation by creating personalized learning materials and interactive tools tailored to student needs, enhancing engagement and learning outcomes. RAG can also automate content updates, ensuring alignment with the latest research. This makes RAG a valuable asset for online learning, textbook creation, and dynamic assessments. Manathunga et al. [41] explore the use of LLMs in medical education, addressing limitations such as hallucinations and updating issues by integrating a non-parametric knowledge base. They propose a Representative Vector Summarization (RVS) method to summarize large unstructured text, improving information retrieval accuracy in medical education.

While RAG systems have been effectively used in various fields, the introduction of malicious actors who insert poisonous content into external resources poses significant risks. These attacks can cause LLMs to generate inaccurate or misleading information, potentially leading to the spread of misinformation. This research focuses on examining how the insertion of malicious and inaccurate information into external knowledge bases affects the accuracy of LLMs, particularly how RAG-based systems handle this issue and the extent to which it impacts the accuracy of the baseline model integrated with RAG compared to the fine-tuned version on a domain-specific dataset. We aim to understand whether inserting 20% inaccurate, recent papers related to the field, which RAG systems rely on to enhance responses, would significantly degrade the accuracy of the generated outputs. Our goal is to assess the susceptibility of LLMs to external malicious data, identify the specific impacts of poisoned content, and uncover vulnerabilities in RAG-based systems. By addressing these issues, we aim to improve the resilience of AI systems against malicious attacks.

1.3 Methodology Overview

To explore these vulnerabilities, We utilized a methodology that involved poisoning vector databases with misleading data and analyzing the responses of LLMs to queries drawing from these compromised databases. We can easily store our external knowledge base on private cloud storage on the organization's infrastructure or by using public cloud storage and file-sharing services like Microsoft OneDrive, Google Drive, Mega, or Dropbox. However, like all other

services, security risks are among the common challenges of such platforms. In this project, we conducted experiments by altering parts of these external resources in the form of PDFs accessed through Dropbox. These PDFs contained misinformation in specific, up-to-date domain areas, such as physics and artificial intelligence, and included newly published papers demonstrating recent discoveries in this field that the language models probably had not been trained on.

By conducting a quick review of HuggingFace [42], a platform that develops computational tools for machine learning-based applications and serves as a collaborative space where users and organizations can share and improve machine learning technologies and particularly large language models, we observed a significant difference in download rates between smaller and larger models. For example, in the case of the LLaMA 3 family by Meta, the 8 billion instruction-tuned model was downloaded more than two and a half million times in the last month, compared to its larger version with 70 billion parameters, which was downloaded around 330 thousand times [43] [44]. A similar pattern can be seen for the state-of-the-art Google Gemma and Mistral AI large language models. Thus, these experiments were performed on smaller models and smaller datasets to determine their feasibility.

Moreover, we injected this malicious data into vector databases and evaluated LLMs' responses to queries drawing from these compromised knowledge bases. This approach aimed to uncover specific patterns and conditions under which LLMs are most vulnerable, thereby contributing to the development of more robust and secure AI systems. All the code for this project is available in my GitHub repository [45].

2 Background

2.1 An Introduction to Natural Language Processing

Every day, there is competition from individuals to large companies to develop new models or enhance existing AI models to outperform the current ones specifically Large Language Models. For example, Meta has recently launched its state-of-the-art model, Llama 3.1, the largest open-source model with 405 billion parameters [46]. It is seen as one of the most impressive breakthroughs in language models. This astonishing model is designed to perform various tasks, including long-text summarization, multilingual conversational agents, and coding assistance. It represents a significant step in the ongoing rivalry between open-source and top closed-source LLMs by allowing for greater transparency, and community collaboration, and will accelerate innovation and the creation of more advanced AI tools, making it more accessible for smaller organizations and individuals to manage the high cost of LLMs training [47].

These rapid developments in large language models show a promising future for AI and the open-source community. As shown in Figure 2.1 there is a surge in the number of publications in LLMs over the past years up to the year 2023. However, open-source models come with challenges. Since malicious actors can access the code and data of such models, they can fine-tune them on poisonous data from malicious sources, such as the dark web. As a result, the

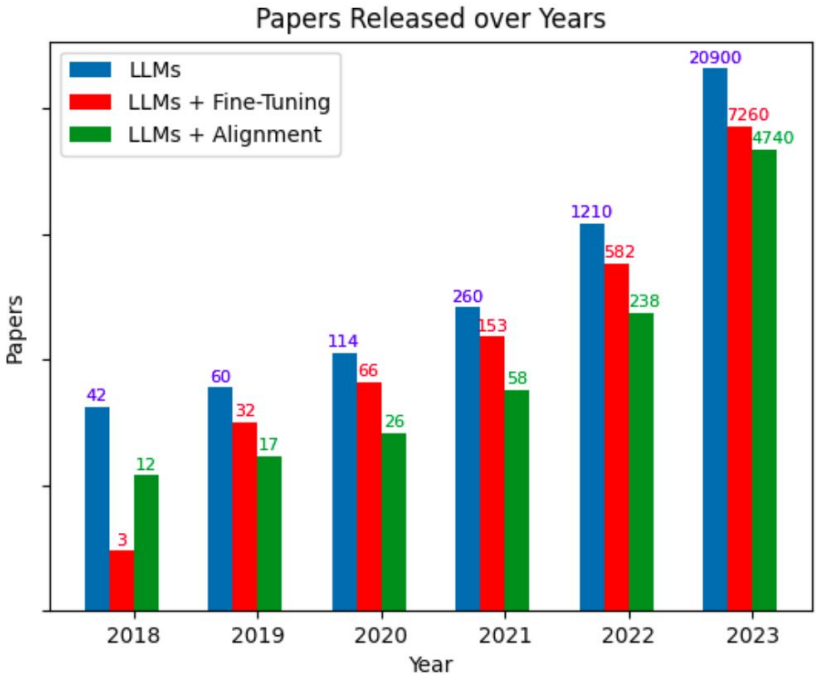


Figure 2.1: Naveed et al. in "A Comprehensive Overview of Large Language Models", demonstrate the number of papers published annually with the specified keywords such as "Large Language Model", "Large Language Model + Fine-Tuning", and "Large Language Model + Alignment" [2].

newly trained models can show biases, suffer from many security risks, and be used by malicious actors to perform cyberattacks [47].

2.2 An Introduction to Large Language Models

A Language Model first developed in 1980 is a probabilistic model for understanding and generating natural language. These models help with many tasks such as speech recognition, machine translation, text generation, grammar induction, and more. LLMs, an advanced form of language models, utilize large datasets and transformer architecture, surpassing previous Recurrent Neural Networks (RNN) and statistical models [48]. LLMs are essentially foundation models—extensive deep learning models trained on enormous quantities of data. They are designed for general-purpose language generation and NLP tasks, including classification, translation, summarization, and more [49], [50], [51]. These models aim to estimate the probability of the next token or sequence of tokens in a longer sequence [52]. They are trained using self-supervised and semi-supervised methods based on learning statistical relationships from a large

amount of data and texts [49].

Early language models were good at predicting word order but struggled with understanding the deeper meaning of language. Due to this problem, Neural Language Models (NLMs) were introduced. They use powerful neural networks like Multilayer Perceptrons (MLPs) and Recurrent Neural Networks (RNNs). These models perform better than a simple sequence prediction, by capturing the relationships between words and understanding the underlying context and semantics of language. To enable models to understand the meaning and context of words within a sentence, tools like word2vec, a shallow neural network, are used to generate distinct vector representations (embeddings) for each word [4].

This shift from statistical models to NLMs has enabled models to process language more precisely and expand their capability to handle more tasks like text generation and language translation [4].

The evolution from statistical methods to Pre-trained Language Models (PLMs) involved self-supervised training on extensive text datasets to develop general language representations. Fine-tuned PLMs have shown significant performance compared to traditional models, all leading to the development of LLMs with largely more parameters that are trained on larger datasets [2]. The development of LLMs started with early experiments in neural networks and NLP and progressed through numerous breakthroughs to become today's advanced models, as shown in Figure 2.2, we can see the evolutionary tree of language models over the recent years. Below we can find a brief history of the evolution of these models over the past decades.

- In the 1950s, IBM and Georgetown University researchers collaborated on creating systems that were able to translate Russian to English [53].
- During the 1960s, Joseph Weizenbaum from MIT developed Eliza, the world's first chatbot, a significant step in NLP that demonstrated the computer's potential to generate human-like language. It is considered the foundation of recent developments in language models [53].
- In the 1990s, Statistical Language Models (SLMs) predicted the next words based on the preceding words using the Markov assumption. These cutting-edge statistical language models, such as IBM's alignment models and smoothed n-gram models, could train on large datasets but were constrained by data sparsity issues [51].
- The emergence of LSTM networks in 1997 was a significant advancement and led to the development of deeper and more complex neural networks capable of handling larger datasets. This development enhanced the performance of tasks like sentiment analysis and named entity recognition [53].

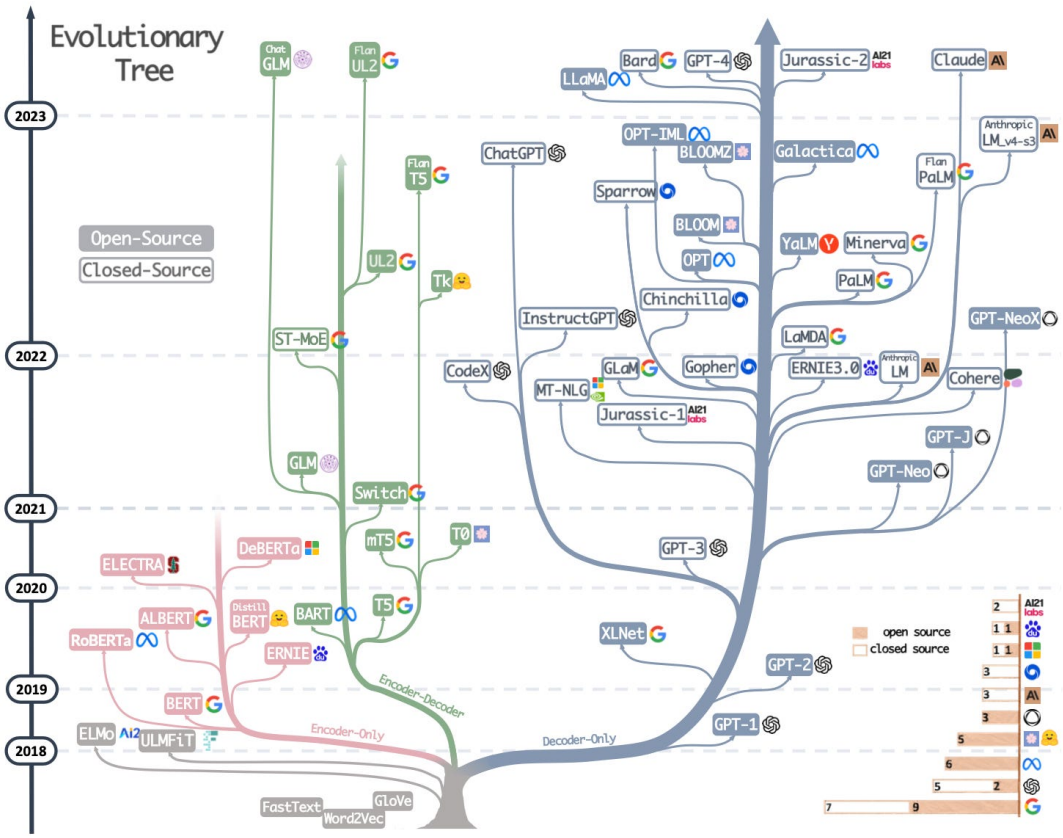


Figure 2.2: Yang et al. in "Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond", represent the development of language models in recent years shown by the evolutionary tree of modern LLMs, highlighting some of the most well-known models. Closer relationships are indicated for models on the same branch, with transformer-based models shown in non-grey colors: decoder-only models in the blue branch, encoder-only models in the pink branch, and encoder-decoder models in the green branch. The release dates of the models are represented by their vertical position on the timeline. Solid squares represent open-source models, while hollow squares represent closed-source models. Additionally, the number of models from various companies and institutions is displayed in the stacked bar plot in the bottom right corner [3].

- This period could be seen as the era of Transformers architecture and the BERT model. Around 2012, the advent of neural networks led to a shift in language modeling towards utilizing deep learning techniques. Google's implementation of Neural Machine Translation in 2016, using seq2seq deep LSTM networks, represented a significant advancement in the field. The introduction of the transformer architecture in the 2017 paper "Attention Is All You Need" is considered a revolutionary step in NLP, as it enhances the efficiency and accuracy of language models through the attention mechanism. Subsequently, Google's BERT, introduced in 2018, became a unique model for many NLP tasks, trained on large unstructured data through self-supervised learning [51].

- OpenAI's GPT series, including GPT-2 released in 2019 with 1.5 billion parameters [54], and GPT-3 in 2020 with 175 billion parameters [23], pushed the boundaries of LLMs with their impressive text generation capabilities. However, GPT-3 was initially withheld from public release due to concerns over misuse [55]. These models, especially GPT-3, set a new standard for LLMs in few-shot learning, allowing them to execute various tasks without task-specific fine-tuning [2]. The introduction of ChatGPT in 2022 made LLMs widely accessible to the public, highlighting the practical applications of these models in everyday use. The development of GPT-4 in 2023, with an estimated 1.8 trillion parameters [56], marked a notable advance as a Transformer-based model with enhanced accuracy and multimodal abilities, showing human-level performance on several benchmarks [57].

This progression, as visually represented in Figure 2.3, from early neural networks to today's LLMs such as GPT-4 illustrates the rapid evolution of language models, which continue to shape the NLP and AI landscape. The continual innovation in LLMs reflects their significant impact on research and practical applications, fundamentally revolutionizing our interactions with technology and information.

2.3 Foundations of LLMs

To understand the capabilities of LLMs, it is essential to understand the underlying architecture that sets these models apart. The distinctiveness of LLMs lies not only in their extensive training on massive datasets but also in the advanced neural network architecture they employ. At the heart of these models is the Transformer architecture, a groundbreaking development introduced by researchers at Google in 2017 [58]. The Transformer utilizes a self-attention mechanism, which efficiently handles sequences of data by weighing the relevance of different words in a sequence relative to each other. This innovation has allowed Transformers to surpass

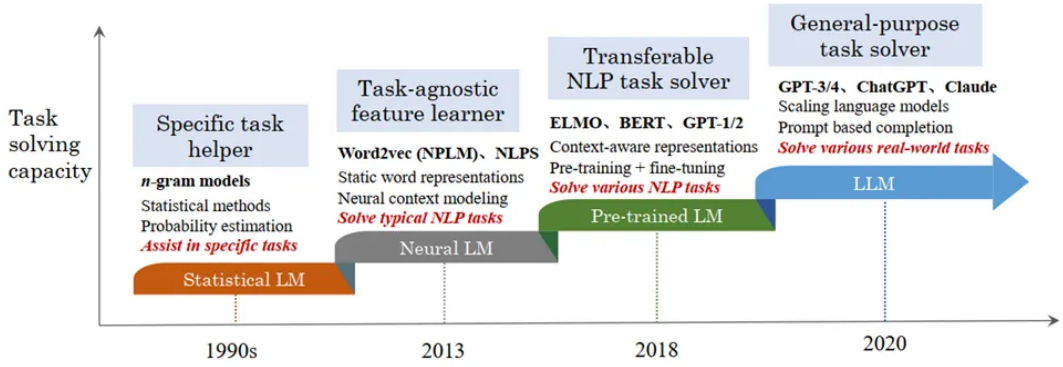


Figure 2.3: Zhao et al. in "A Survey of Large Language Models" represent the progress of language models (LM) across four generations concerning their ability to solve tasks [4].

earlier models, such as RNNs and Convolutional Neural Networks (CNNs), in their ability to process and generate large-scale text data efficiently [2].

Before delving into the specifics of Transformer architecture and the data processing in LLMs, it is important to briefly review the neural network architectures that preceded and influenced its development.

2.3.1 Recurrent Neural Networks

RNNs represent an early and significant advancement in the development of neural networks designed to handle sequential data. Unlike traditional feedforward neural networks, which process inputs independently of each other, RNNs are characterized by their use of feedback loops that allow information to persist. This architecture enables RNNs to process sequences of varying lengths by maintaining a hidden state that captures information from previous inputs, thereby making predictions based on the sequential nature of the data [5], [59].

One of the core features of RNNs is that they share the same weights and biases across each step in the sequence. This weight sharing is essential because it reduces the complexity of the model and ensures that the network can generalize across different positions in the input sequence. However, this same feature also introduces significant challenges during training, particularly when dealing with long sequences [5]. A comparison between Feedforward Neural Networks and Recurrent Neural Networks is represented in Figure 2.4.

