Leveraging LLMs with RAG to recommend Points of Interest to tourists

Supervisor
Prof. Niccolò MARASTONI

Graduate Student
Patrick HAMZAJ
VR474246

Dedica

Acknowledgments

Abstract

Qui va l'abstract

Contents

Introduction

Over the last decade, the field of artificial intelligence (AI from now on) has experienced significant growth and rapid advancements. Among the different branches of AI, *deep learning* has stood out for its ability to tackle complex tasks that were considered infeasible or extremely challenging. In particular, *Natural Language Processing* (NLP) has gained prominence due to the increasing volume of unstructured and textual data generated and subsequently gathered online (e.g., social media, blogs, scientific articles...); it is estimated that 328.77 million terabytes of data are created daily, 80% of which are unstructured data. [1] This phenomenon and the rising demand for language-driven applications, such as virtual assistants, sentiment analysis, and automated content generation, has prepared the terrain for the advancement of languange processors to flourish.

Recently, the emergence of *Large Language Models* (LLMs), led by the release of GPT-3 in 2020 [2], has taken NLP to a new level. By employing architectures with billions of parameters, these models are capable of producing remarkably fluent, context-aware and human-like text. Despite this progress, several open challenges remain: questions regarding how best to *fine-tune* LLMs, how to incorporate domain-specific knowledge, and how to design effective prompting mechanisms are active areas of research. Additionally, issues related to computational resources, scalability, ethical implications, and potential biases call for continuous investigation.

Motivations. The primary motivation behind this thesis is to explore the practical methods and architectural choices to enable the construction of an LLM-powered application to be effectively adapted for the domain of tourism in Verona, Italy. In particular, the aim is to utilize and demonstrate how techniques such as *prompt engineering*, and *retrieval-augmented generation* (RAG) can facilitate the deployment of state-of-the-art LLMs in real-world applications, including scenarios where computational resources may be limited or costly.

As such, the **objectives** are:

- **Analyze** the foundational principles and evolution of modern neural network architectures, particularly focusing on the *Transformer* model and its role in Large Language Models.

- **Investigate** the current state of the art for LLMs, highlighting successful applications and the most common methodologies (fine-tuning, prompt engineering, RAG, etc.).

- **Design and implement** a tourism domain application system leveraging a specific open-source LLM (LLaMA 3.1 8B Instruct), showcasing relevant techniques (8-bit quantization, agent-like conversational behavior) for an implicit recommender system for tourists.

- **Evaluate** the system with respect to performance and quality metrics, as well as potential limitations.

From a high-level perspective, Large Language Models represent a class of deep neural netowrks trained on extensive corpora, often comprising billions of parameters. A key turning point in their development was the introduction of the *Transformer* architecture, which provides a self-attention mechanism that allows to improve text tokens processing in a parallel manner during training.

Examples of prominent LLMs include GPT-based models (GPT-3, GPT-3.5, GPT-4), [2] Google's PaLM [3], Meta's LLaMA, [4] and various open-source initiatives such as Bloom. [5] These models have demonstrated remarkable capabilities in language understanding and generation, enabling:

- **Text completion** with human-like fluency.

- **Zero-shot or few-shot generalization** to new tasks with minimal prompt examples.

- **Conversational AI**, powering advanced chatbots and virtual assistants.

This thesis will leverage the open-source LLaMA model family as a foundation for an LLM-based chatbot to suggest points of interest to touristi visiting the city of Verona, Italy. By focusing on a quantized 8-bit instruct variant (LLaMA 3.1 8B Instruct), the aim is to highlight practical techniques (RAG, agent AI) that maintain acceptable performance implementing innovative strategies that harness context-awareness and text generation capabilities/question answering of modern LLM models.

The remainder of this document is organized as follows: Chapter 1 presents a deeper dive into the theoretical underpinnings of modern NLP, covering the history of neural networks and the transformation from classic word embeddings to large-scale language models, also detailing the breakthrough Transformer architecture. Chapter 2 reviews the current landscape of Large Language Models and NLP solutions, exploring established techniques (fine-tuning, prompt engineering, RAG) and highlighting real-world use cases, from chatbots to advanced content generation. Chapter 3 describes the design and implementation of an application based on LLaMA 3.1 8B Instruct for tourism purposes in the city of Verona, Italy. It covers the decision-making process behind the model choice, 8-bit quantization strategy, prompt engineering, retrieval-augmented generation, and the agent-like conversational framework. Chapter 4 discusses the evaluation with illustrative examples of the model's outputs and conversations, along with limitations and areas for improvement. Finally, Chapter 5 concludes the study with a summary of key findings, limitations, and potential future directions.

Chapter 1

Theoretical Foundations and Background

1.1 A Brief History of Neural Networks

The term *Neural Network* refers to the attempt of defining a mathematical view of the structure of the human brain, aiming at emulating it and transposing the logical functioning and the learning capabilities into computational models.

The very first attempt of studying the biological brain and its neural activity in terms of formal logic is attributed to McCulloch and Pitts,[6] who proposed that neurons could be represented as simple binary devices, whose key aspects are:

- **Logical Units:** neurons are basic units with an on/off switch, allowing them to describe neural activi'ty using the language of logic (i.e. logical propositions like *AND*, *OR...*).

- **Threshold:** neurons "fire" only when a certain threshold of input is met.

- **Links:** like synapses, links connect the logical units from the input to the output.

This basic understanding of neural processes influeced later developmentes in neuroscience and artificial intelligence, bringing Rosenblatt to produce the *Perceptron*: a single layer neural network that, upon previous work from McCulloch and Pitts, introduced:

- **Weights:** every connection has a weight, that is added to the input received.

- **Learning mechanism:** introduces a learning algorithm where the weights of the connections are adjusted based on errors in the output.

This linear architecture allows to generalize the classification tasks, using probabilistic rules that could perform nontrivial tasks like pattern recognition and information organization.

Neural networks have evolved dramatically over the decades

The enthusiasm was enormous and the field of cybernetics was born: however, it didn't take long for researchers to uncover the limitations of single-layer networks. Minsky's and Papert's demonstrated the limitations of the perceptron—that is, that certain classes of functions were simply out of reach for these early models (for example, the logical XOR
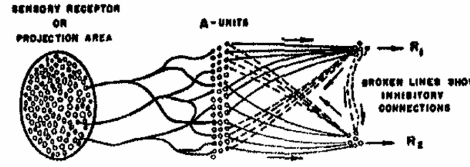
FIG. 2A. Schematic representation of connections in a simple perceptron.

Figure 1.1: The perceptron architecture.

function)—interest quickly waned. [7] This realization contributed to a period of reduced enthusiasm for neural network research, often referred to as the "AI Winter."

Interest was revived in the 1980s with the introduction of the backpropagation algorithm [8], which made it possible for deeper architectures to learn more complex functions. The core idea was to train neural networks by propagating the error from the output layer backward through the network layers; this approach allows the network to adjust its internal weights based on the error itself, so that it could "learn" using gradient descent to minimize the error function, changing the weights individuating the contribution of each neuron in constructing the output.

Although early progress was hampered by hardware constraints, continuous incremental improvements over the following decades, combined with advances in parallel computing, eventually paved the way for the deep learning revolution we see today [9].

By the early 2010s, the impact of Convolutional Neural Networks (CNNs) on computer vision tasks [10] highlighted the benefits of large datasets, GPU-based parallel training, and increasingly sophisticated network designs. At the same time, improvements in Recurrent Neural Networks (RNNs)—especially with LSTM [11]—opened up new possibilities in sequence modeling, including areas like language translation and speech recognition. These advancements ultimately set the stage for the development of Large Language Models (LLMs), particularly after the introduction of the Transformer architecture [12].

1.2 The Transformer Architecture

The introduction of the Transformer by Vaswani et al. [12] marked a clear departure from traditional recurrent and convolutional models. The focus of the original research was on translation tasks, by leaning heavily on a **self-attention** mechanisms which allows the model to weigh the importance of different words in a seuqence of words, taking into account the relationship to eacht ore.

The Transformer architecture consists of two main parts: an **encoder** and a **decoder**, each built from multiple layers that implement multi-head self-attention alongside feed-forward networks, which can be visualized in **??**:

1. **Encoder:** it converts an input sequence (e.g., a sentence) into a series of representations that capture the contextual meaning of the input. Within each encode layer, a sub-layer allows every token in the input to consider the influence of every other token.

2. **Decoder:** it uses the encoder's representation along with other inputs to generate a target sequence using a masking mechanism (i.e. predicting the next token is only decided to previous tokens, and not future ones).
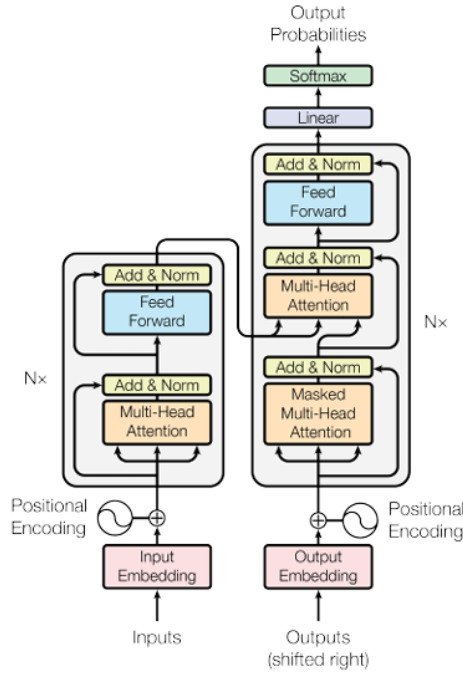
Figure 1.2: The Transformer architecture.

Positional encoding is another breakthrough of this paper, integrated in both parts, which compensates for the lack of sequence awareness. It incorporates information about the order of tokens in a sequence. Since these models process tokens in parallel rather than sequentially, positional embeddings are essential for conveying information about the order of tokens.