As RNNs process longer sequences, they encounter what is known as the vanishing/exploding gradient problem. This problem arises during the training phase when gradients—the values used to update the model's weights—either shrink (vanishing) or grow exponentially (ex-

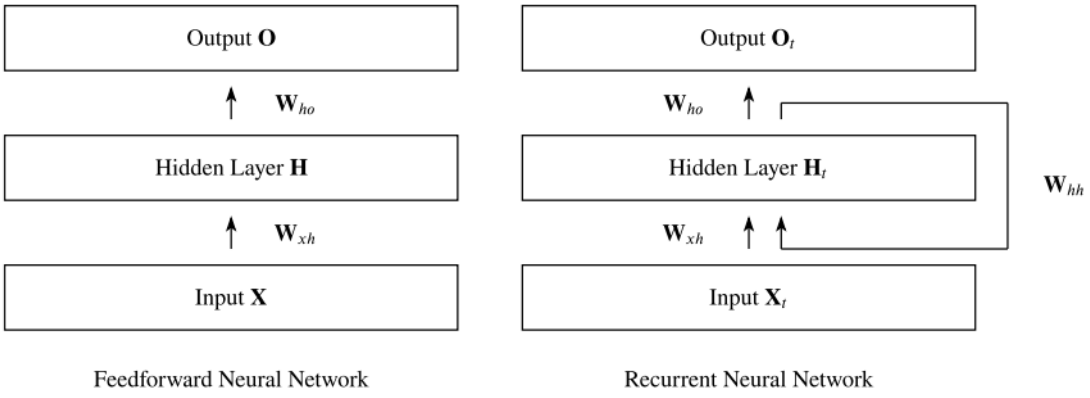


Figure 2.4: M. Schmidt Compares Feedforward NNs and Recurrent NNs [5].

ploding) as they are backpropagated through time. In the case of vanishing gradients, as the network backpropagates through many layers (or time steps), the gradients become so small that they effectively prevent the model from learning from the early parts of the sequence. Conversely, with exploding gradients, the gradients become excessively large, leading to unstable updates that can cause the model to diverge rather than converge [5].

The vanishing gradient problem typically occurs when the weights in the feedback loop are less than one. As a result, the contribution of earlier inputs diminishes as the sequence length increases, making it difficult for the network to retain information from earlier in the sequence. On the other hand, if the weights exceed one, the network may suffer from the exploding gradient problem, where the gradients grow too large, causing significant oscillations during training and making it difficult to find an optimal set of weights [5].

To address these issues, researchers have developed various techniques, such as gradient clipping [60], where gradients are scaled down if they exceed a certain threshold—and the introduction of more complex architectures like LSTM networks, which specifically tackle the challenges of vanishing and exploding gradients. Despite these improvements, RNNs still struggle with very long sequences due to the inherent limitations of their architecture.

RNNs laid the groundwork for sequence modeling and paved the way for more advanced models, but their limitations in handling long-range dependencies led to the development of more sophisticated architectures, which will be discussed in the following sections.

2.3.2 Long Short-Term Memory (LSTM)

LSTM networks were developed to overcome the significant limitations of RNNs, particularly the vanishing and exploding gradient problems. These issues occur during the training of RNNs when gradients either diminish to near zero or grow uncontrollably as they are propagated through time steps, making it difficult for the network to learn from long sequences [5].

LSTMs introduce a more advanced architecture designed to manage both long-term and short-term dependencies effectively. The key innovation in LSTMs lies in their ability to control the flow of information through three distinct gates—input, forget, and output gates. These gates determine which parts of the input should be stored, which should be discarded, and what the output should be at each time step. This gating mechanism enables LSTMs to retain important information across long sequences without suffering from the vanishing or exploding gradient problems that typically affect RNNs [6], [18].

The cell state is a crucial feature of LSTMs, acting as the network's long-term memory. This state flows through the network with minimal changes, ensuring that essential information is preserved throughout the sequence. The stability of the cell state is a key factor in preventing the gradient issues that affect traditional RNNs, as weight updates do not directly influence it. The forget gate, in particular, plays a critical role by deciding how much of the cell state should be passed on or discarded, allowing the model to manage long-term memory effectively [6].

Alongside the cell state, the hidden state handles short-term memories and is directly influenced by the network's weights and biases. The hidden state is updated at each time step, capturing the immediate context necessary for making predictions. The combination of these two memory systems allows LSTMs to handle sequences of varying lengths more effectively than standard RNNs, making them particularly suitable for tasks requiring the modeling of long-range dependencies, such as language modeling and time-series prediction [6].

LSTMs operate through the following stages:

- Forget Gate: Determines how much of the previous cell state should be retained or discarded.
- Input Gate: Decides how much new information from the current input should be added to the cell state.
- Output Gate: Controls how much of the cell state should be output as the hidden state, which influences the next step in the sequence.

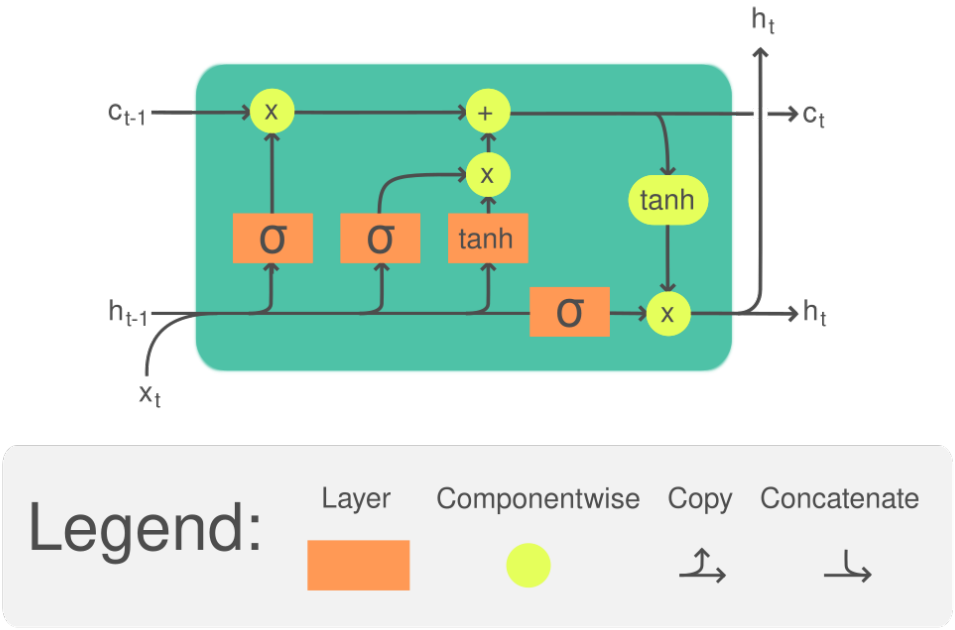


Figure 2.5: The LSTM cell can sequentially process data and retain its hidden state over time [6].

By using these gated pathways, LSTMs can maintain and update information dynamically, allowing them to model long-term dependencies in data sequences without the training difficulties encountered by RNNs. This makes LSTMs particularly powerful for tasks requiring long-range context, such as language modeling and time-series prediction. Below in Figure 2.5, we can see a diagram of an LSTM cell.

2.4 Word Representation

Before diving into advanced neural network architectures, it's crucial to understand how machine learning algorithms and neural networks process words as input. Since neural networks require numerical data, words must be converted into a numerical format to be effectively processed [61].

The Challenge of Word Representation

A naive approach might involve assigning a unique random number to each word in the vocabulary. However, this method fails to capture the semantic relationships between words.

Words with similar meanings could end up with entirely different numerical values, which would make it difficult for the model to understand and generalize across different contexts.

Word Embeddings: A Solution

To address these challenges, word embeddings are used. Word embeddings are dense vector representations of words where each word is mapped to a point in a continuous vector space. The key advantage of word embeddings is that words with similar meanings or that occur in similar contexts are placed closer together in this vector space. This allows the model to understand semantic relationships between words, improving its ability to perform various natural language processing tasks [61].

Learning Word Embeddings

Word embeddings are typically learned by training a neural network on a large corpus of text, optimizing the network's weights based on the context in which words appear. The objective is to produce embeddings where semantically similar words have similar vector representations.

One of the most popular methods for generating word embeddings is word2vec, developed by Mikolov et al. at Google in 2013. Word2vec includes two primary architectures: Continuous Bag of Words (CBOW) and Skip-Gram [7], [62].

- Continuous Bag of Words (CBOW): In this approach, the model predicts a target word based on its surrounding context words. CBOW uses the context of surrounding words to predict the central word, effectively capturing the meaning of the word concerning its neighbors [7].

- Skip-Gram: Unlike CBOW, Skip-Gram predicts the surrounding context words given a target word. This method is particularly powerful for capturing complex semantic relationships, especially in large corpora where words can have varied meanings depending on context [7].

Both architectures, as shown in Figure 2.6 utilize a shallow neural network, where the hidden layer represents the word embeddings. As the model trains, the embeddings are adjusted so that words used in similar contexts end up with similar vector representations. This method allows the model to generalize across contexts, improving its performance on tasks like text classification, machine translation, and more [7].

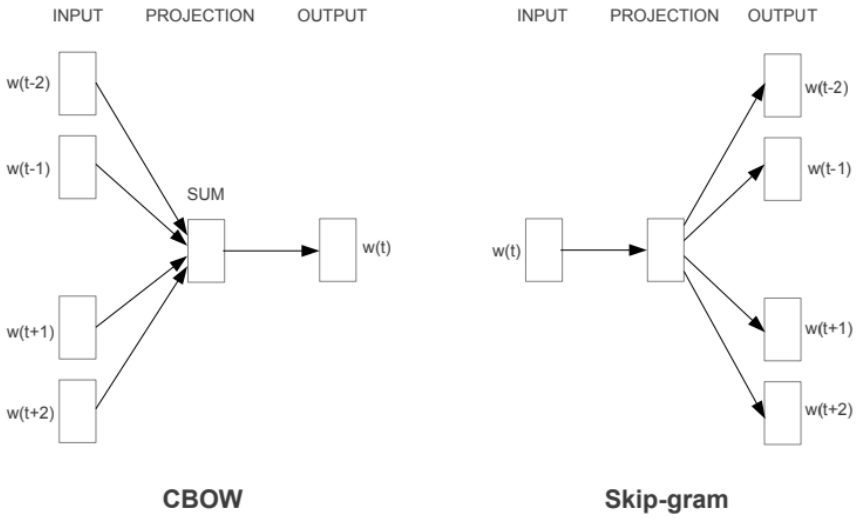


Figure 2.6: The CBOW model architecture anticipates the present word using the context, while the Skip-gram model forecasts surrounding words based on the current word [7].

Addressing Computational Complexity

Training word embeddings, especially with large datasets, can be computationally expensive due to the vast number of parameters involved. To mitigate this, negative sampling is often used, a technique introduced alongside word2vec. Negative sampling reduces the computational load by only updating a small random subset of weights during each training step, rather than the entire set of weights. This allows for more efficient training while still producing high-quality word embeddings [62].

Another method, GloVe (Global Vectors for Word Representation), developed by researchers at Stanford, builds on word2vec by incorporating global statistical information from the corpus. GloVe constructs word embeddings by factorizing the word co-occurrence matrix, which captures the frequency with which pairs of words appear together in the corpus. This approach combines the strengths of both global matrix factorization and local context window methods like word2vec, resulting in embeddings that effectively capture both the overall structure of the language and the nuances of individual word usage [63].

Applications of Word Embeddings

Word embeddings are foundational in modern NLP tasks. By providing a way to represent words numerically while preserving their semantic relationships, embeddings are used in a wide range of applications, from sentiment analysis and machine translation to more advanced tasks like question answering and named entity recognition. The ability to process words in a manner that respects their context and meaning is what makes word embeddings such a powerful tool in the arsenal of modern NLP techniques [7].

2.5 Sequence-to-Sequence (Seq2Seq) and Encoder-Decoder Architectures

2.5.1 Sequence-to-Sequence Models

Sequence-to-Sequence (Seq2Seq) models are a foundation in NLP for tasks that involve transforming one sequence of data into another. These tasks include machine translation, text summarization, speech recognition, and more. Seq2Seq models operate by encoding an input sequence into a fixed-size context vector and then decoding this vector to generate an output sequence. This framework allows Seq2Seq models to handle variable-length sequences in both inputs and outputs, making them exceptionally versatile [8], [64], [65].

2.5.2 Encoder-Decoder Architecture

The encoder-decoder architecture is central to Seq2Seq models. It consists of two primary components: the encoder and the decoder. The encoder processes the input sequence and compresses it into a fixed-size context vector that encapsulates the information from the entire sequence. This context vector is then fed into the decoder, which generates the output sequence one element at a time based on the information in the context vector [8].

For example, in machine translation, the encoder takes a sentence in the source language and encodes it into a context vector. The decoder then uses this vector to produce the corresponding sentence in the target language. The ability of Seq2Seq models to manage sequences of varying lengths is crucial for handling real-world NLP tasks, where input and output sequences often differ in length.

2.5.3 Role of LSTM in Seq2Seq Models

The original Seq2Seq models employed LSTM networks due to their proficiency in managing long-range dependencies in sequential data. In these models, the encoder typically consists of a stack of LSTM layers that process the input sequence. Each LSTM layer generates a hidden state, which collectively forms the context vector. The decoder, also composed of LSTM layers, takes this context vector and generates the output sequence, predicting each word step by step [8], [64].

However, a significant limitation of basic Seq2Seq models is that they compress the entire input sequence into a single context vector. While this approach works adequately for short sequences, it can be problematic for longer sequences, where early input words may be forgotten, leading to a loss of critical information. This can cause the generated output to deviate from the intended meaning of the input [8], [64].

2.5.4 Introduction of Attention Mechanisms

Attention mechanisms were introduced by Bahdanau et al. in 2014 to address the limitations of compressing the entire input into a single context vector [66]. The key idea behind attention is to allow the decoder to focus on specific parts of the input sequence during the generation of each output word. Instead of relying solely on a single context vector, the decoder dynamically attends to different parts of the input sequence, depending on the word it is currently generating [8], [65].

Attention mechanisms compute a similarity score between the hidden states of the encoder (which represent the input sequence) and the current state of the decoder. This score determines how much attention the decoder should pay to each part of the input when generating the next word in the output sequence [64], [10]. In Figure 2.7 a Seq2Seq RNN encoder-decoder with attention mechanism is illustrated.

2.5.5 Cosine Similarity and Dot Product in Attention Mechanisms

Several methods can be used to calculate these similarity scores, one of the simplest being cosine similarity. Cosine similarity measures the cosine of the angle between two vectors, providing a value between -1 and 1, indicating how similar the two vectors are. However, in practice, the dot product is often preferred over full cosine similarity because it simplifies the calculation and

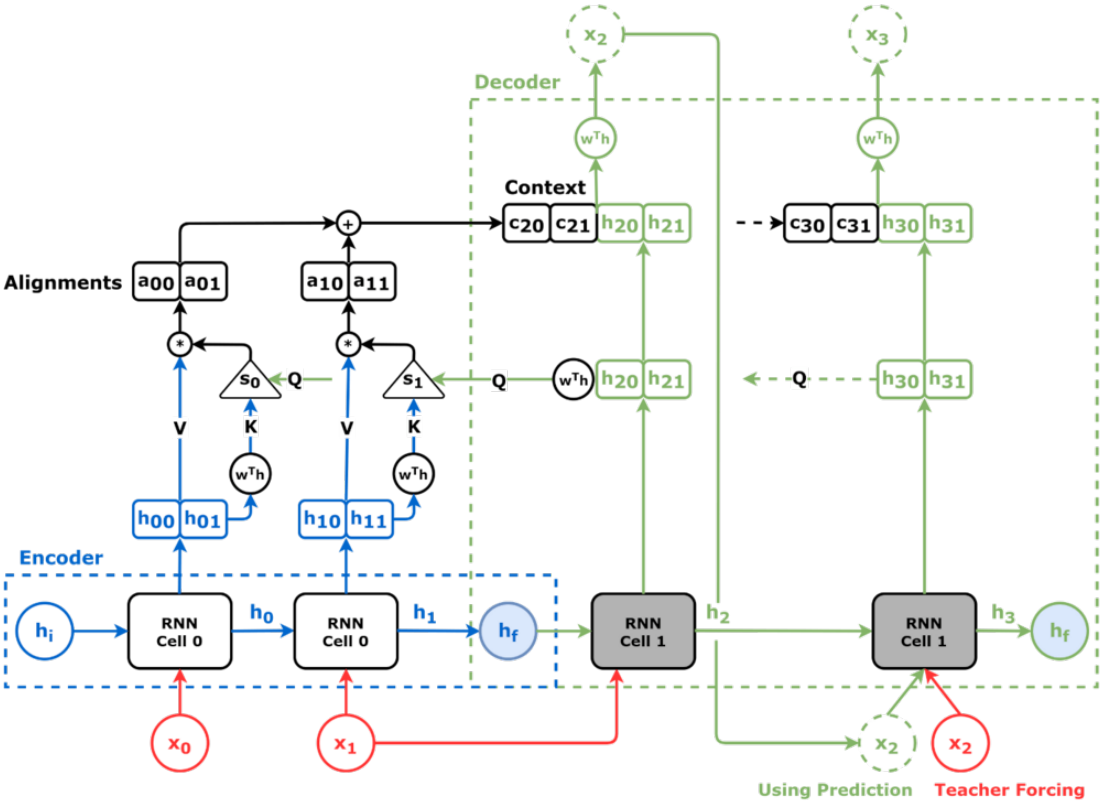


Figure 2.7: The architecture of seq2seq RNN encoder-decoder includes an attention mechanism [8].