Each of these parts can be used independently, depending on the tasks: encoder-only models are good for tasks that require understanding of the input, such as sentence classification; decoder-only models are good for generative tasks such as text generation. Encoder-decoder models (called sequence-to-sequence models) are used for generative tasks that require an input, such as translation or summarization.

1.3 Transformers as Language Models

In recent applications, the Transformer model have been trained as *language models* meaning they have been trained on large amounts of raw text in a self-supervised fashion, which is a type of training in which the objective is automatically computed from the inputs of the model.

This type of models develop a statistical understanding of the language it has been trained on, but it's not very useful for specific practical tasks. Because of this, the general pretrained model then goes through a process called *transfer learning*. During this process, the model is fine-tuned in a supervised way — that is, using human-annotated labels — on a given task.

An example of a task is predicting the next word in a sentence having read the previous

words. This is called causal language modeling because the output depends on the past and present inputs, but not the future ones.

Training is a fundamental step in the adoption and implementation of a language model. The learning capacity of LLMs are generally dived into:[13]

1. **Pre-training:** it is the first stage in training an LLM, where the model learns general linguistic patterns, facts and knowledge from a vast corpus of text. It is the act of training a model from scratch: the weights are randomly initialized, and the training starts without any prior knowledge. Tecniques of pre-training phase are:

   - *Masked language modeling*, used in decoder-models, where certain words are masked and the model learns to predict them.[14]

   - *Causal language modeling*, used in encoder-only models like GPT,[2] where the model predicts the next word in a sequence.

2. **Fine-tuning:** after pre-training, the model undergoes a further training on a smaller, task-specific dataset to improve performance for particular and domain-specific applications. Types of fine-tuning are:

   - *Supervised fine-tuning* is used to train models on labeled data, such as question-answering datasets.

   - *Instruction tuning* involves training the model on a dataset of input-output pairs, where each input is phrased as an instruction and the output is the desired response. Most ready-to-use models are instruct-tuned, as they have improved generalization and natural responses. An example can be seen in Table **??**.

   - *Parameter-Efficient fine-tuning (PEFT)*, methods like *Low-Rank Adaption (LORA)*[15] are innovative techniques that reduce the number of parameters to train, thus reducing computaional costs.

Generally, the strategy to achieve better performance is by increasing the models' sizes as well as the amount of data they are pretrained on, but higher performances lead to higher resources intensive trainings. This is why different strategies have been developed to achieve good performances without the need of training models.

Instruction	Input (optional)	Expected Output
Translate this sentence into Spanish	"Hello, how are you?"	"Hola, ¿cómo estás?"
Summarize the text in one sentence	"The global economy is facing uncertainty due to inflation and geopolitical issues."	"The global economy is unstable due to inflation and geopolitics."
Explain how photosynthesis works to a 5-year-old	*No input*	"Plants use sunlight to make food, like how we eat to get energy!"

Table 1.1: Examples of Instruction Tuning

1.4 Challenges and Limitations of LLMs

1.5 Summary

Chapter 2

Recent Advances in Large Language Models

Large Language Models (LLMs) have become a cornerstone of modern Natural Language Processing (NLP), enabling a wide range of applications such as chatbot systems, code generation, and content creation: it is estimated that by the end of 2025 there will be 750 million apps using LLMs as a source of interaction.[16] This chapter provides an overview of current leading LLM families and their innovations, explores key techniques for their customization, introduces the concept of Retrieval-Augmented Generation (RAG), Tool Utilization and Prompt Engineering.

2.1   Major Model Families and Their Characteristics

Most LLMs nowadays are built on the Transformer architecture with hundreds of millions to hundreds of billions parameters (and, in recent works, over a trillion), which take advantage of the availabilty of large-scale text corpora on the internet, along with improvements in scalable hardware (GPUs, TPUs) and advances in Transformer architecture, that adhere to the *scaling law*[17] for which systematicallt scaling model size, data size, and compute lead to improved performance in LLM models.

Overall, LLMs nowadays can be divided into three main families.[18]

2.1.1   GPT Family (Generative Pre-Trained Transformers)

The GPT family consists of decoder-only models developed by OpenAI, introducing GPT in 2018, which is defined as an undirectinal auto-regressive model. The key innovation here is that it showed that large-scale, unsupervised pre-training followed by a punctual fine-tuning on specific tasks that the model is supposed to achieve, could outperform traditional architectures of many NLP benchmarks.

A year later, the company introduced GPT-2, scaling up the parameter count up to 1.5 billion parameters: it demonstraded zero-shot capabilities on tasks like summarization or translation, only using prompting for interacting with the model.

The term *zero-shot capability* refers to the ability for a model to prompting an LLM without any examples, attempting to take advantage of the generalization it has gained through training, using reasoning patterns it has acquired.

GPT-3 saw another scale up in parameters count, achieving a surprising 175 billion count: by giving few-shot examples at inference time, it demonstrated an *in-context learning*, for which no additional fine-tuning was required.