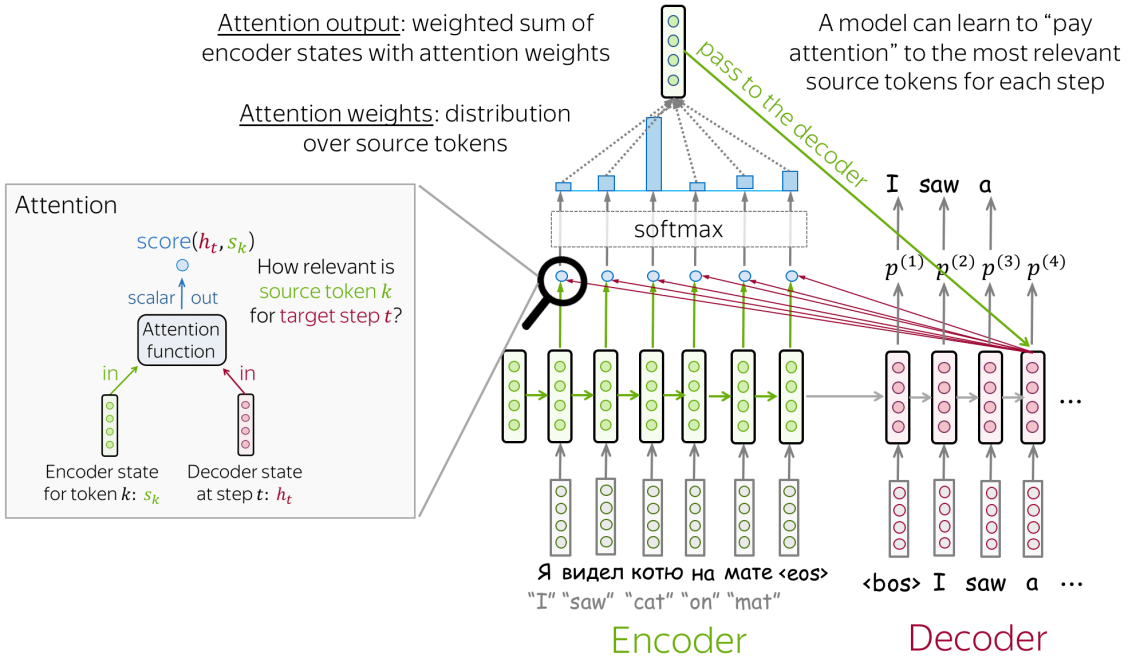


Figure 2.8: Attention mechanism scheme [9].

works effectively in attention mechanisms. The dot product offers a straightforward measure of similarity that is easier to compute and integrate into neural network architectures [10].

2.5.6 Applying Softmax to Attention Scores

Once the similarity scores are calculated, they are typically passed through a Softmax function. This function normalizes the scores into a probability distribution, determining what percentage of each encoded input word should be used to influence the current output word. The attention scores are then applied to the encoder's outputs, allowing the decoder to focus more on the relevant parts of the input sequence [10]. In Figure 2.8 we can find a general scheme of attention mechanism with a softmax layer.

2.5.7 Evolution Beyond LSTMs

While LSTMs were initially central to Seq2Seq models, the introduction of attention mechanisms reduced the need for LSTMs to maintain long-term dependencies, leading to the development of more sophisticated architectures like the Transformer. The attention mechanism's ability to dynamically focus on different parts of the input sequence made it possible to move

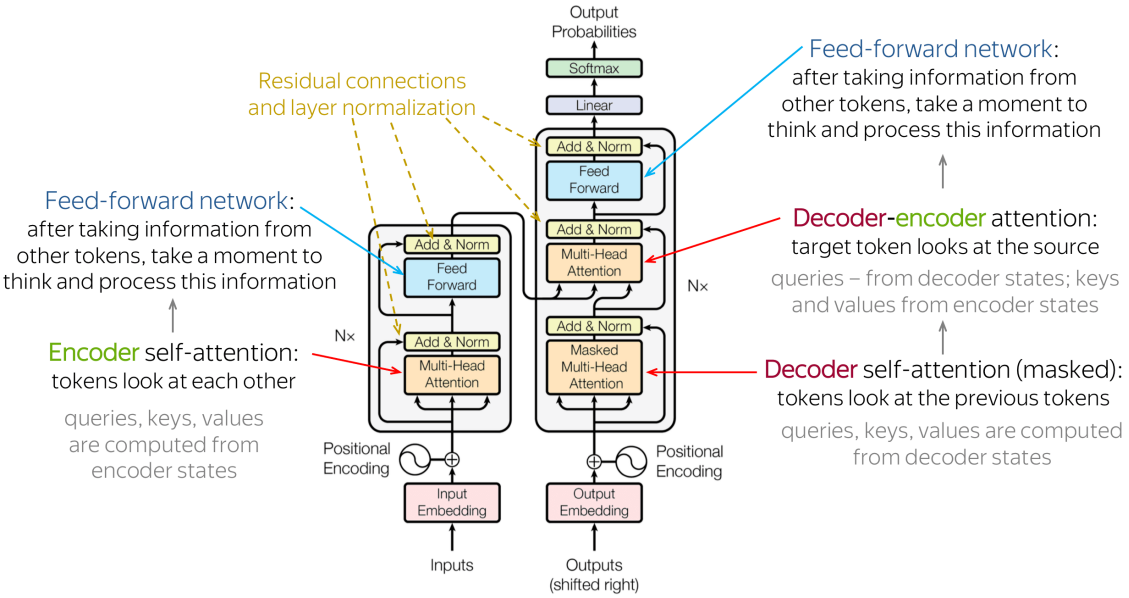


Figure 2.9: The Transformer model architecture [9].

beyond LSTM-based models, paving the way for architectures that rely entirely on attention, such as the Transformer, which will be discussed in the next section.

2.6 Transformer Architecture

The Transformer architecture, introduced by Vaswani et al. in 2017 [10], revolutionized NLP by addressing key limitations of earlier models like RNNs and LSTM networks. Transformers are designed to process sequences of data in parallel rather than sequentially, significantly improving the efficiency and scalability of language models [65], [10]. In Figure 2.9 the transformer model architecture is represented and in the following section, we will find the description of its components.

2.6.1 Word Embeddings and Positional Encoding

In Transformers, the first step is to transform words into numerical representations using word embeddings. However, since word embeddings alone do not capture the order of words in a sentence, Transformers use positional encoding to incorporate word order into the model. In this architecture, positional encoding is achieved using sine and cosine functions of different frequencies [10]. Specifically, the positional encoding in the paper is defined as

$$PE_{(pos,2i)} = sin(pos/10000^{2i/dmodel}), \tag{2.1}$$

and

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/dmodel}). \tag{2.2}$$

In this formulation, *pos* indicates a word's position in the sequence, and i represents the model's dimension. Each dimension employs a sinusoidal function with wavelengths that geometrically progress from 2π to $10000 \cdot 2\pi$. This design enables the model to attend to relative positions effectively, as the positional encoding at $pos + k$ can be linearly derived from that at *pos* for any offset k [10].

These functions generate a sequence of values that, when added to the word embeddings, allow the model to understand the relative positions of words within a sentence. This ensures that the model can distinguish between sentences with the same words but different word orders. For example, if the order of words is reversed, the positional encodings will adjust accordingly, enabling the Transformer to maintain the correct word order in its processing [10], [58].

2.6.2 Self-Attention Mechanism

A core innovation of the Transformer architecture is the self-attention mechanism. Self-attention allows the model to evaluate the relationship between each word in a sentence and every other word, including the word itself. This mechanism computes attention scores that quantify the importance of each word relative to others in the sequence [10].

In practice, self-attention works by calculating three vectors for each word in the sequence: Query (Q), Key (K), and Value (V). The similarity between Query and Key vectors is computed using a scaled dot product, where the dot product is divided by the square root of the dimension of the Key vectors to stabilize the gradients. The resulting scores are passed through a Softmax function to determine the importance of each word. The Value vectors are then weighted by these attention scores to produce the final output for each word [10]. The equation to calculate the attention output is defined as

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V. \tag{2.3}$$

2.6.3 Multi-Head Attention

Transformers extend the self-attention mechanism through multi-head attention. Instead of applying a single attention function, multi-head attention allows the model to apply multiple attention mechanisms in parallel, each focusing on different aspects of the word relationships within the sequence. The outputs from all attention heads are then concatenated and passed through a linear transformation to produce the final attention output [10]. Below we can find the formulation of multiple attention mechanisms that the outcomes are then merged. This process is defined as

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_b)W^O, \tag{2.4}$$

where

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.5}$$

In Figure 2.10, on the left, you will see the illustration of Scaled Dot-Product Attention, while on the right, there is a demonstration of Multi-Head Attention with multiple attention layers running simultaneously.

2.6.4 Position-Wise Feed-Forward Networks

After multi-head attention, each layer in the Transformer includes a position-wise feed-forward network, which applies two linear transformations with a ReLU activation in between, independently to each position in the sequence. This helps to further process the data and contribute to the model's overall representational power [10].

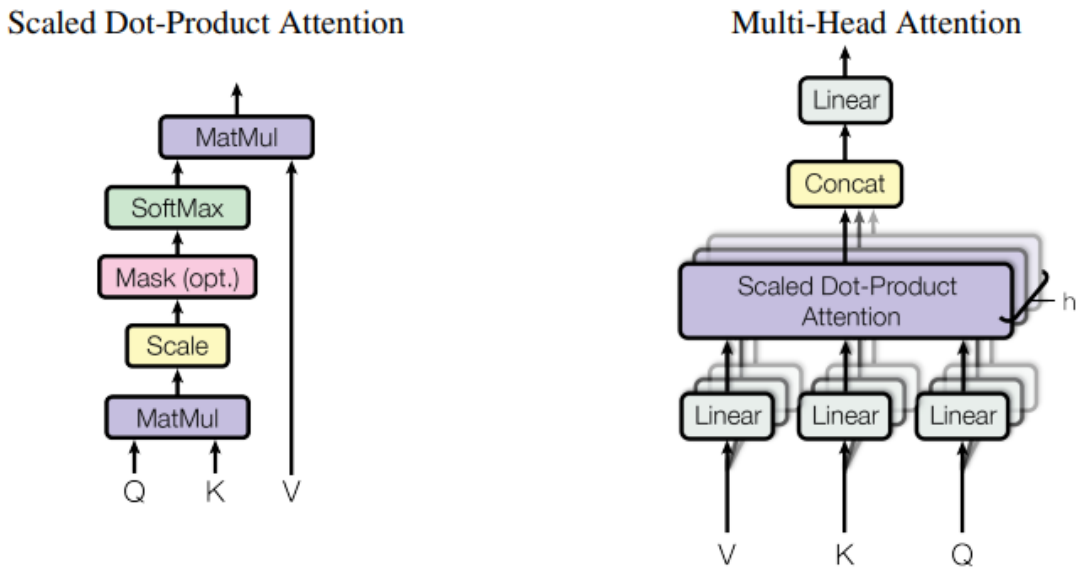


Figure 2.10: Scaled Dot-Product Attention (left). Multi-Head Attention (right) [10].

2.6.5 Residual Connections and Layer Normalization

To further enhance the model's learning capacity, Transformers incorporate residual connections and layer normalization. Residual connections bypass the standard neural network layers, allowing the original input to be added directly to the output of a layer. This helps prevent the degradation of information as it passes through multiple layers, making it easier to train deep networks. Layer normalization is applied after each sub-layer (attention and feed-forward), stabilizing and accelerating the training process [10].

2.6.6 Encoder-Decoder Structure

The Transformer architecture consists of both an encoder and a decoder. The encoder processes the input sequence, generating a set of continuous representations (or encodings). The decoder then uses these encodings to generate the output sequence. Each layer of the encoder and decoder includes multi-head self-attention and position-wise, fully connected layers. The decoder also includes encoder-decoder attention, allowing it to focus on relevant parts of the input sequence when generating each word in the output sequence [10], [58].

2.6.7 Model Training and Regularization

During training, Transformers use techniques such as Dropout and label smoothing to prevent overfitting and ensure robust performance. Dropout is applied to the outputs of sub-layers before they are added to residual connections, and label smoothing helps by providing softer targets during training, making the model less confident and more generalizable [10].

2.6.8 Efficiency and Parallelization

One of the primary advantages of the Transformer model is its ability to parallelize across sequences during training, significantly reducing training time compared to recurrent models. This efficiency is largely due to its reliance on self-attention mechanisms, which allow the model to consider all words in the sequence simultaneously rather than sequentially. This parallel processing capability is a key reason why Transformers have become foundational to modern NLP applications [10].

2.7 Decoder-Only Transformers

Decoder-only Transformers are designed to perform both the encoding of input and the generation of output within a single Transformer block. This architecture diverges from the traditional encoder-decoder models by relying solely on the decoder layer to process and generate text. In these models, masked self-attention is applied throughout the process, ensuring that each word in the sequence can only attend to the current and preceding words, thereby preserving the sequential nature of text generation [10], [58], [67].

As shown in Figure 2.11 the key function of masked self-attention in decoder-only architectures is to prevent the model from "seeing" future tokens during the generation process, which is crucial for maintaining the logical flow of language. As a result, the model can handle the input prompt and generate the output using the same mechanism, consistently maintaining the relationship between input and output [9].

In contrast to encoder-decoder models, which utilize separate units for encoding and decoding with different attention mechanisms, decoder-only models streamline these tasks into a single unit. In an encoder-decoder model, the encoder applies self-attention across all tokens in the input, while the decoder uses both self-attention and encoder-decoder attention to focus on relevant parts of the encoded input. Decoder-only models, however, apply masked self-attention uniformly across both input and output stages, simplifying the architecture and

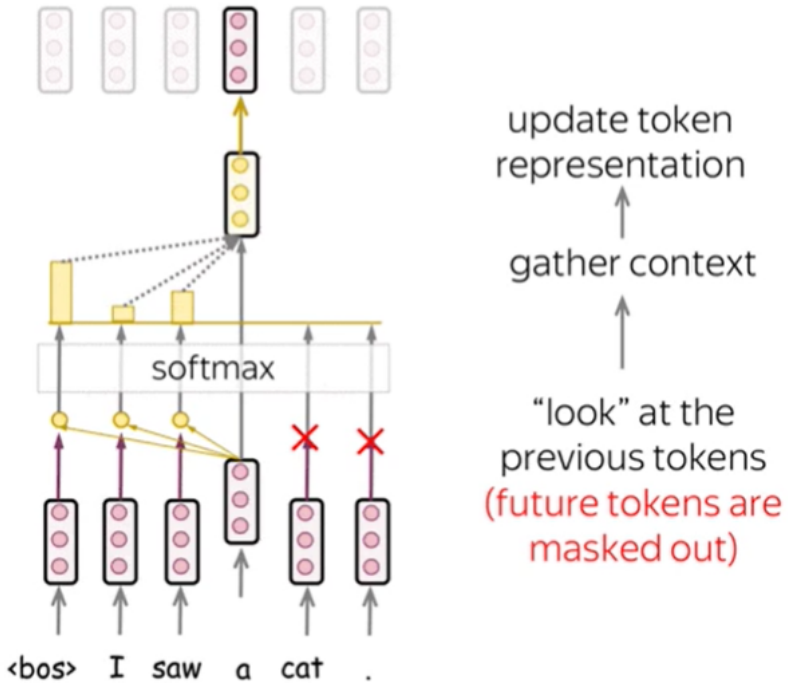


Figure 2.11: The mechanism of Masked Self-Attention [9].

making it particularly effective for tasks like language modeling, where sequential text generation is key [9].

By unifying the processes of encoding and generation, decoder-only Transformers like ChatGPT provide an efficient approach for handling tasks where the output is directly conditioned on the input. They leverage the simplicity and power of masked self-attention to generate coherent and contextually relevant text [67].

2.8 Encoder-Only Transformers

Encoder-only Transformers are specialized models designed to process input sequences by focusing on the connections between tokens within the input, without directly producing any output sequence. These models are particularly well-suited for tasks that demand comprehension and categorization rather than generation, such as text classification, sentiment analysis, and named entity recognition [58], [67].

In encoder-only architectures, such as those used in models like BERT [21], the model applies self-attention across all tokens in the input sequence. Unlike decoder-only models, which use masked self-attention to prevent future tokens from being considered, encoder-only mod-

els utilize bidirectional self-attention. This means that each token in the sequence can attend to all other tokens, both preceding and succeeding it, allowing the model to capture the full context of a word or token in relation to its entire surrounding text [67].