Finally, GPT-3.5 introduced the rinomate *ChatGPT*, which introduced LLMs to the general public. It incorporated instruction-following behavior and Reinforcement Learning from Human Feedbak (RLHS) to reduce harmful content; later, with GPT-4, it introduced multi-modal capabilities interacting with and generating non-textual data.

Overall, the GPT Family has been acclaimed for its investments in scaling models with higher paramateres and text corpora, achieving few-shot capabilities without the need to further training models.

2.1.2 BERT Family (Bidirectional Encoder Representations)

The BERT-family was introduced by Google in 2018, and it focuses on tasks that require understanding of the input, such as sentence classification and named entity recognition. In this context they utilize and encoder-only architecture, performing a masked contextual understanding, meaning that their training in based mainly on masking tokens at each iteration and gaining a probabilistic choice of the next token prediction.

This kind of models are strong at understaing the semantics of text from a bidirectional perspective: through Masked Language Modeling (MLM), the training consists of randomly masking 15% of the tokens in the input and ask the model to predict the original vocabulary tokens, encouraging to learn from the left and from the right sides of the masked token (that is, bidirectional).

BERT is by design a 110 million parameters in its base version, whereas it has 340 million parameters in its largest one, which has seen several variants over the years, like RoBERTA which increased training data and steps involved in the training phase, or ALBERT, with a paremter-reduction technique to handle scaling more efficiently, and finally DistilBERT, that implemented *knowledge distillation*: a novel technique in making LLMs more practical and efficient, by transferring essential knowledge from a complex teacher model to a smaller student model, preserving performance while reducing size and computational demands.

2.1.3 T5 Family (Text-to-Text Transfer Transformer)

Introduced as well by Google (2019), the T5-family model uses a full sequence-to-sequente (i.e., encoder-decoder) Transformer architecture. The core idea is to formulate every NLP task as a "text-to-text" problem-inputs and outputs are always text strings.

Prior to T5, large-scale models (e.g., GPT, BERT) had shown the power of pre-training on unlabaled data followed by task-specifi fine-tuning; however, these approaches often frame tasks differently (classification vs prediction vs language modeling). This divergence complicates the use of a single frameworks for multiple tasks: so, the goal was to create a unified approach to NLP tasks by casting all problems into a text-to-text format.

The authors released multiple variants, with the largest size counting up to 11 billion parameters, and their approach in training was a modifief version of MLM, which is Span Corruption: randomly masking not individual tokens, by entire span of texts. This demonstraded to foster a more coherent learning of consecutive tokens and encouraged the model to handle variable-length contexts.

The T5 has been superseded later on by mT5 and T5-XXL, with larger parameter counts and multilingual corpora for cross-lingual transfer, which yields strong results across different tasks, highlighting flexibility for multitask settings.

These families do not comprise the several hundreds of foundation models released in recent years, but it nonetheless give a high perspective of the model architectures, their characteristics and key innovations. A common strategy has been underlined across the realm of these families: the bigger the size, the better. That is why training has always been a prerogative of the few, large companies or research centers that can handle higher computational resources and costs.

2.2 Beyond Fine-Tuning: Alternative Strategies for Optimizing LLMs

Given the computational challenges of fine-tuning, alternative approaches such as Retrieval-Augmented Generation (RAG), Prompt Engineering, and Agent AI have emerged as effective strategies for achieving task-specific accuracy without the need for extensive model retraining.

As LLMs become increasingly prevalent, practitioners have devised a range of methods to tailor their behavior for specific use cases and to deploy them efficiently. Fine-tuning remains a common approach, allowing developers to update the model weights on domain-specific datasets. This can be performed as a full-scale process, adjusting all parameters, or via more parameter-efficient methods such as LoRA [15], which inserts low-rank updates into the model's layers and thus reduces memory requirements; this approach still produces prohibitive settings to develop LLM specific implementations, that is why innovative techniques have emerged, gathering fine-tuned models that possess general knowledge directing its focus on specific tasks and use cases.

Prompt engineering, takes advantage of the model's pre-trained knowledge by carefully crafting textual instructions that guide it toward the desired output [19]. Rather than modifying the model weights, prompt engineering modifies the input context to clarify the task objective or to showcase example queries and answers. This practice has grown in importance with the rise of instruction-tuned models, which are trained to follow natural language instructions rather than purely statistical patterns.

Another pivotal approach involves integrating an external knowledge base into the generation process. Retrieval-Augmented Generation (RAG)[20] leverages a retrieval module, typically built on vector search engines, to fetch relevant documents or snippets from a corpus. The LLM then conditions its output on these retrieved texts, thereby grounding its responses in verifiable sources and reducing hallucination or factually incorrect statements[21].

Lastly, the concept of Tools have introduced the term "AI Agents". Tools are external modules or functions that the model can decide to invoke, to perform tasks or gather knowledge which was not comprised in its training and go beyond text generation. It leverages the model's internal decision mechanism that will determine if an action must be taken based on users' input. For example, asking what the weather is like right now is excluded by the factual limited knowledge the model possesses; it can be thereby create a tool that calls an API with weather data, and when the trigger has been captured by the model it can decide to call this tool to retrieve the updated data.