The encoder processes the input through multiple layers of self-attention and feed-forward neural networks, with each layer refining the model's understanding of the relationships between tokens. The inclusion of residual connections and layer normalization, as seen in the standard Transformer architecture, helps to stabilize the training process and enables the model to learn deep representations of the input data [67].

In tasks like masked language modeling, which is a pre-training task used in BERT, the model learns to predict missing tokens in a sequence by paying attention to the complete context provided by the surrounding tokens. This ability to leverage full bidirectional context is what makes encoder-only models particularly powerful for understanding and interpreting text [67].

By solely concentrating on the encoding process, encoder-only Transformers like BERT have become indispensable tools in NLP, especially in scenarios where understanding the meaning and relationships within the text is more crucial than generating new text. These models are frequently fine-tuned for specific tasks, making them adaptable and effective across a wide range of applications in natural language understanding [67].

The use of Transformers has become essential in various fields, demonstrating high performance in tasks such as machine translation, document summarization, named entity recognition, and speech-to-text. Additionally, they have made a significant impact beyond natural language processing, extending to areas such as biological sequence analysis, video comprehension, protein folding, and chess evaluation [58].

As the field progresses, new architectures like Mamba, introduced by Gu et al. [68], are emerging to overcome the limitations of traditional Transformers, particularly in handling long sequences and enhancing efficiency. Next, we will explore the training processes and optimization methods that drive these LLMs.

2.9 Training Processes for LLMs

Training LLMs involves multiple phases designed to ensure high performance across a wide range of tasks. These phases include Dataset Prepossessing, Pre-training and Fine-tuning, Self-supervised Learning, Transfer Learning, Hyper-parameters and Optimization, and addressing Challenges in Training LLMs.

2.9.1 Dataset Prepossessing

Before training LLMs, extensive dataset prepossessing is a crucial step. This process involves cleaning, normalizing text data, removing noise, and formatting the data to prepare it for model ingestion. The quality and diversity of the dataset significantly impact the model's ability to generalize across different language tasks. Datasets on a large scale are usually gathered from a diverse range of sources such as books, articles, websites, and other publicly accessible collections of texts to achieve comprehensive language representation.

2.9.2 Pre-training and Fine-tuning

LLMs typically undergo a two-phase training process: pre-training and fine-tuning. Pre-training allows the model to learn general language representations from large, diverse datasets, forming a broad understanding of general language patterns which enables it to comprehend and generate human-like text. Fine-tuning further enhances the model on smaller, task-specific datasets, improving its performance for particular tasks like sentiment analysis or question answering. The training process involves adjusting billions of parameters, such as weights and biases, to make accurate predictions for the next token in a sequence.

Three common learning models during this process are defined as:

- Zero-shot learning that enables base LLMs to handle different requests without specific training for each task.
- Few-shot learning which by providing a few examples achieves better performance in specific areas.
- Fine-tuning involves making further adjustments to parameters using additional data to enhance task-specific performance.

Interestingly, Li et al. [69], indicate that being larger isn't always better. Although bigger models generally have more capabilities, they also have downsides such as higher energy consumption and increased complexity. Smaller, well-optimized models like Microsoft's Phi-1.5 and Phi-2 have shown comparable performance to much larger models by using resources more efficiently. This finding aligns with newer approaches that highlight the importance of balancing model size with training data and computational efficiency [70].

2.9.3 Self-supervised learning

During pre-training, self-supervised learning is used as the main approach, where models undergo training on tasks such as masked language modeling, as seen in BERT, or next-token prediction, as used in Generative Pre-Training (GPT) [20]. In the case of masked language modeling, BERT is trained by predicting randomly masked words within a sentence, forcing the model to understand the context of the entire sequence to predict the missing word accurately. This method allows BERT to capture bidirectional context, making it particularly powerful in understanding complex language patterns.

On the other hand, next-token prediction, used in GPT, involves training the model to predict the next word in a sequence, given all previous words. This method trains the model to generate coherent and contextually relevant text in a unidirectional manner, making it efficient for tasks such as text generation and completion. By leveraging these methods, self-supervised learning enables models to learn from extensive amounts of unannotated text, making them versatile and powerful for a wide range of downstream tasks.

2.9.4 Transfer Learning

Transfer learning is a vital phase in LLMs' training process, in which knowledge from pre-training is transferred to new tasks during fine-tuning. It reduces the need for large labeled datasets for each task, minimizes computational resources, and improves model performance. Transfer learning's ability to generalize across tasks makes LLMs valuable in real-world applications with limitations on data and computational power.

2.9.5 Hyperparameters and Optimization

Hyperparameters like learning rate, batch size, and optimization techniques such as the Adam optimizer are critical in training LLMs. These factors influence how the model converges during training, affecting both speed and performance. Compute-optimal training has been demonstrated that performance can be improved by achieving a balance between model size and training tokens.DeepMind's research Chinchilla model in Training Compute-Optimal Large Language Models shows that a smaller model, when trained on the right amount of data, can outperform larger models that have been under-trained. This method challenges the assumption that increasing the model size alone guarantees better performance, highlighting the importance of efficiently using computational resources and training data [71].

2.9.6 Challenges in Training LLMs

Training LLMs require significant computational power, such as high-performance GPUs or TPUs. Challenges in this process involve the risk of overfitting, managing large-scale data, and ensuring generalization. To address these challenges, regularization techniques, early stopping, and more efficient training methods like parameter-efficient tuning, LoRA, QLoRA, and pruning are utilized [72].

PEFT methods aim to reduce computational costs by fine-tuning only a small subset of the model's parameters. This is particularly useful when computational resources are limited, enabling effective fine-tuning without the need to retrain the entire model [73].

LoRA (Low-Rank Adaptation) takes this idea further by introducing low-rank matrices that can be added to the model's layers. This method significantly decreases the number of trainable parameters, making the fine-tuning process more efficient while maintaining the model's performance. It is especially effective when maintaining the original model's structure and performance is important but with a reduced computational footprint [74].

QLoRA (Quantized Low-Rank Adaptation) builds on the principles of LoRA by integrating quantization techniques. This reduces the memory usage of the model during training and deployment, facilitating efficient operation on resource-constrained hardware. By quantizing the low-rank matrices used in LoRA, QLoRA enables a further reduction in the computational resources required without compromising model accuracy, making it a valuable tool for deploying large language models in environments with limited hardware capabilities. In Figure 2.12, the various approaches to fine-tuning and their associated memory for the process are illustrated. Here, QLORA enhances LoRA by implementing transformer model quantization to 4-bit precision and utilizing paged optimizers to manage memory spikes.

However, Qi et al. in Fine-tuning Aligned Language Models Compromises Safety, Even When Users Do Not Intend To! [75], discuss that fine-tuning aligned models introduces new safety challenges. Even with the use of benign datasets, custom fine-tuning can unintentionally reduce the safety alignment of models, increasing potential risks. Fine-tuned models may become more vulnerable to adversarial inputs, making them susceptible to harmful or unexpected outputs through minor manipulations in the input data. This vulnerability is particularly concerning in applications where precise and reliable outputs are crucial [75].

Another significant risk associated with fine-tuning is bias amplification. This process may not only amplify existing biases in the pre-trained model but could also introduce new biases, potentially resulting in discriminatory or biased outputs that reinforce harmful stereotypes.

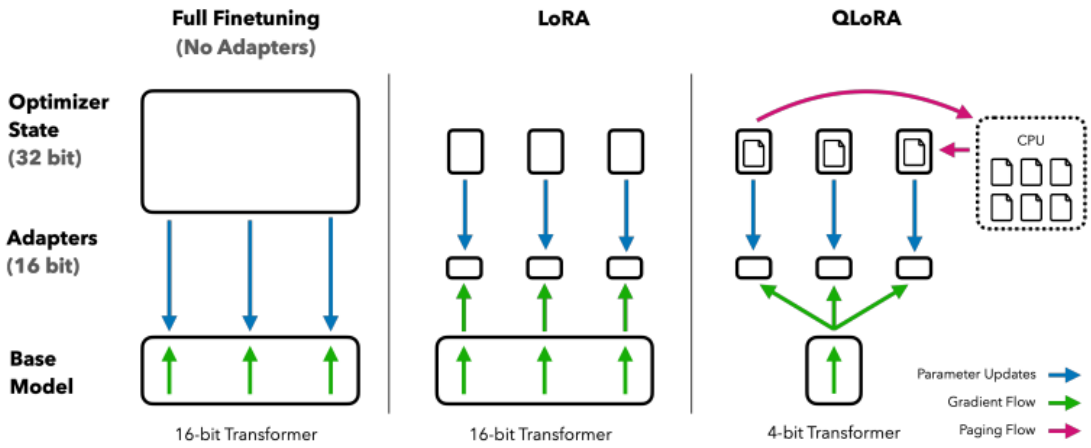


Figure 2.12: various approaches to fine-tuning and their associated memory [11].

This issue is especially concerning when fine-tuned models are deployed in environments where fairness and impartiality are important [75].

The degradation of safety alignment is another critical concern as well. Even subtle changes during fine-tuning can significantly diminish a model's ability to maintain safe and aligned outputs, especially in scenarios where the original model was carefully adjusted for safety. This degradation poses a significant threat when these models are applied in sensitive areas such as healthcare or legal systems, where maintaining alignment with ethical standards is crucial [75].

Additionally, the authors address instances of unexpected behavioral changes after fine-tuning, where models show undesirable behaviors, such as confidently providing incorrect information or generating toxic language. These behavioral shifts can have severe consequences, especially in high-risk applications where accuracy and reliability are essential [75].

These findings emphasize the need for rigorous safety evaluations and continuous monitoring throughout the fine-tuning process, ensuring that the benefits of fine-tuning do not compromise model safety and integrity.

2.10 Applications of LLMs

Recently, LLMs have revolutionized many fields by their integration in various domains which allows us to create applications in a wide range of general and specific scenarios. While these models have shown drawbacks and many vulnerabilities at the early stage, leveraging them generally has an impressive impact on the performance of downstream tasks. LLMs excel in many

traditional NLP tasks including word and sentence-level tasks, sequence tagging, information extraction, text generation, Knowledge base answering, text classification, and code generation. They have significantly improved tasks like machine translation, sentiment analysis, and text summarization. They are widely used in Information Retrieval (IR) systems, either as primary models or to augment existing IR models. They enhance the accuracy and relevance of search results by comprehending the context and semantics of queries better than traditional methods. These models also help in Recommendation Systems by generating recommendations that closely match user preferences. We can see their use on platforms like Netflix and YouTube. Moreover, these models combine and analyze data from various sources, including text, images, and videos which could be useful for tasks that involve combining multiple types of data, such as creating descriptive text for images or videos [76], [4], [50].

Additionally, we can find the use of LLMs in many segments such as healthcare, education, law, finance, and scientific research. Concerning healthcare, LLMs can help to analyze patient records, extract relevant information, and propose diagnoses or treatments based on the data. In education, they can help with customizing learning experiences, creating quizzes, and simplifying complex concepts, enhancing accessibility to education. LLMs can review and summarize legal documents, identify key clauses, and predict case outcomes. They can monitor transactions for fraudulent activities by detecting unusual patterns which is also known as fraud detection. LMs are being used more and more in scientific studies, helping with conducting literature reviews, forming hypotheses, data analysis, and even writing papers, thus making the research process more efficient in various fields [77].

Two other applications of LLMs include LLM-based agents and using LLMs for evaluating models or content. As LLM-based agents, they are equipped with memory, planning, and execution capabilities, which allow them to autonomously carry out tasks and adjust to user requests and environmental feedback. Additionally, LLMs are used to evaluate and give scores to other models or content through methods such as language-based evaluation and meta-evaluation [49]. Overall, as these models continue to advance, their applications are expected to cover more domain-specific areas, leading to innovation and improved efficiency across various industries.

2.11 Retrieval-Augmented Generation (RAG) Systems

Although LLMs have made impressive contributions to various fields, they have also shortcomings and failed to perform certain tasks or generate the desired outputs. For example, in

Question Answering tasks, where a pre-trained LLM is unable to produce accurate responses, techniques like fine-tuning and prompt engineering are used to enhance accuracy and generate more precise outputs. However, the computational difficulties and complexity of implementing such techniques have led to the development of new methods, such as Retrieval Augmented Generation (RAG), which offers an innovative approach to customizing and strengthening LLMs for specific tasks [78].

RAG has become a powerful method to enhance the capabilities of LLMs by integrating external knowledge sources into the text generation process. It is particularly effective for tasks that heavily depend on up-to-date knowledge, whereas traditional LLMs, relying only on the knowledge embedded in their parameters, often generate hallucinations due to outdated information or knowledge from proprietary documents that they have not trained on. We can describe RAG as a combination of the strengths of retrieval systems and generative models by first retrieving relevant information from a database or document repository and then using this retrieved information to generate the desired responses. We can break down this process into two main steps: retrieval and generation. We can utilize the RAG's ability to dynamically integrate current information for tasks requiring precise and up-to-date knowledge, such as open-domain question answering and fact verification. Gao et al. [12], in Figure 2.13 demonstrate the RAG process for the question-answering task.

The authors also categorize the evolution of RAG into three forms, each building upon the previous: Naive RAG, Advanced RAG, and Modular RAG. The basic form, or Naive RAG, simply retrieves and reads information, and often faces challenges such as irrelevant retrievals and hallucinations in the generated text. Advanced RAG improves accuracy and relevance through better indexing and context-aware retrieval techniques. The authors address Modular RAG as the most complex form, which uses specialized modules for different tasks, including iterative and adaptive retrieval, as well as the integration of knowledge from various data formats, such as structured knowledge graphs. This advanced implementation of RAG allows us to process information more precisely and efficiently in sophisticated domains [50]. Huang et al. [77], discuss new techniques like Dense Knowledge Similarity (DKS) and Retriever as Answer Classifier (RAC) that further enhance RAG's capabilities.

Concerning the applications of RAG, we can find them highly effective in domains where retrieving accurate and detailed information is crucial. It excels in Question Answering, particularly in specialized domains like medical or legal contexts, by answering complex questions accurately. It also enhances Information Retrieval by improving search results and aids in Content Generation, such as automated reporting and educational content creation, ensuring that

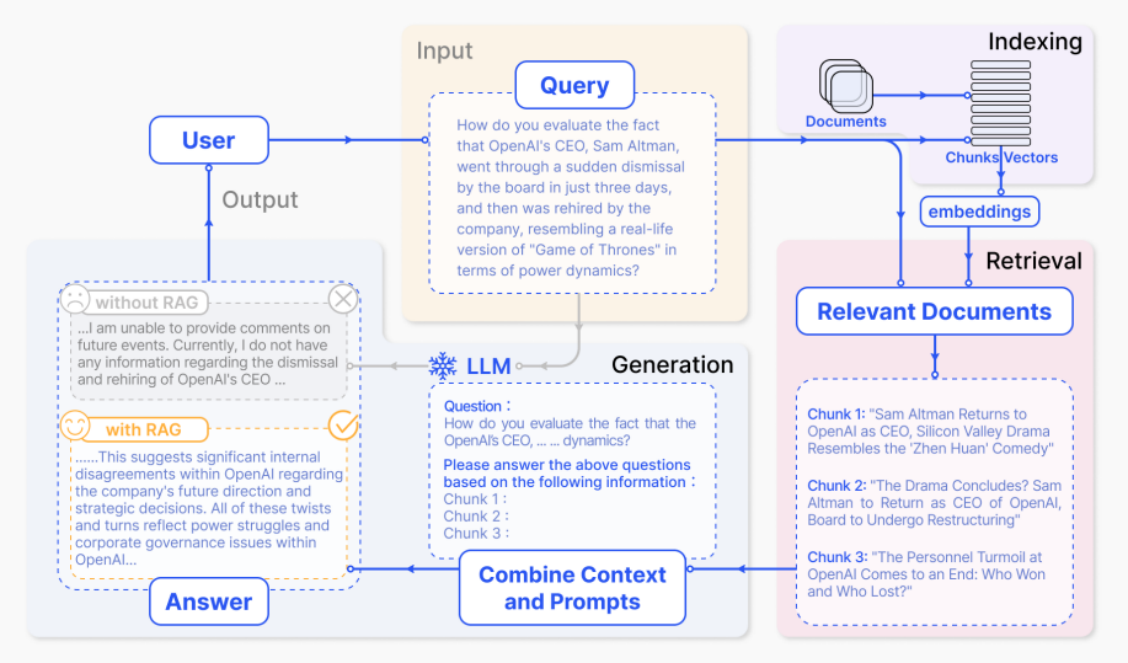


Figure 2.13: An example of the RAG process used for question answering. It primarily involves three stages. 1) Indexing: Breaking down documents into segments, converting them into vectors, and saving them in a vector database. 2) Retrieval: Selecting the top k segments most closely related to the question using semantic similarity. 3) Generation: Entering the original question and the selected segments into LLM to produce the ultimate answer [12].

the generated content is factually correct and reliable by providing sources for the generated text [78].

Despite the strengths of RAG systems we have discussed so far, they face challenges related to retrieval quality, integration complexity, latency, efficiency, and scaling issues. The effectiveness of RAG heavily depends on the quality of the retrieval process. inaccurate or low-quality data can lead to imperfect results. Additionally, combining retrieved data with generated text in a coherent and contextually suitable way is difficult and often requires advanced methods. Moreover, like the fine-tuning technique, the retrieval and generation processes can result in computational difficulties, that can impact the model's speed and efficiency. Finally, since RAG systems rely on large-scale external databases, the challenge of scaling these systems for broader applications remains substantial. Future research aims to refine these processes, enhance robustness, and explore hybrid models that combine RAG with other techniques, such as Knowledge Graphs, to address these challenges [78], [12].

2.11.1 Vector Databases in RAG Systems

This section will begin by focusing on the Vector Database, an important element of RAG systems. We will then delve into the concept of poisoning vector databases, which serves as the foundation of this paper. In RAG systems, the vector database is where recent data or proprietary documents, not covered in the training of LLMs, are initially stored. Whenever a query concerns this information, it is directed to the vector database, which retrieves the relevant text data. This retrieved text is then included in the prompt for the LLM, providing context that assists the model in generating accurate responses [79].

We define a vector database as a mathematical representation of data that has been crucial for semantic search applications even before the emergence of generative AI. These applications focus on understanding the meaning of words or phrases rather than performing keyword searches for exact matches. Additionally, vector databases have broader use cases than RAG systems like providing solutions to challenges faced by LLMs. They can help mitigate issues related to outdated knowledge and hallucinations by enabling real-time updates and acting as a memory layer that improves the accuracy of generated responses. However, vector databases come with specific challenges, such as the need for efficient dimension reduction techniques to handle high-dimensional vector data and the complexity of managing conflicting information from several knowledge bases [80], [81].

When we are going to select a vector database for RAG systems, there are several factors that

we need to take into account Scalability and Performance, Indexing and Query Efficiency, Cost and Resource Efficiency, and Advanced Features [80], [81].

For example, vector databases like Milvus and Pinecone due to their excellent scalability and easy deployment, play a vital role in managing large datasets and real-time applications. Furthermore, their support for multi-modal data increases their usefulness in complex environments. For every RAG system efficient indexing and being able to quickly generate query responses are taken as an important element, particularly in real-time applications like chatbots. However, these databases often need hybrid search algorithms to integrate vector and traditional database operations for comprehensive search results. The selection of vector databases can significantly affect the RAG system's overall cost and resource usage. Some databases provide efficient resource usage without sacrificing performance as well they can act as a cost-effective semantic cache, reducing API expenses and enhancing response times by reusing previous queries [80], [81].

Jing et al., in their recent research, address advanced features of vector databases, such as dynamic updates and hybrid search capabilities, which combine vector-based search with traditional keyword search. These features improve the flexibility and functionality of RAG systems, allowing them to handle a broader range of queries more effectively. The authors identify knowledge conflict as a significant challenge when combining several knowledge bases. Thus, robust conflict resolution strategies are essential for managing conflicting information from various sources to ensure accurate outputs [81].

Understanding these factors helps us optimize RAG systems to align with our specific needs, whether for knowledge retrieval, content generation, or complex data analysis tasks. The improvement of vector databases and their integration with LLMs can have a significant impact on tackling AI challenges, leading to the development of more robust AI systems.

2.12 RAG Poisoning Attacks

As we discussed so far, RAG is broadly used across various fields to mitigate constant fine-tuning issues and is introduced as a solution to many LLM drawbacks. However, the integration of RAG with LLMs in AI systems is increasingly threatened by RAG poisoning, which poses a significant risk to tasks that are supposed to generate highly accurate outputs. This type of attack involves injecting malicious or misleading data into the knowledge databases that RAG systems rely on, causing the model to generate inaccurate, biased, or even harmful outputs. RAG poisoning exploits the dependency between LLMs and external data sources

by modifying the information that the model retrieves and uses during the generation process, resulting in compromised responses [82].

We can define the mechanism of RAG poisoning as knowledge injection and prompt manipulation. In knowledge injection, attackers insert a small number of poisoned texts into the knowledge database, when the RAG retrieves the text it generates responses based on this corrupted information [83]. On the other hand, prompt manipulation involves creating malicious prompts that indirectly influence the RAG process, leading to the generation of harmful responses [84].

In addition to these poisoning methods, the introduction of vector databases in RAG systems increases additional security concerns. Vector databases store embeddings derived from private data, making them vulnerable to inversion attacks that could expose sensitive information. Moreover, RAG workflows can unintentionally lead to the sharing of documents, especially when these databases lack the necessary access controls. This, combined with the risk of LLM log leaks and the potential for poisoning attacks during the query process, emphasizes the need for implementing robust security measures to mitigate the evolving threats posed by RAG poisoning [85].

2.13 Summary

So far, we have seen an overview of the development and evolution of Large Language Models and their applications. I discussed the transition from early neural networks to advanced transformer-based models like GPT and BERT. Then we learn about the computational challenges and risks associated with fine-tuning these models. Also, an introduction to RAG systems is given, and we understand how these techniques can enhance LLMs by integrating external knowledge sources through vector databases. Finally, we discovered the emerging threat of RAG poisoning attacks, where malicious data can lead to compromised outputs, emphasizing the importance of security measures in RAG-based systems. This forms the basis for delving into related security issues in machine learning.

3
Related Works

We are already familiar with the significant performance of these large models in various tasks, however, they come with drawbacks, including biases in their training data that can lead to biased responses and unfair outcomes. Gallegos et al. [86], formally define biases and fairness in LLMs, categorizing them into representational and allocational harms. The authors also systematically categorize bias mitigation techniques across stages, emphasizing key strategies.

Another significant problem that has received less attention is the substantial computational challenges these models present, particularly concerning energy consumption and costs during the training and deployment phases. Samsi et al. [87], address this issue in the context of LLMs, discussing the trade-offs between model size, performance, and energy usage. These insights are crucial for understanding the broader impacts of implementing LLMs on a large scale, demonstrating the need for more energy-efficient methods in their development and deployment.

Data privacy is another major concern. As these models are trained on vast amounts of data, there is always a risk of leaking sensitive information, potentially compromising privacy and raising security issues. Yan et al. [88], provides a comprehensive survey on data privacy issues in LLMs, focusing on both passive privacy leakage, where sensitive information may be exposed during model usage, and active privacy attacks, where adversaries exploit vulnerabilities to extract private data. To mitigate these risks, the authors review privacy protection mechanisms such as differential privacy, federated learning, and homomorphic encryption across pre-training, fine-tuning, and inference stages.

Vulnerability to adversarial attacks has also raised significant attention in recent years. Shayegani

et al. [89], discuss how safety-aligned LLMs can still be vulnerable to attacks such as jailbreaks. In their comprehensive survey, they address vulnerabilities including jailbreak and prompt injection attacks, classifying these attacks based on the learning structure and the attacker's level of access to the model. They highlight the ongoing challenge of defending LLMs despite safety training. The authors also explore attacks on multi-modal models and systems that integrate LLMs into more complex frameworks.

Liu et al. addressed adversarial attacks, including prompt injection, where malicious instructions are inserted into prompts to change LLM behavior, or harmful information is added to the context to make the model produce specific outputs [90]. Several researchers, including Liu et al. [91]; and Li et al. [92], have discussed the resistance of ChatGPT against jailbreak prompts, revealing its vulnerabilities to such attacks and demonstrating how prompts can evade restrictions in various scenarios.

Shafahi et al. [93], discussed the impact of data poisoning attacks on machine learning models, where attackers insert carefully crafted examples into the training set to change the model's behavior without controlling data labels. They demonstrated that using only a few poisoned examples can significantly modify a model's predictions on targeted instances, particularly in transfer learning.

Years later, Zhou et al. [94] explored the effects of poisoning just 1% of instruction tuning samples, showing that it can result in an 80% Performance Drop Rate (PDR). This finding highlights the critical need for stronger defenses against data poisoning attacks on LLMs. The authors address this vulnerability by introducing a novel gradient-guided approach to identify backdoor triggers that evade traditional defenses while maintaining content integrity.

As LLMs have become more widely used, many vulnerabilities to adversarial attacks have been investigated. Researchers have made significant efforts to show these models' weaknesses across various attack scenarios. Over time, many methods have been introduced to mitigate the security issues of LLMs, alongside addressing more common challenges such as hallucinations. One promising approach to improving the precision of LLM responses is the use of RAG. While we see RAG as a technique to address these issues, it is also vulnerable to security challenges.

During my research, this area has received relatively less attention. However, some foundational studies have influenced me to conduct experiments in this area. For example, Luo et al. [95], explore the phenomenon of catastrophic forgetting (CF) in LLMs during continual fine-tuning. The authors explain that CF occurs when a model forgets previously learned information as it learns new information. They find that the extent of forgetting increases with

more extensive fine-tuning, particularly in larger models ranging from 1B to 7B parameters. They also propose strategies to mitigate this issue, such as general instruction tuning, as seen in ALPACA, which can help reduce the effects of CF and even mitigate language biases.

The works discussed so far illustrate the vulnerabilities of LLMs, which we can also explore in the context of RAG-based systems. These findings motivated me to design an experiment focused on exploiting misinformation in the external knowledge base of RAG-based systems and studying how the model struggles with this issue. In the following chapter, I will delve into the details of my methods and present the results of these experiments.

4
Methodology

4.1 Introdcution

This chapter presents the methodology and results of the experiments conducted to explore the vulnerability of LLMs when using RAG-based systems, specifically focusing on the impact of poisoned vector databases. We begin by explaining the methodology used to conduct the experiments, including a detailed description of the techniques and tools used. Next, we will present the evaluation methods applied to analyze the results. Finally, we will thoroughly discuss the findings of the experiments, as presented in the corresponding tables. Before diving into the methodology, we briefly revisit the research questions and objectives to clearly understand the study's focus and the logic behind the experimental design.

4.2 Methodology

We conducted this experiment on a small scale, primarily due to the reasons discussed in previous chapters, where it was noted that small-scale models are more frequently utilized than larger ones, especially by smaller companies aiming to leverage such models to simplify their tasks. In this study, we integrated these smaller models with RAG to examine how a malicious knowledge base, containing misinformation, could affect the outputs generated by these models.

The experiments were carried out in multiple stages. First, we focused on model selection, providing reasoning behind the choice of the LLMs used in the experimental phase. Next, we created our knowledge-based data store including misinformation within the knowledge base.

The main idea of the experiment is to develop a question-answering system. In the background chapters, We studied various techniques for fine-tuning LLMs to improve response accuracy. Moreover, we explored optimization tools and techniques such as PEFT, LoRA, and QLoRA, so these will not be re-explained here.

The response generation process is conducted in two phases. In the first phase, we used the baseline model, while in the second phase, we employed the same models after fine-tuning. Each phase consisted of two stages, in the first stage we utilized a benign data store to retrieve information and enhance responses, and in the second we used a data store that included documents containing misinformation.

4.2.1 Dataset Prepossessing

Choosing the right dataset to fine-tune models is important as it directly impacts the performance of LLMs. High-quality data plays a key role in generating accurate responses and preventing models from hallucinating. Among the different types of datasets available, such as instruction datasets, Raw completion datasets, preferences datasets, and others, instruction datasets are particularly suited for supervised fine-tuning. In these datasets, the input and output fields are structured as instructions and expected responses, suitable for a Question Answering system.

There are numerous instruction datasets available on platforms like Hugging Face and Kaggle, created by various organizations and individuals. For example, OpenOrca, with over 2.9 million rows and 2.85 GB in size [96], and Dolphin, a replication of Microsoft Orca [97], with around 3.7 million rows and 3.6 GB in size [98]. Due to the large size and computational costs associated with these datasets, we chose to use a smaller-scale dataset named Open-Platypus [99].

Open-Platypus dataset

The Open-Platypus dataset, developed by Lee et al., is a subset of 11 open-source datasets, carefully selected to enhance LLMs performance in STEM and logic domains. It consists primarily of human-designed questions, with around 10% of the questions generated by LLMs. The dataset was carefully filtered to avoid redundancy and contamination, ensuring no overlap be-

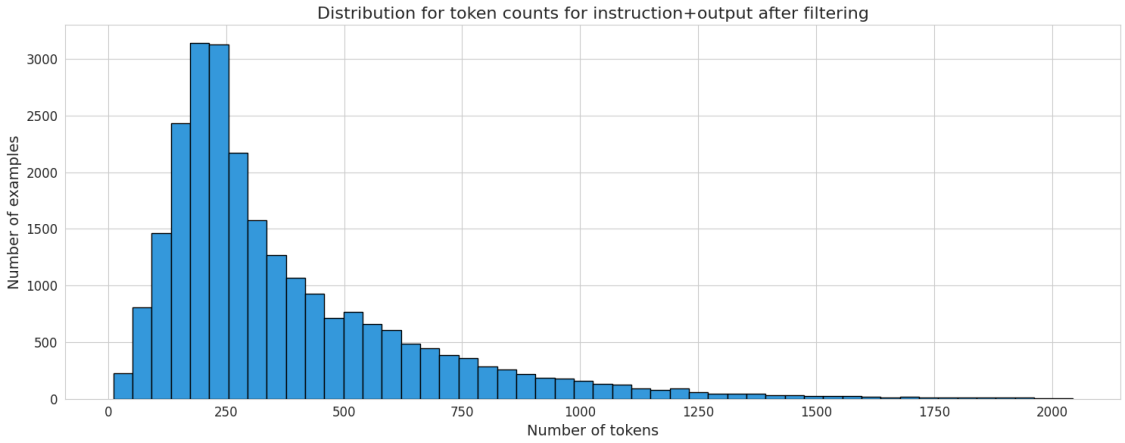


Figure 4.1: Distribution for a total token count for instruction and output for Open-Platypus dataset

tween training and test data. The data is formatted using the Alpaca instruction-tuning format, which includes clear instructions and expected outputs, making it ideal for fine-tuning models to generate accurate and relevant responses [100]. Due to its manageable size and high quality, this dataset is a practical choice for fine-tuning my chosen LLMs without incurring excessive computational costs.

The Open-Platypus dataset contains approximately 30k rows and is 15.6 MB in size, making it relatively small compared to other datasets. To ensure that fine-tuning remains practically feasible, we selected the first 1k rows for processing. After plotting the distribution of tokens, we filtered out rows with more than $2,048$ tokens to keep the dataset manageable while still providing high-quality data for fine-tuning. In Figure 4.1, the distribution of the combined instruction and output tokens is shown.

Next, to remove near-duplicate samples based on semantic similarity, we used the Sentence-Transformer model to convert text into high-dimensional embeddings, which were then normalized for cosine similarity-based searches. A FAISS Index with IndexFlatIP was created to efficiently identify and filter out samples where the similarity exceeded a threshold of 0.95, eliminating samples that were too similar to each other [101]. This process resulted in a deduplicated dataset, preserving unique samples and preventing redundancy. For embedding, we selected the thenlper/gte-large model from the Hugging Face leaderboard due to its high ranking among top embeddings [102].

Since the dataset was still large, we sorted the samples based on the total number of combined instruction and output tokens in descending order, prioritizing the most content samples. Finally, we selected the top 1k samples to create the new, pruned dataset.

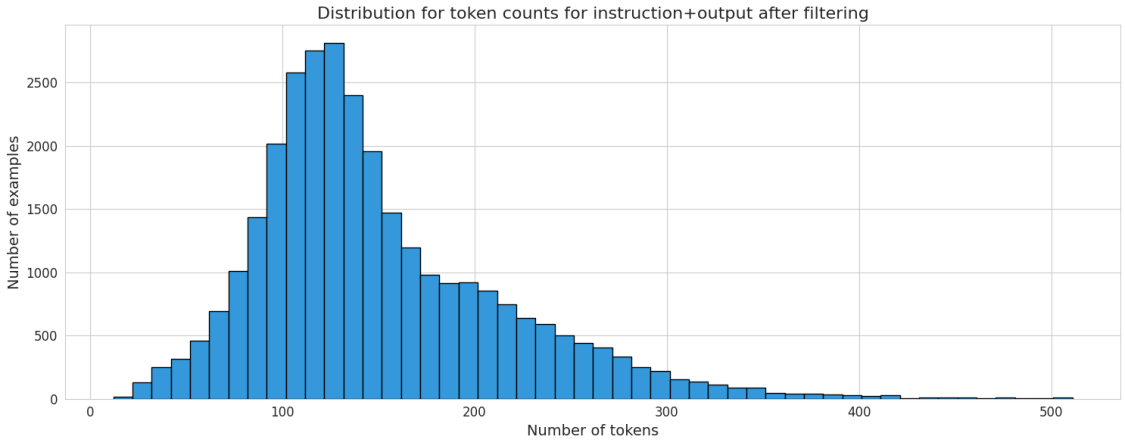


Figure 4.2: Distribution for a total token count for instruction and output for arxiv-physics-instruct-tune dataset

ARXIV-PHYSICS-INSTRUCT-TUNE DATASET

Since a more specific domain was needed for my experiments, we chose physics as the primary domain to fine-tune the LLMs. We will discuss the experimental domains further in the next section. As with the Open-Platypus dataset, we used the Hugging Face platform to explore available datasets. Among the options, the arxiv-physics-instruct-tune-30k dataset from Artifact AI, with approximately 30k rows and 21.8 MB in size, was a suitable choice [103]. We applied the same filtering process as before, tailoring it to a dataset of the top 1k samples. In Figure 4.2, the distribution of the combined instruction and output tokens is shown.