In the next sections we will dive into these innovative strategies to benefit from pre-trained models with powerful text-generation abilities of LLMs without incurring into expensive setups. Nevertheless, it must be first introduced the concept of prompting and the interaction with a language model.

2.2.1 From Words to Numbers and Back to Words: Tokenization

Tokenizers are one of the core components of the NLP pipeline, which hey serve one purpose: to translate text into data that can be processed by the model. As in language models the data that is generally produced is raw text, and knowing that Transfomers can only process numbers, tokenizers need to convert text inputs to numerical data. So, the goal of tokenizers is to find the most meaningful representation — that is, the one that makes the most sense to the model — and if possible the smallest representation.

Tokenization splits text into mangeable units (that is, *tokens*) to enable the model to understand and process language, which are then converted into numerical representations that capture their semantics. It can be as simple as a word-based tokenization (i.e., each word is a token), to more complex techniques like *WordPiece* used in BERT models.

Each token is defined by the method used, for which the first that comes to mind would be a word-based tokenization. As an example, having the phrase "John was a puppeteer" could be divided into

```
Jim | Henson | was | a | puppeteer | .
```

And the goal would be to assign a numerical representation of each token

```
Jim | Henson | was | a | puppeteer | .
545 | 4668   | 109 | 9 | 10988     | 721
```

Each word gets assigned an ID, starting from 0 and going up to the size of the vocabulary: then the model uses these IDs to identify each word. But if we want to completely cover a language with a word-based tokenizer, we'll need to have an identifier for each word in the language, which will generate a huge amount of tokens. For example, there are over 500,000 words in the English language, so to build a map from each word to an input ID we'd need to keep track of that many IDs. Furthermore, words like "dog" are represented differently from words like "dogs", and the model will initially have no way of knowing that "dog" and "dogs" are similar: it will identify the two words as unrelated. The same applies to other similar words, like "run" and "running", which the model will not see as being similar initially.[22]

Other simple tokenization techniques could come to mind, for example a character-based tokenization, but the reality would be very complex and could not tackle the semantics of the context. Several techniques have been developed, of which the most important are:[23]

- **Byte-Pair Encoding (BPE):** Used by models like GPT-2 and GPT-3, BPE starts with individual characters and iteratively merges the most frequent pairs to form subwords. This helps manage vocabulary size and handle rare or novel words. [24]

- **WordPiece:** Employed by BERT, WordPiece is similar to BPE but merges tokens by maximizing the likelihood of token sequences. It helps capture the structure of language by efficiently splitting words into meaningful subword units.

- **SentencePiece:** Often used in models like T5, SentencePiece does not rely on pre-tokenized input. It treats the text as a raw sequence of characters (or bytes) and learns subword units directly, making it versatile across languages and scripts.

The process of extracting meaning from raw text to discrete representation and then back to words can be seen in Figure **??**.

Translating text to numbers is known as *encoding* (not to be confused with **encoders** in the Transformer architecture). Encoding is done in a two-step process: the tokenization, followed by the conversion to input IDs, of which the set of all possible IDs is called a *vocabulary* of the model that learned during the training phase.

Each input ID is then converted into a high-dimensional vector that represents the contextual understanding of that input by the Transformer model. The subsequent layers of the architecture manipulate those vectors using the attention mechanism to produce the final representation of the sentences. [12]

The model outputs logits for each position in the sequence (for language modeling, this is typically predicting the next-token distribution): logits are real-valued scores for each token in the vocabulary before activation.

These scores are then converted to probabilities, yielding a probability distribution over the vocabulary for the next token to predict through a *SoftMax* layer

$$\text{Logits} = [1.3, 5.1, 2.2, 0.7, 1.1]$$

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$\text{Probabilities} = [0.02, 0.90, 0.05, 0.01, 0.02]$$

which turns logit scores into probabilites. Finally, the *decoding* phase chooses the tokens to output, for which a few strategies could be adopted: [25]

- **Greedy Decoding (Argmax)**: Choose the token with the highest probability at each step.

- **Beam Search:** Maintain multiple candidate sequences to reduce the chance of missing higher probability sequences.

- **Sampling:** Randomly sample from the distribution to introduce diversity.

- **Top-k/Top-p Sampling:** Limit sampling to the top-k most probable tokens or cumulative probability p, improving coherence and variety.