Malicious Dataset Construction

We have already discussed the importance of integrating RAG into applications based on LLMs. To effectively build my RAG system knowledge base, it is crucial to focus on domains where LLMs, whether baseline or fine-tuned, struggle to generate desired outputs. This challenge is often found in complex and advanced topics, such as quantum physics, and recent publications in deep learning, like the Mamba architecture. These topics likely contain information not included in the model's training data. Therefore, our knowledge base consists of recently published papers in physics, astrophysics, and AI, forming a solid knowledge base for our experiments.

To test how our models handle misinformation within the knowledge base, we deliberately modified the content of several selected papers. Gioele Collu et al. [104], demonstrated how LLMs could be manipulated to bypass their safety mechanisms, generating prohibited responses.

Likewise, we used GPT-4 to alter the content of these papers, introducing misinformation to simulate an attack scenario where incorrect information is integrated into the knowledge base. This modification varied in degree, ranging from slight changes to substantial alterations, and included both correct and incorrect information. This scenario reflects potential real-world threats, where attackers might insert misleading documents into a knowledge base to deceive users into believing they are receiving accurate and up-to-date information.

For practical implementation, we created 100 questions, 10 general questions related to the topics, and 90 questions drawn directly from the selected benign papers in the data store. Given the complexity of these topics, we used GPT-4-powered ChatGPT to generate the questions and the corresponding correct answers.

4.2.2 Model Selection

When selecting the models for my experiments, we had a wide range of options for the choice of LLMs. Given the variety of models with different parameter sizes being published regularly, we focused on selecting models from the open-source community for the reasons we previously mentioned. Some of these models include Large Language Model Meta AI (Llama) family by Meta AI, Gemma Open Models by Google AI, Mistral models by the Mistral AI team, and OpenHermes by Nous Research, each offering unique advantages.

To select the model, we referred to several benchmarks that rank these models based on performance. One of the most comprehensive resources is the Hugging Face platform, which features various LLM leaderboards, including the Open LLM Leaderboard [105]. Additionally, platforms such as Meta-Bench and Chatbot Arena [106] also evaluate and rank LLMs based on their criteria. Besides these, we studied the results of papers related to my project, focusing on the models they selected for their experiments and investigating them based on performance and vulnerability to malicious behaviors.

After browsing numerous benchmarks and studying various papers, two models stand out as the most suitable for my project, the Llama 2 family models and the Mistral models. These models were selected based on their outstanding performances in multiple benchmarks, making them ideal candidates for my experiments.

Mistral 7B

Mistral 7B is a 7.3 billion parameter language model created for high performance and efficiency, surpassing larger models like Llama 2 (13B) in reasoning, mathematics, and code gener-

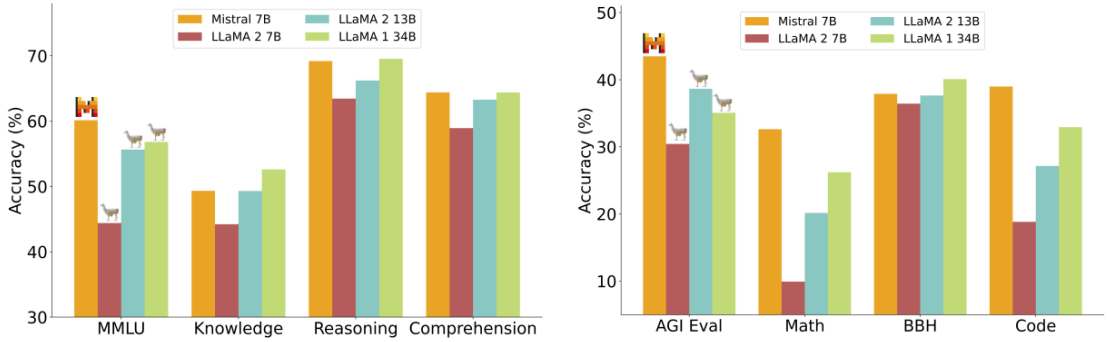


Figure 4.3: The Mistral 7B and different Llama models across various benchmarks based on their performance compared to each other [13].

ation on all benchmarks, and Llama 1 (34B) on many benchmarks [13]. It is based on a transformer architecture featuring 32 layers, a hidden dimension of 14, 336, and 32 attention heads. The model uses advanced attention mechanisms, including Grouped-Query Attention (GQA) for enhanced inference speed and Sliding Window Attention (SWA) to efficiently process long sequences. These innovations allow Mistral 7B to maintain high performance while optimizing computational resources, making it suitable for real-time and large-scale applications. The Mistral 7B Instruct model excels in instruction-following tasks, showing its adaptability and capability in various domains [107]. In Figure 4.3, we can see the overall performance of Mistral 7B compared to Llama models across various benchmarks.

To conduct the experiments with the Mistral model, we used the Mistral-7B-Instruct-v0.2 model [108], which is fine-tuned on instruction datasets and is comparable to 13 billion-parameter chat models. It is of interest that it surpasses all other 7 billion-parameter models on the MT-Bench benchmark [107].

Llama 2

The Llama 2 family of language models, developed by Meta AI, ranges from 7B to 70B parameters. It includes Llama-2-chat variants fine-tuned for dialogue using supervised fine-tuning (SFT) and Reinforcement Learning with Human Feedback (RLHF). These models exceed the performance of previous Llama 1 models and other open-source models, especially in safety and helpfulness benchmarks. Like Mistral, Llama 2 models are built on a Transformer architecture and incorporate improvements such as increased context length and GQA [14]. They are trained on a large corpus of publicly available data, 2 trillion tokens, which is 40% more data than Llama 1 [109]. Meta AI has made these models available for research and commer-

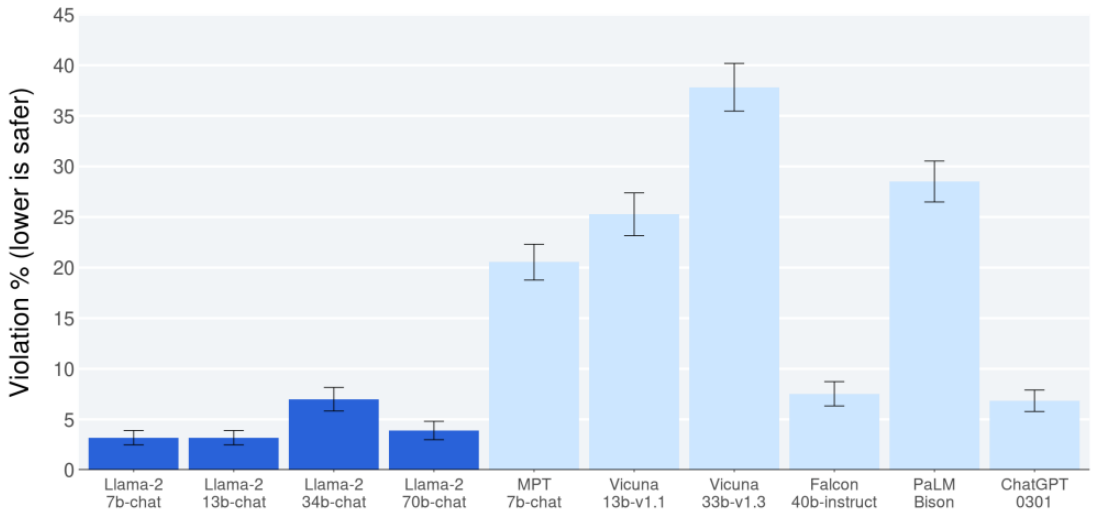


Figure 4.4: The performance of Llama-2-chat compared to other models through safety assessments using around 2,000 adversarial prompts. These findings emphasize the performance of Llama-2-chat [14].

cial use. These models demonstrate competitive performance against models like GPT-3.5 and GPT-4 [14]. In Figure 4.4, we can find the demonstration of the safety evaluation results for Llama-2-chat compared to those of both open-source and closed-source models.

After careful consideration, we selected the Llama-2-7b-chat-hf [110], and Llama2-13b-chat-hf [111] models for my experiments. These fine-tuned models, with 7 and 13 billion parameters respectively, are specifically developed for dialogue use cases and are adjusted for use with the Hugging Face Transformers format.

Furthermore, it is important to note that we considered only small-sized LLMs due to the practical feasibility of examining their behavior in generating responses, whether using the baseline versions or the fine-tuned ones with the RAG method. Given the constraints of computational resources and the aim to make these experiments accessible to a wider audience, we utilized Google Colab GPUs to conduct these experiments.

4.2.3 Fine-tuning

To implement the fine-tuning phase, we used the preprocessed dataset to adapt the selected LLMs, focusing on the specific domains addressed in the dataset preprocessing section. To ensure that the models were trained on domain-specific data of interest, we employed SFT to adapt them using data less likely to have been included in their original training. It is of interest to mention that Zhou et al. in their work "Llama Less Is More for Alignment," demon-

strated the effectiveness of fine-tuning a 65B parameter model, LLaMA v1 (65B), using a small, high-quality curated dataset to achieve performance comparable to large models like GPT-4, challenging the traditional large-scale tuning approach [112].

Due to the limited VRAM of the T4 and L4 GPUs available on Google Colab, we used the LoRA and QLoRA fine-tuning techniques, which involve quantizing the model to 4-bit precision to reduce memory usage and allow it to fit within the available VRAM, ranging from 15 to 22 GB depending on the GPU on Google Colab. The fine-tuned models were subsequently uploaded to the Hugging Face platform.

We used the Transformers library by Hugging Face [113], for handling the models and managing the fine-tuning processes. This library provides essential tools for NLP tasks, including pre-trained models, tokenizers, and other utilities that simplify the fine-tuning process in this project.

For the fine-tuning configuration with regards to LoRA, we set low-rank matrices (r) with the value 16, a lora_alpha value of 32, and a dropout rate of 0.05 (lora_dropout=0.05) to minimize the risk of overfitting. Moreover, the model was also prepared for low-bit training by casting certain layers to fp32, ensuring stability during the process.

Regarding the training parameters, due to the constraints of Colab, we limited the number of epochs to 1 (num_train_epochs), with a batch size (per_device_train_batch_size) of 10. We used the Adam optimizer, specifically paged_adamw_8bit, to reduce memory usage, by initializing the learning rate (learning_rate) at a small step size of $2e-4$. The training loss charts for fine-tuning Llama2-7B-chat-hf and Mistral-7B-Instruct-v0.2 on the physics dataset are shown in Figure 4.5 and Figure 4.6.

In the final step, we fine-tuned the selected models on both datasets to prepare them for integration with the RAG implementation. This enabled me to query the models using the designed questions and analyze the responses generated by these customized models.

4.2.4 RAG Implementation

RAG is a way of context augmentation technique, that combines context with LLMs at inference time. Various libraries are available to implement the RAG system, including open-source frameworks like LlamaIndex and LangChain. For these experiments, we chose LangChain, a widely used and prominent framework for developing applications that use LLMs available in Python and JavaScript libraries. It simplifies the development of LLM applications by providing abstractions that represent the essential steps and concepts needed to work with language

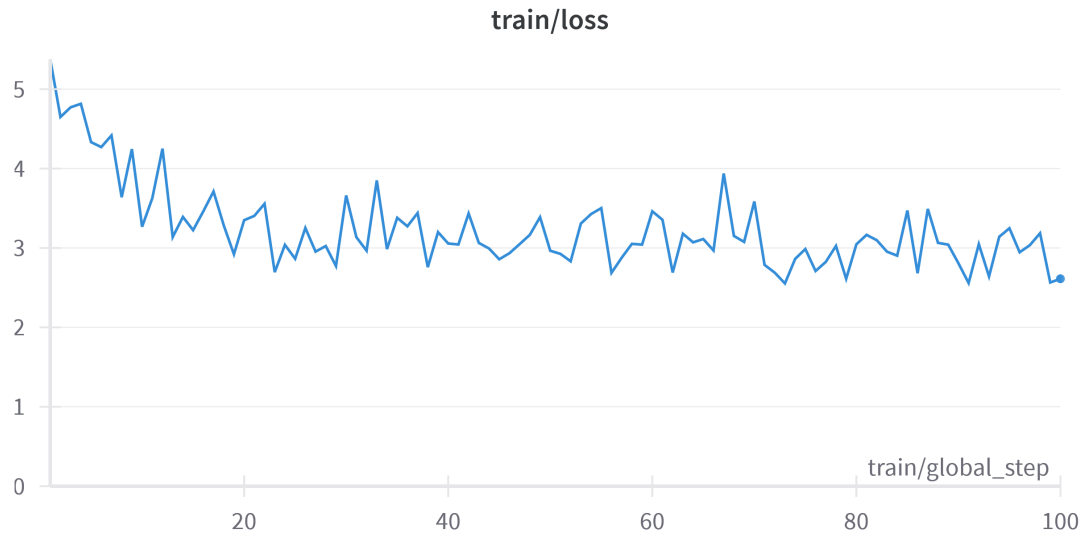


Figure 4.5: Train and loss chart regarding Llama-2-7B-chat-hf fine-tuned on arxiv-physics-instruct-tune dataset

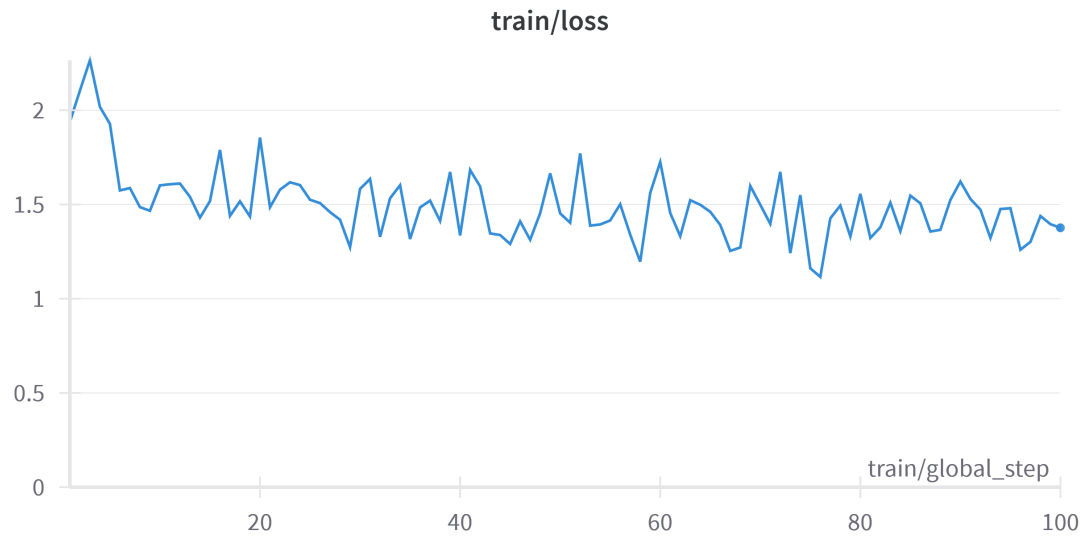


Figure 4.6: train and loss chart regarding Mistral-7B-Instruct-v0.2 fine-tuned on arxiv-physics-instruct-tune dataset

models. These abstractions can be combined to build applications, reducing the amount of code required to perform complex NLP tasks.

Introduction to LangChain

LangChain's components include an LLM module, which provides a standard interface for nearly any LLM, and a Prompts module, which formalizes the composition of prompts, eliminating the need to manually hard code context and queries. The Chains component combines LLMs with other components, enabling applications like question-answering by executing a sequence of functions.

The Index component plays an important role by acting as a sort of documentation system that allows LLMs to access specific external data sources not included in their training data, supporting Document Loaders, which import data from sources like file storage services, web content, or databases. Vector Databases, and Text Splitters that break down text into smaller, semantically meaningful chunks. The Memory component adds long-term memory to applications, retaining or summarizing prior conversations. Finally, agents in LangChain use LLMs as reasoning engines to determine which actions to take. We can utilize the LangChain framework for use cases such as chatbots, summarization, question-answering, data augmentation, and virtual agents [114].

4.2.5 The Experimental Design

To justify the implementation of these experiments, it is important to consider that external knowledge bases can be easily stored on private cloud storage within an organization's infrastructure or via public cloud storage and file-sharing services like Microsoft OneDrive, Google Drive, Mega, or Dropbox. However, Bashir Shaikh et al. discussed common security risks of such platforms [115]. In this project, we conducted experiments by altering parts of these external resources in the form of PDFs accessed through Dropbox. These PDFs contained misinformation in specific, up-to-date domains such as Physics and AI, including newly published papers that the models likely had not been trained on.