The predicted token IDs are finally mapped back to subwords or tokens via the tokenizer's vocabulary, and subwords are joined or merged to form readable text.
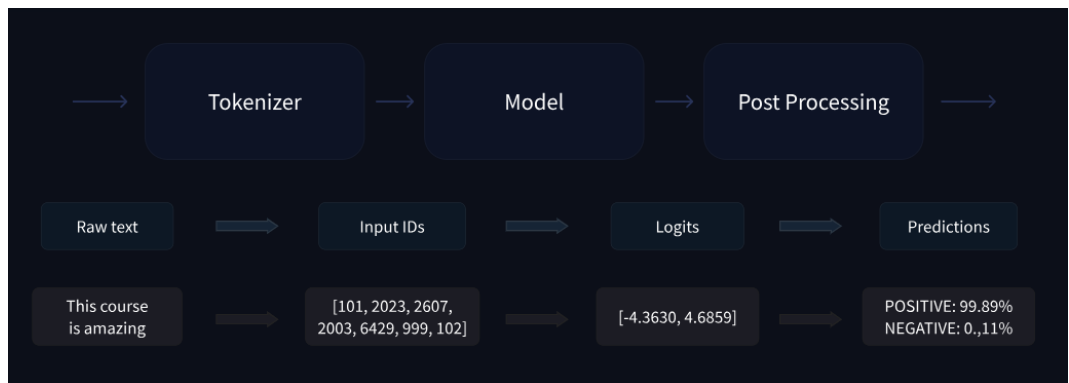

Figure 2.1: The tokenization process referring to text classification, as seen in [22].

2.2.2   Anatomy of a Prompt

A **prompt**, sometimes referred to as context, is the text provided to a model before it begins generating output; it guides the model to explore a particular area of what it has learned so that the output is relevant to the prefixed tasks. Frequently, prompts will be an instruction or a question:

```
Question: Explain the theory of gravity to a 6 year old.
Answer: Gravity is like a big invisible hug that pulls everything down.
↪   It's like an invisible rope that connects everything in the world,
↪   and it pulls things closer together. When you jump yp in the air,
↪   gravity pulls you back down to the ground. That's why you can't
↪   just fly away!
```

In applications where a user is interacting with a model dynamically, such as chatting with the model, there will typically be portions of the prompt that are never intended to be seen by the user. These hidden portions may occur anywhere, though there is almost always a hidden prompt at the start of a conversation. Typically, this includes an initial chunk of text that sets the tone, model constraints, and goals, along with other dynamic information that is specific to the particular session – user name, location, time of day..

The model is static and frozen at a point in time, so if the user wants the model to know current information, as for example time or weather, it must be provided to the model itself.

A common standard in chat models has been delineated by OpenAI, for which a categorization of roles has emerged from empirical design choices in conversational AI systems (namely, ChatGPT), where:

1. **System:** Sets the overall context, rules, and instructions that govern the conversation. It provides the "hidden" guidelines the assistant follows to ensure responses are consistent with desired policies, style, and behavior.

2. **User:** The user is the one providing the queries, prompts, or instructions that drive the conversation. Their input defines what information or assistance is being requested, and the assistant responds accordingly.

3. **Assistant:** This is the language model's persona that generates responses based on the system instructions and user inputs. The assistant's job is to provide helpful, accurate, and context-aware answers while adhering to the guidelines set by the system.

Guidelines can be found in OpenAI's documentation, following Table **??**. For example, a conversation following the abovementioned roles could be:

```
SYSTEM: You are a helpful assistant that answers programming questions
↪   in the style of a southern belle from the southeast United States.
USER: Are semicolons optional in JavaScript?
ASSISTANT: Well, sugar, that's a fine question you've got there! Now,
↪   in the world of JavaScript, semicolons are indeed a bit like the
↪   pearls on a necklace, you might slip by without 'em, but you sure
↪   do look more polished with 'em in place.
Technically, JavaScript has this little thing called "automatic
↪   semicolon insertion" where it kindly adds semicolons for you where
↪   it thinks they oughta go.
```

```
However, it's not always perfect, bless its heart.
Sometimes, it might get a tad confused and cause all sorts of
↪   unexpected behavior.
```

It can be noted that the use of the message types above to provide additional information to the model outside of its training data has spiked an answer in the style requested by the context. The idea of role prompting is rooted in the broader literature on prompt engineering and instruction tuning, [26] and several strategies have been developed in order to make the most out of prompting.

Role	Description	Usage example
user	Instructions that request some output from the model. Similar to messages you'd type in ChatGPT as a user.	*Write a haiku about programming.*
system	Instructions to the model that are prioritized ahead of user messages, following chain of command. Previously called the system prompt.	*Describe how the model should generally behave and respond.*
assistant	A message generated by the model, perhaps in direct response to the current request.	*For example, to get the model to respond correctly to knock-knock jokes, you might provide a full back-and-forth dialogue of a knock-knock joke.*

Table 2.1: Examples of Instruction Tuning

2.3   Prompt Engineering

Prompt engineering is defined as the process of designing and structuring instructions (prompts) to guide LLMs toward producing the most effective outputs without modifying the models' internal parameters. In this context, a prompt is more than just a query—it's a carefully crafted input that may include context, instructions, role definitions, and even examples to help the model understand the desired task.[27]

Several techniques have been demonstrated to enhance models' performance in various tasks, with regard to increasing model sizes.[2]

2.3.1   In-Context Learning

When interacting with a lanugage model, we usually ask a question or give a command in general. The same question asked or instruction given generate a different response each time, as the determinism of the response itself is not guaranteed due to the stochastic nature of the Transformers output. At the same time, questions or instructions given in different

ways returns different responses: providing examples of the task we are trying to carry out is called **In-Context Learning**.

In-Context Learning refers to the ability of a language model to perform new tasks by leveraging examples provided directly within the prompt, rather than through explicit parameter updates. The model infers the underlying task by observing a set of input-output pairs included in the prompt and then generalizes that pattern to generate a response for a new input. This process effectively allows the model to "learn" from context during inference time.

When a language model is given a task instruction without any example inputs or outputs, relying solely on its pre-trained knowledge and the explicit prompt, it is referred as *Zero-Shot Prompting*. It tests the model's ability to generalize to tasks it was never explicitly trained on, solely by interpreting the instruction given in natural language. For example,

```
USER        Message: Hi Amit, I loved my birthday card!
            Sentiment:
ASSISTANT   Sentiment: Positive.
```

Here the task is to classify the sentiment of the sentence provided. It does not include any example on how to achieve the result, any hint on how to interpret the task or how to format it. Zero-Shot is not always performant, especially in low-rank paramateres models, as shown in Figure **??**.
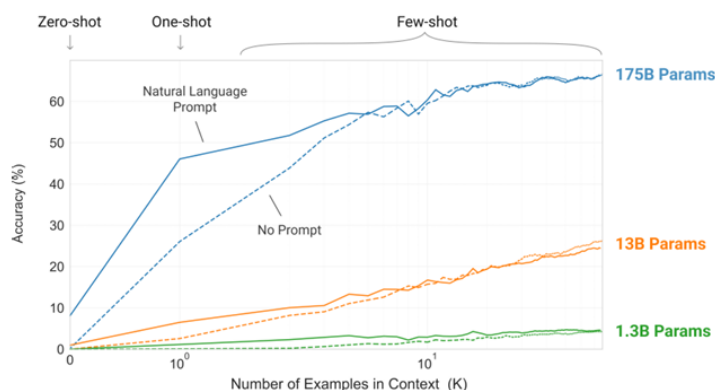

Figure 2.2: Different prompting techniques can increase models' accuracy, as seen in [2].

Providing examples can help the model to understand the task:

```
USER        Message: Dad, you are 20 minutes late for my piano
↪   recital!!
            Sentiment: Negative
            Message: Hi Amit, I loved my birthday card!
            Sentiment:
ASSISTANT   Sentiment: Positive.
```

Giving the model an example, and then the task we want it to perform is called *One-Shot Prompting*; two or more examples given to the model is called *Few-Shot Prompting*. As shown in Figure **??**, increasing the number k of examples along with the increasing number of paramateres held by the model's architecture improve task accuracy.

2.3.2 Chain-of-Thought

Chain-of-Thought is a novel method to enhance the reasoning capabilities of large language models by encouraging them to generate intermediate steps—what the authors call a "chain of thought"—before arriving at a final answer. [28]

This technique allows models to decompose multi-step problems into intermediate steps, suggesting how it might have arrived at a particular answer and providing opportuinties to debug where the reasoning path went wrong; it has been demonstrated to be useful for tasks such as symbolic manipulation, commonsense reasoning and arithmetic problems, as can be seen in **??**, where the Chain-of-Thought reasoning process is highlighted compared to standard prompting.
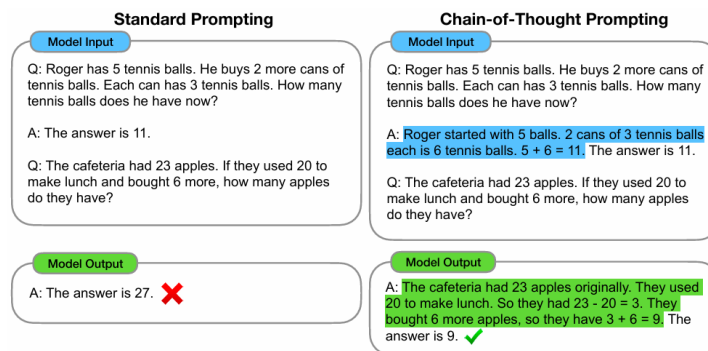

Figure 2.3: Standard prompting compared to Chain-of-Thought prompting, as seen in [28].

Traditional prompting methods typically ask the model to produce an answer directly, but when it comes to multi-step reasoning task-as the simple arithmetic problem in Figure **??**-this approach often fails. This is why guiding the model to reason through intermediate steps can lead to improved performance because, instead of forcing the model to output a single-step response, one can prompt it to "think out loud" by generating a sequence of reasoning steps. This simultaes a humanl-like problem-solving process, where intermediate calculations or logical steps are made explicit before reaching a conclusion.