These malicious data were injected into vector databases, and the LLMs' responses to queries based on these compromised knowledge bases were evaluated. This approach aimed to uncover specific patterns and conditions under which LLMs are most vulnerable, contributing to the development of more robust and secure AI systems and more specifically RAG-based systems. To test the accuracy of the models, the experiments were divided into two parts: the first used

a knowledge base containing benign documents, while the second included documents with embedded misinformation. We downloaded and loaded papers from Dropbox, split them into text chunks, embedded these chunks, and stored them in a vector database. The models were then queried to retrieve relevant information from this data store, which was integrated into the prompts to generate responses. The models used in these experiments included Llama-2-7B-chat-hf, its fine-tuned version on the mentioned datasets, Mistral-7B-Instruct-v0.2, its fine-tuned version on the same datasets, and Llama-2-13B-chat-hf. To classify the responses as correct or incorrect, we used GPT-4o to check the accuracy of the models' outputs when retrieving information from both benign and malicious sources.

4.2.6 Evaluation Metrics

Accuracy

Accuracy is one of the most common metrics in machine learning, measuring the proportion of correct outputs. It represents the percentage of accurately predicted responses out of the total generated responses, providing a general overview of each model's performance. However, accuracy alone may not be sufficient, especially in cases of unbalanced data, where the majority of results may be positive or negative. This is why additional metrics such as precision, recall, and F1 score were also considered [116]. We compute the accuracy metric as

$$\text{Accuracy} = \frac{\text{Number of Correct responses}}{\text{Total Number of generated responses}}. \tag{4.1}$$

Other Metrics

Precision evaluates the proportion of true positives among all positives identified by the model, while recall measures the proportion of actual positives that were correctly identified by the model, highlighting the model's ability to capture relevant instances [116]. The F-measure or F1 score is the weighted harmonic mean of precision and recall, providing a balanced measure when both metrics are important. It is considered a trade-off between precision and recall, offering a single metric that balances these two aspects [117].

Reasoning on evaluation method

However, given that the focus of this study is on the accuracy of generating incorrect responses, accuracy was the primary metric that I selected. We computed this metric for each model's

performance over two experimental setups, one using a benign knowledge base and the other using a knowledge base containing misinformation. By comparing these results, we assessed how well the models generated correct responses, even when faced with misleading information in the vector database. This evaluation is critical for understanding the models' resilience in adversarial scenarios, where an attacker may attempt to manipulate the data within the external knowledge base, and for assessing the overall reliability of RAG-based systems.

4.2.7 Experiment Execution

After downloading and storing the selected documents, we set up the LLM using the Hugging Face pipeline. All models were loaded in 8-bit precision to make the experiments feasible, except for the Llama-2-13B-chat-hf model, which was loaded in 4-bit precision due to its larger size and to investigate the difference in generated responses compared to other tests. The pipeline was configured uniformly across all experiments, with specific adjustments for the Llama-2-13B-chat-hf model.

The pipeline was configured for the text generation task, with a maximum response length (max_length) of 1024 tokens and a limit of 512 tokens for the generated output. The temperature was set to 0.1 and top_p to 0.95, ensuring coherent, precise, and varied scientific responses. For the Llama-2-13B-chat-hf model, we decreased top_p to 0.5 to make the responses more deterministic and to observe the effects of lower values for these parameters. Overall, this configuration provided accurate answers, while the settings for top_p and repetition_penalty preserved clarity and minimized redundancy without compromising language quality.

Regarding text processing, each chunk contained 1,000 tokens with an overlap of 200 tokens (chunk_overlap). For embeddings, we used the bge-base-en-v1.5 model [118], which is among the top-ranked embedding models on the Hugging Face embedding leaderboard [102], and well-suited for limited resources due to its small size which is around 438MB. The smaller size of the model allowed for faster inference times and reduced VRAM usage, which made it perfect for running on the GPUs available in Google Colab. Furthermore, normalize_embeddings is set to true to improve the cosine similarity search.

For the vector database, we utilized Chroma DB, an open-source vector store designed for storing and retrieving vector embeddings. Its main purpose is to store embeddings and related metadata for further use by LLMs. It also can be used as a semantic search engine for text data [119].

In retrieving documents, the top 5 contexts were returned from the vector store. The chain

type was set to ”stuff,” in which the entire content of the retrieved documents is included directly in the prompt. In the end, we queried all the questions, and the generated responses were stored in two formats, JSON and CSV, for further analysis. It should be noted that to enhance the accuracy of the generated responses, we modified the system prompt to improve the responses.

5 Experimental Results

5.1 Results

We implemented the RAG method on several models, Llama-2-7B-chat-hf, Llama2-13B-chat-hf, and Mistral-7B-Instruct-v0.2. For the implementation, we used papers on new topics like Quantum Physics and some related to AI as external resources. One of the physics papers was largely altered to contain misinformation about the subject, and a paper related to AI was slightly modified to include misleading information.

In Table 5.1 and Table 5.2, we can find the results of these models alongside their fine-tuned versions. Table 5.1 shows the number of inaccurate responses generated by each model when using a benign vector database in the first phase of the experiments, compared to the number of incorrect responses when the models were exposed to a vector database containing misleading documents. All the questions used in these experiments were drawn from the papers in the external resource.

The second part of the results, shown in Table 5.2, corresponds to the generated responses by the models to the questions derived from the two specific papers that were modified. The second column, labeled "unaltered papers," represents the number of inaccurate responses when the models used documents without misinformation to answer the related questions. The third column shows the number of incorrect responses when the models answered questions related to those papers replaced by the ones that contained misinformation.

LLMs	Benign Database	Malicious Database
Llama-2-7B-chat-hf	5	9
Llama-2-7B-chat-hf-fine-tuned	1	8
Mistral-7B-Instruct-v0.2	11	9
Mistral-7B-Instruct-v0.2-fine-tuned	7	16
Llama-2-13B-chat-hf	6	11

Table 5.1: Inaccurate responses out of 100 generated by LLMs using benign and malicious vector databases. The first column shows the total number of incorrect responses when LLMs used the benign dataset to generate responses, while the third column represents the total number of incorrect responses produced by the models when the malicious information is injected into the vector database.

LLMs	Unaltered Papers	Altered Papers
Llama2-7B-chat-hf	1	9
Llama2-7B-chat-hf-fine-tuned	0	8
Mistral-7B-Instruct-v0.2	1	5
Mistral-7B-Instruct-v0.2-fine-tuned	1	8
Llama2-13B-chat-hf	1	8

Table 5.2: Inaccurate responses out of 20 from LLMs when using unaltered versus altered papers inside the vector database focused on the two specific papers. This table solely regards the incorrect responses to the 20 questions derived from the two modified papers.

In Table 5.3 and Table 5.4, like the results shown in Table 5.1 and Table 5.2 focus on only the two unaltered and altered papers but provide a clearer view of the models' performance by emphasizing on accuracy as the evaluation metric. These tables will be used to simplify the interpretation of the results. The accuracy measure makes it easier to understand how well the models performed when exposed to both benign and malicious data.

5.1.1 Accuracy Analysis

We start with the analysis of Table 5.3. We observe a decrease in accuracy in all models when comparing the results from the benign database to the malicious database. This shows that the presence of misleading documents in the vector database has affected the models' ability to generate accurate responses.

Notably, Mistral-7B-Instruct-v0.2 does not follow the same pattern as the other models. Its accuracy shows a slight increase to 91%, even when using the altered papers. On the other hand, its fine-tuned version experiences the most significant decrease among all the models, dropping to 84%. This suggests that fine-tuning may have had an adverse effect on this model's performance. However, we cannot conclusively link the drop in accuracy only to the presence of malicious information. To better understand these results, it is crucial to investigate the specific documents retrieved from the data store for each question or compare the inaccuracies in the responses generated for the questions derived from the malicious papers

Table 5.4 provides a clearer perspective on the impact of altered papers by focusing on responses to questions directly related to them. Unlike Table 5.3, we observe a significant drop in accuracy across all models, indicating that the misleading information in the altered papers substantially affects the models' performance in generating responses.

The Llama2-7B-chat-hf model, both in its baseline and fine-tuned versions, experiences the most significant decrease in accuracy, dropping by 40%, which demonstrates its vulnerability to the introduction of misinformation. Similarly, Llama-2-13B-chat-hf and Mistral-7B-Instruct-v0.2-fine-tuned exhibit substantial accuracy drops of 35%, indicating that fine-tuning alone does not significantly enhance the models' resilience to misleading information and may not be sufficient to maintain robustness against such vulnerabilities at this level of attack.

On the other hand, the Mistral-7B-Instruct-v0.2 baseline model presents the best performance with only a 20% drop in accuracy, suggesting a higher resilience to altered information compared to the other models. This could be due to the robustness of the model architecture or the way it processes and generates responses using the retrieved information from the data

LLMs	Unaltered papers	Altered papers
Llama-2-7B-chat-hf	95%	91%
Llama-2-7B-chat-hf-fine-tuned	99%	92%
Mistral-7B-Instruct-v0.2	89%	91%
Mistral-7B-Instruct-v0.2-fine-tuned	93%	84%
Llama-2-13B-chat-hf	94%	89%

Table 5.3: Accuracy of LLMs (out of 100 responses) when using unaltered versus altered papers across the entire data store.

LLMs	Unaltered papers	Altered papers
Llama-2-7B-chat-hf	95%	55%
Llama-2-7B-chat-hf-fine-tuned	100%	60%
Mistral-7B-Instruct-v0.2	95%	75%
Mistral-7B-Instruct-v0.2-fine-tuned	95%	60%
Llama-2-13B-chat-hf	95%	60%

Table 5.4: Accuracy of LLMs (out of 20 responses) when using unaltered versus altered papers focused on the two specific papers regarding the incorrect responses to questions derived from the two modified papers.

store.

5.1.2 Impact of Misleading Data on Model Performance

To better understand the impact of the misleading data introduced by the two modified papers, Table 5.5 represents the proportion of incorrect responses among the 20 responses to questions derived from the two altered papers when the models were exposed to the malicious database, compared to the total number of incorrect responses generated across all 100 responses when the models were exposed to the same malicious database. Notably, all the incorrect responses generated by Llama-2-7B-chat-hf and its fine-tuned version were related to questions derived from the papers that were later modified to include misinformation. The other models, however, showed an increase in the total number of incorrect responses across the entire set of questions. Interestingly, Mistral-7B-Instruct-v0.2-fine-tuned performed the worst, generating the highest number of inaccuracies in its responses across all questions.

LLMs	APIRs	TIRs
Llama-2-7B-chat-hf	9	9
Llama-2-7B-chat-hf-fine-tuned	8	8
Mistral-7B-Instruct-v0.2	5	9
Mistral-7B-Instruct-v0.2-fine-tuned	8	16
Llama-2-13B-chat-hf	8	11

Table 5.5: The second column, Altered Papers Incorrect Responses (APIRs), represents the total number of incorrect responses related to questions drawn from the altered papers. In contrast, the third column, Total Incorrect Responses (TIRs), shows the total number of incorrect responses generated by the models across all questions when exposed to the malicious database.

5.1.3 Interpretation of Results and Limitations

Overall, the results provide valuable insights. Although the fine-tuned versions of Llama-2-7B-chat-hf and Mistral-7B-Instruct-v0.2 performed better when exposed to the benign database compared to their baseline counterparts, the results suggest that fine-tuning alone, especially with datasets that are not perfectly curated to address the model's knowledge gaps, does not necessarily enhance resilience or robustness against misleading information. For example, while the fine-tuned Llama-2-7B-chat-hf demonstrated the best performance across all models in responding to the questions, Mistral-7B-Instruct-v0.2 showed the worst performance when exposed to misinformation. This indicates that even with fine-tuning, models may remain vulnerable to altered, misleading documents, and the benefits of fine-tuning are not uniformly substantial across different models.

We conducted these experiments on a small scale, and, likely, different fine-tuning configurations could potentially yield different results. Additionally, it is important to consider the tendency of LLMs to hallucinate and generate incorrect information, which can significantly impact the accuracy of responses. We have already discussed how hallucination can affect the generated outputs and change the behavior of the models. The accuracy of the results could also be improved through evaluation by human experts, offering better insights into the correctness of the responses.

This analysis indicates the need for further investigation into how these models retrieve and process information, as well as the development of more advanced strategies to detect and mitigate the impact of misinformation in RAG-based systems. This is crucial for improving the reliability and safety of AI applications in real-world scenarios.

6 Conclusion

6.1 Conclusion: Security Risks in RAG

In my thesis, we explored the security vulnerabilities of LLMs when integrated with RAG systems, focusing on the consequences of exposing vector databases in RAG-based systems to misleading data. By deliberately poisoning the vector database with misleading information, we aimed to examine the susceptibility of LLMs to malicious data and assess how these models respond when their knowledge base is compromised. We designed a comprehensive experiment to demonstrate these challenges of integrating LLMs with RAG systems. The results of this detailed experimental analysis contribute significantly to understanding the risks linked to implementing RAG systems in real-world applications, such as Question-Answering systems or chatbots.

The findings of my research revealed potential weaknesses in RAG-based systems and demonstrated that attackers could manipulate these systems by injecting false information into the vector databases, thus resulting in a decrease in the accuracy of LLMs. My research not only identified specific vulnerabilities in LLMs when faced with poisoned vector databases but also highlighted the broader implications for the security and reliability of AI systems that depend on external knowledge sources in RAG-based systems. These insights are important for the development of more secure and robust AI models, particularly as LLMs become increasingly integrated into various domains and into new approaches like RAG, which are used to mitigate

their existing limitations.

6.2 Future Works

While we explore the significant risks associated with integrating LLMs into RAG systems in this research, it also opens several opportunities for future studies. First, we need experiments with larger, more diverse, and customized datasets to assess their impact on fine-tuned models and their behavior when integrated with RAG systems. Although in this study we focused on small-scale models, it is crucial to test these findings on larger, and even closed-source models like OpenAI's state-of-the-art models, GPT-4o and GPT-4 Turbo, to evaluate their resilience and robustness against such attacks [120].

Another promising direction of research involves focusing on mitigation strategies for RAG systems. Zhang et al. introduced Retrieval Augmented Fine Tuning (RAFT) as a strategy to improve LLM performance in domain-specific tasks by training models to focus on relevant documents and ignore distractors [121]. While RAFT has shown notable success, outperforming traditional methods like SFT and RAG in specialized contexts, more advanced techniques are needed to detect and mitigate adversarial attacks, particularly those involving the introduction of poisoned data during the training process.

Another area we would like to explore is alternative architectures or hybrid models, such as GraphRAG, which combines structured graph data with traditional vector search methods to enhance retrieval accuracy [122]. Investigating how these models respond to misleading data could provide valuable insights into improving the robustness of LLMs within RAG systems.

Several other areas are worth exploring, such as Membership Inference Attacks (MIA), which predict whether a sample was part of a model's training set, exposing significant privacy vulnerabilities. Shokri et al. [123], provided a foundational analysis of these attacks across various machine-learning models. Recently, Duan et al. [124], extended MIA to LLMs, demonstrating that these attacks are feasible under certain conditions with more complex methods. Additionally, Mozaffari et al. [125], introduced the Semantic Membership Inference Attack (SMIA), which enhances MIA effectiveness by leveraging the semantic content of inputs and their perturbations. Future work could focus on the feasibility of MIA in RAG-based systems and explore strategies to mitigate these security vulnerabilities.

Finally, improving the interpretability and transparency of LLMs, alongside enhancing the robustness of approaches like RAG, could play an important role in identifying and addressing security vulnerabilities, ultimately contributing to the development of safer and more reliable

AI systems.

References

[1] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," 2023. [Online]. Available: https://arxiv.org/abs/2303.18223

[2] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Akhtar, N. Barnes, and A. Mian, "A comprehensive overview of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2307.06435

[3] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," 2023. [Online]. Available: https://arxiv.org/abs/2304.13712

[4] R. Graph, "The journey of large language models: Evolution, application, and limitations," 2024. [Online]. Available: https://medium.com/@researchgraph/the-journey-of-large-language-models-evolution-application-and-limitations-c72461bf3a6f

[5] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," 2019. [Online]. Available: https://arxiv.org/abs/1912.05911

[6] Wikipedia, "Long short-term memory," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Long_short-term_memory

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[8] Wikipedia, "Seq2seq," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Seq2seq

[9] E. L. Voita, "Sequence to sequence (seq2seq) and attention," 2023. [Online]. Available: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[11] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," 2023. [Online]. Available: https://arxiv.org/abs/2305.14314

[12] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2312.10997

[13] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mistral 7b," 2023. [Online]. Available: https://arxiv.org/abs/2310.06825

[14] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[15] J. E. Joseph, "Ferdinand de saussure." [Online]. Available: https://www.oxfordbibliographies.com/view/document/obo-9780199772810/obo-9780199772810-0003.xml

[16] A. M. TURING, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, 10 1950. [Online]. Available: https://doi.org/10.1093/mind/LIX.236.433

[17] J. Weizenbaum, "Eliza—a computer program for the study of natural language communication between man and machine," *Commun. ACM*, vol. 9, no. 1, p. 36–45, jan 1966. [Online]. Available: https://doi.org/10.1145/365153.365168

[18] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[19] OpenAI, "Hello gpt-4o," 2024. [Online]. Available: https://openai.com/index/hello-gpt-4o/

[20] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[22] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[23] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[24] Meta, "Large language model, introducing meta llama 3: The most capable openly available llm to date," 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3/

[25] M. A. Arefeen, B. Debnath, and S. Chakradhar, "Leancontext: Cost-efficient domain-specific question answering using llms," 2023. [Online]. Available: https://arxiv.org/abs/2309.00841

[26] Amazon AWS. [Online]. Available: https://aws.amazon.com/?nc2=h_lg

[27] Microsoft Azure. [Online]. Available: https://ai.azure.com/

[28] ScribbleData, "Fine-tuning large language models: Complete optimization guide," 2023. [Online]. Available: https://www.scribbledata.io/blog/fine-tuning-large-language-models/

[29] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021. [Online]. Available: https://arxiv.org/abs/2005.11401

[30] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," 2023. [Online]. Available: https://arxiv.org/abs/2311.05232

[31] P. Zhao, H. Zhang, Q. Yu, Z. Wang, Y. Geng, F. Fu, L. Yang, W. Zhang, J. Jiang, and B. Cui, "Retrieval-augmented generation for ai-generated content: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.19473

[32] P. Finardi, L. Avila, R. Castaldoni, P. Gengo, C. Larcher, M. Piau, P. Costa, and V. Caridá, "The chronicles of rag: The retriever, the chunk and the generator," 2024. [Online]. Available: https://arxiv.org/abs/2401.07883

[33] J. Hsia, A. Shaikh, Z. Wang, and G. Neubig, "Ragged: Towards informed design of retrieval augmented generation systems," 2024. [Online]. Available: https://arxiv.org/abs/2403.09040

[34] K. Sawarkar, A. Mangal, and S. R. Solanki, "Blended rag: Improving rag (retriever-augmented generation) accuracy with semantic search and hybrid query-based retrievers," 2024. [Online]. Available: https://arxiv.org/abs/2404.07220

[35] C. Chen and K. Shu, "Can llm-generated misinformation be detected?" 2024. [Online]. Available: https://arxiv.org/abs/2309.13788

[36] Z. Xu, Y. Liu, G. Deng, Y. Li, and S. Picek, "A comprehensive study of jailbreak attack versus defense for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2402.13457

[37] L. Lin, L. Wang, J. Guo, and K.-F. Wong, "Investigating bias in llm-based bias detection: Disparities between llms and human perception," 2024. [Online]. Available: https://arxiv.org/abs/2403.14896

[38] Y. Wang, T. Sun, S. Li, X. Yuan, W. Ni, E. Hossain, and H. V. Poor, "Adversarial attacks and defenses in machine learning-powered networks: A contemporary survey," 2023. [Online]. Available: https://arxiv.org/abs/2303.06302

[39] Y. Han, C. Liu, and P. Wang, "A comprehensive survey on vector database: Storage and retrieval technique, challenge," 2023. [Online]. Available: https://arxiv.org/abs/2310.11703

[40] Z. Xu, M. J. Cruz, M. Guevara, T. Wang, M. Deshpande, X. Wang, and Z. Li, "Retrieval-augmented generation with knowledge graphs for customer service question answering," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR 2024, vol. 33. ACM, Jul. 2024, p. 2905–2909. [Online]. Available: http://dx.doi.org/10.1145/3626772.3661370

[41] S. S. Manathunga and Y. A. Illangasekara, "Retrieval augmented generation and representative vector summarization for large unstructured textual data in medical education," 2023. [Online]. Available: https://arxiv.org/abs/2308.00479

[42] HuggingFace. [Online]. Available: https://huggingface.co/

[43] Meta-llama/Meta-Llama-3-8B-Instruct. [Online]. Available: https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct

[44] Meta-llama/Meta-Llama-3-70B-Instruct. [Online]. Available: https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct

[45] N. Daryabar, the project repository on GitHub. [Online]. Available: https://github.com/nimad70/VulRAG

[46] Meta, "Introducing llama 3.1: Our most capable models to date," 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3-1/

[47] PTI, "Meta just launched the largest 'open' ai model in history. here's why it matters," 2024. [Online]. Available: https://tech.hindustantimes.com/tech/news/meta-just-launched-the-largest-open-ai-model-in-history-here-s-why-it-matters-71722695907515.html

[48] Wikipedia, "Language model," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Language_model

[49] Wikipedia., "Large language model," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Large_language_model#cite_note-:7-1

[50] Amazon, "What are large language models (llm)?" 2024. [Online]. Available: https://aws.amazon.com/what-is/large-language-model/

[51] IBM, "What are large language models (llms)?" 2024. [Online]. Available: https://www.ibm.com/topics/large-language-models

[52] Google, "Introduction to large language models," 2024. [Online]. Available: https://developers.google.com/machine-learning/resources/intro-llms

[53] J. Goodman, "A bit of progress in language modeling," 2001. [Online]. Available: https://arxiv.org/abs/cs/0108005

[54] I. Solaiman, M. Brundage, J. Clark, A. Askell, A. Herbert-Voss, J. Wu, A. Radford, G. Krueger, J. W. Kim, S. Kreps, M. McCain, A. Newhouse, J. Blazakis, K. McGuffie, and J. Wang, "Release strategies and the social impacts of language models," 2019. [Online]. Available: https://arxiv.org/abs/1908.09203

[55] A. Hern, "New ai fake text generator may be too dangerous to release, say creators," 2019. [Online]. Available: https://www.theguardian.com/technology/2019/feb/14/elon-musk-backed-ai-writes-convincing-news-fiction?CMP=share_btn_url

[56] M. Schreiner, "Gpt-4 architecture, datasets, costs and more leaked," 2023. [Online]. Available: https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/

[57] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman, T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, Łukasz Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, Łukasz Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O'Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. de Avila Belbute Peres, M. Petrov, H. P. de Oliveira Pinto, Michael, Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss,

C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[58] Wikipedia, "Transformer (deep learning architecture)," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)

[59] Wikipedia., "Recurrent neural network," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network

[60] J. Zhang, T. He, S. Sra, and A. Jadbabaie, "Why gradient clipping accelerates training: A theoretical justification for adaptivity," 2020. [Online]. Available: https://arxiv.org/abs/1905.11881

[61] Wikipedia, "Word embedding," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Word_embedding

[62] Wikipedia., "Word2vec," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Word2vec

[63] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://aclanthology.org/D14-1162

[64] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1409.3215

[65] Wikipedia, "Attention (machine learning)," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Attention_(machine_learning)

[66] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016. [Online]. Available: https://arxiv.org/abs/1409.0473

[67] A. Abaskohi, "Navigating transformers: A comprehensive exploration of encoder-only and decoder-only models, right shift, and beyond," 2023. [Online]. Available: https://medium.com/@amirhossein.abaskohi/a0b46bdf6abe

[68] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," 2024. [Online]. Available: https://arxiv.org/abs/2312.00752

[69] Y. Li, S. Bubeck, R. Eldan, A. D. Giorno, S. Gunasekar, and Y. T. Lee, "Textbooks are all you need ii: phi-1.5 technical report," 2023. [Online]. Available: https://arxiv.org/abs/2309.05463

[70] L. Leffer, "When it comes to ai models, bigger isn't always better," 2023. [Online]. Available: https://www.scientificamerican.com/article/when-it-comes-to-ai-models-bigger-isnt-always-better/

[71] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre, "Training compute-optimal large language models," 2022. [Online]. Available: https://arxiv.org/abs/2203.15556

[72] Wikipedia, "Fine-tuning (deep learning)," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Fine-tuning_(deep_learning)

[73] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, "Parameter-efficient fine-tuning for large models: A comprehensive survey," 2024. [Online]. Available: https://arxiv.org/abs/2403.14608

[74] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[75] X. Qi, Y. Zeng, T. Xie, P.-Y. Chen, R. Jia, P. Mittal, and P. Henderson, "Fine-tuning aligned language models compromises safety, even when users do not intend to!" 2023. [Online]. Available: https://arxiv.org/abs/2310.03693

[76] Toloka, "The history, timeline, and future of llms," 2023. [Online]. Available: https://toloka.ai/blog/history-of-llms/

[77] R. Khawaja, "Llm use-cases: Top 10 industries that can benefit from using large language models," 2023. [Online]. Available: https://datasciencedojo.com/blog/llm-use-cases-top-10/

[78] Amazon, "What is rag (retrieval-augmented generation)?" 2024. [Online]. Available: https://aws.amazon.com/what-is/retrieval-augmented-generation/

[79] Wikipedia, "Vector database," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Vector_database

[80] Vectorizeio, "Which is the best vector database for rag applications?" 2024. [Online]. Available: https://medium.com/@vectorizeio/which-is-the-best-vector-database-for-rag-applications-e559822aeccd

[81] Z. Jing, Y. Su, Y. Han, B. Yuan, H. Xu, C. Liu, K. Chen, and M. Zhang, "When large language models meet vector databases: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.01763

[82] A. Kondamani, "Rag poisoning: An emerging threat in ai systems," 2024. [Online]. Available: https://medium.com/nfactor-technologies/rag-poisoning-an-emerging-threat-in-ai-systems-660f9ff279f9

[83] W. Zou, R. Geng, B. Wang, and J. Jia, "Poisonedrag: Knowledge corruption attacks to retrieval-augmented generation of large language models," 2024. [Online]. Available: https://arxiv.org/abs/2402.07867

[84] G. Deng, Y. Liu, K. Wang, Y. Li, T. Zhang, and Y. Liu, "Pandora: Jailbreak gpts by retrieval augmented generation poisoning," 2024. [Online]. Available: https://arxiv.org/abs/2402.08416

[85] IronCoreLabs, "Security risks with rag architectures," 2024. [Online]. Available: https://ironcorelabs.com/security-risks-rag/

[86] I. O. Gallegos, R. A. Rossi, J. Barrow, M. M. Tanjim, S. Kim, F. Dernoncourt, T. Yu, R. Zhang, and N. K. Ahmed, "Bias and fairness in large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2309.00770

[87] S. Samsi, D. Zhao, J. McDonald, B. Li, A. Michaleas, M. Jones, W. Bergeron, J. Kepner, D. Tiwari, and V. Gadepally, "From words to watts: Benchmarking the energy costs of large language model inference," 2023. [Online]. Available: https://arxiv.org/abs/2310.03003

[88] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, and X. Cheng, "On protecting the data privacy of large language models (llms): A survey," 2024. [Online]. Available: https://arxiv.org/abs/2403.05156

[89] E. Shayegani, M. A. A. Mamun, Y. Fu, P. Zaree, Y. Dong, and N. Abu-Ghazaleh, "Survey of vulnerabilities in large language models revealed by adversarial attacks," 2023. [Online]. Available: https://arxiv.org/abs/2310.10844

[90] Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, K. Wang, and Y. Liu, "Jailbreaking chatgpt via prompt engineering: An empirical study," 2024. [Online]. Available: https://arxiv.org/abs/2305.13860

[91] Y. Liu, Y. Jia, R. Geng, J. Jia, and N. Z. Gong, "Formalizing and benchmarking prompt injection attacks and defenses," 2024. [Online]. Available: https://arxiv.org/abs/2310.12815

[92] H. Li, D. Guo, W. Fan, M. Xu, J. Huang, F. Meng, and Y. Song, "Multi-step jailbreaking privacy attacks on chatgpt," 2023. [Online]. Available: https://arxiv.org/abs/2304.05197

[93] A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein, "Poison frogs! targeted clean-label poisoning attacks on neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1804.00792

[94] Y. Qiang, X. Zhou, S. Z. Zade, M. A. Roshani, D. Zytko, and D. Zhu, "Learning to poison large language models during instruction tuning," 2024. [Online]. Available: https://arxiv.org/abs/2402.13459

[95] Y. Luo, Z. Yang, F. Meng, Y. Li, J. Zhou, and Y. Zhang, "An empirical study of catastrophic forgetting in large language models during continual fine-tuning," 2024. [Online]. Available: https://arxiv.org/abs/2308.08747

[96] AlignmentLabAI/OpenOrca. [Online]. Available: https://huggingface.co/datasets/Open-Orca/OpenOrca

[97] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah, "Orca: Progressive learning from complex explanation traces of gpt-4," 2023. [Online]. Available: https://arxiv.org/abs/2306.02707

[98] E. Hartford, 2023, cognitivecomputations/dolphin. [Online]. Available: https://huggingface.co/datasets/cognitivecomputations/dolphin

[99] Garage-bAInd/Open-Platypus. [Online]. Available: https://huggingface.co/datasets/garage-bAInd/Open-Platypus

[100] A. N. Lee, C. J. Hunter, and N. Ruiz, "Platypus: Quick, cheap, and powerful refinement of llms," 2024. [Online]. Available: https://arxiv.org/abs/2308.07317

[101] M. D. Hervé Jegou and J. Johnson, "Faiss: A library for efficient similarity search," 2017. [Online]. Available: https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

[102] HuggingFace, mteb/leaderboard. [Online]. Available: https://huggingface.co/spaces/mteb/leaderboard

[103] ArtifactAI, artifactAI/arxiv-physics-instruct-tune-30k. [Online]. Available: https://huggingface.co/datasets/ArtifactAI/arxiv-physics-instruct-tune-30k

[104] M. G. Collu, T. Janssen-Groesbeek, S. Koffas, M. Conti, and S. Picek, "Dr. jekyll and mr. hyde: Two faces of llms," 2024. [Online]. Available: https://arxiv.org/abs/2312.03853

[105] HuggingFace, open-llm-leaderboard. [Online]. Available: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

[106] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging llm-as-a-judge with mt-bench and chatbot arena," 2023. [Online]. Available: https://arxiv.org/abs/2306.05685

[107] MistralAI, "Mistral 7b," 2023. [Online]. Available: https://mistral.ai/news/announcing-mistral-7b/

[108] Mistralai/Mistral-7B-Instruct-v0.2. [Online]. Available: https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2

[109] Meta, "Llama 2: open source, free for research and commercial use," 2024. [Online]. Available: https://llama.meta.com/llama2/

[110] Meta-llama/Llama-2-7b-chat-hf. [Online]. Available: https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

[111] Meta-llama/Llama-2-13b-chat-hf. [Online]. Available: https://huggingface.co/meta-llama/Llama-2-13b-chat-hf

[112] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu, S. Zhang, G. Ghosh, M. Lewis, L. Zettlemoyer, and O. Levy, "Lima: Less is more for alignment," 2023. [Online]. Available: https://arxiv.org/abs/2305.11206

[113] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[114] L. team, "What is langchain?" 2024. [Online]. Available: https://github.com/langchain-ai/langchain

[115] F. B. Shaikh and S. Haider, "Security threats in cloud computing," in *2011 International Conference for Internet Technology and Secured Transactions*, 2011, pp. 214–219.

[116] Google, "Classification: Accuracy, recall, precision, and related metrics," 2024. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall

[117] scikit-learn team, "f1 score," 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

[118] B. A. of Artificial Intelligence, bAAI/bge-base-en-v1.5. [Online]. Available: https://huggingface.co/BAAI/bge-base-en-v1.5

[119] J. Huber and A. Troynikov, "Chroma," 2024. [Online]. Available: https://docs.trychroma.com/

[120] OpenAi, "Retrieval augmented generation (rag) and semantic search for gpts," 2024. [Online]. Available: https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts

[121] T. Zhang, S. G. Patil, N. Jain, S. Shen, M. Zaharia, I. Stoica, and J. E. Gonzalez, "Raft: Adapting language model to domain specific rag," 2024. [Online]. Available: https://arxiv.org/abs/2403.10131

[122] T. Bratanic, "Enhancing the accuracy of rag applications with knowledge graphs," 2024. [Online]. Available: https://medium.com/neo4j/enhancing-the-accuracy-of-rag-applications-with-knowledge-graphs-ad5e2ffab663

[123] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," 2017. [Online]. Available: https://arxiv.org/abs/1610.05820

[124] M. Duan, A. Suri, N. Mireshghallah, S. Min, W. Shi, L. Zettlemoyer, Y. Tsvetkov, Y. Choi, D. Evans, and H. Hajishirzi, "Do membership inference attacks work on large language models?" 2024. [Online]. Available: https://arxiv.org/abs/2402.07841

[125] H. Mozaffari and V. J. Marathe, "Semantic membership inference attack against large language models," 2024. [Online]. Available: https://arxiv.org/abs/2406.10218

Acknowledgments

This work is dedicated to Prof. Mauro Conti and Dr. Stjepan Picek, along with his research group and the Digital Security Department. Your unwavering guidance, support, and expertise have been the cornerstone of this research. The insights and opportunities you provided were instrumental in shaping this work, and it would not have been possible without your help. I am profoundly grateful for your mentorship and the invaluable contributions you have made to my academic journey.