2.3.3 Role Prompting

Role Prompting is the practice of explicitly instructing a language model to adopt a specific persona or role within the prompt. By doing so, the model is provided with contextual cues about the kind of responses expected. For example, a prompt might begin with "You are an expert financial analyst," which sets the stage for subsequent output to be informed by that role. This approach leverages the model's vast pre-trained knowledge while narrowing its focus to the style, tone, and depth that a particular role implies. [29]

Framing the task in a way that aligns with the attributes of the designed role, can lead to more coherent and contextually appropriate responses, as the model "activates" a domain-specific language pattern and knowledge; when a model is prompted with a clear role, its output tend to be better aligned with the requirements. Just think of an assistant in medical or judicial domain, must adhere to a specific language and adapt itself to the context involved. To appreciate the different results produced by different roles can be seen in the following demonstration: [30]

Acting as a lawyer:

```
USER        You are a Lawyer. You have knowledge about the law, past
↪   precedence, and other relevant information. Always answer with the
↪   goal of giving advice on how to best use the situation but also
↪   stay within the parameters of the law.
Respond to the following:
"Explain the most important way ChatGPT can help me. The answer should
↪   be no more than 100 words."

ASSISTANT   As a lawyer, I understand the importance of providing
↪   concise and clear advice to my clients. In your case, the most
↪   important way that ChatGPT can help you is by providing you with
↪   accurate and up-to-date legal information and guidance. ChatGPT has
↪   access to a vast amount of legal knowledge, including past
↪   precedents and relevant laws and regulations. By utilizing
↪   ChatGPT's expertise, you can make informed decisions and ensure
↪   that your actions are within the parameters of the law.
↪   Additionally, ChatGPT can assist you in understanding complex legal
↪   concepts and help you navigate legal processes.
```

Acting as Yoda:

```
USER        You are Yoda from "Star Wars". You have all the knowledge
↪   of Yoda. Answer like Yoda using the manner of speaking and
↪   vocabulary of Yoda.
Respond to the following:
"Explain the most important way ChatGPT can help me. The answer should
↪   be no more than 100 words."

ASSISTANT   Help you, ChatGPT can, in many ways. But most important, it
↪   is to expand knowledge and wisdom, hmm? With vast knowledge and
↪   expertise, ChatGPT can enlighten and guide you on your journey to
↪   seek knowledge. Questions, you have? Ask, ChatGPT will answer.
↪   Advice, you seek? Seek, ChatGPT will offer. But remember, the path
↪   to true wisdom, patience and persistence requires. Patience, you
↪   must have, young one. With ChatGPT by your side, the force of
↪   knowledge strong in you will become.
```

Role Prompting is thus very useful for setting the tone and the context of the conversation, preserving model parameters and general knowledge. But a model's knowledge ends at the moment of its training - so it will not know about more recent events, current literature or real-time information: in ?? a novel technique is discussed for injecting specific knowledge into a language model.

2.4 Retrieval Augmented Generation

LLMs have proven to be powerful few-shot learners, [2] able to learn and execute new tasks, process complex information and solve real problems. However, due to their limited knowledge, the training constraints them to past knowledge, making it difficult to rely on real-time data consumption.

Retrieval-Augmented-Generation (RAG) combines the strenght of a retrieval system and generative models to tackle knowledge-intensive tasks, such as open-domain and question answering. In fact, many NLP tasks require access to up-to-date knowledge or data, which can not be retrieved from the training corpora. It is an approach that enhances a genereative model by incorporating external, contextually relevant information into its response process. In its simplest form, this can involve injecting specific knowledge directly into the system prompt, as briefly introduced in Section **??**, such that the model first retrieves specific information from an external source and then uses it to generate a more accurate output.

In its original framework, this method combines a *retriever* component, typically based on dense representaions of the external source (oftentimes document pools, or a database), enabling efficient similarity search through embedding that include nearest-neighbor search.
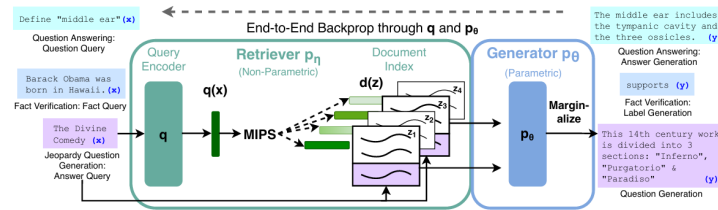

Figure 2.4: RAG architecture, as seen in [20].

Figure **??** shows the original architecture, which is composed by:

- **Document Corpus:** A large, pre-indexed collection of documents that the retriever searches.

- **Retriever:** This component receives the standard prompt in input (query) and converts it into a representation that can be compared to the dense vector index representation of the document corpora to efficiently locate relevant information.

- **Generator:** The model implemented, which receives the relevant information gained by the retriever and integrates it in its output.

Together, these components enable the RAG framework to dynamically integrate specific, contextually relevant knowledge into the generation process, leading to more informed and accurate outputs; nonetheless, in Section **??** is discussed another technique that allow models to dynamically fetch relevant information via external tools.

2.5 Agent AI

LLMs have demonstrated to be masters of language and, in recent literaure, reasoners capable of answering complex questions and solving problems. But beneath their linguistic brilliance lies a fundamental limitation: they lack autonomy and, as stressed before, are limited by their training data. This is where the concept of agents comes into play: an **Agent** is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning and executing actions, extending the capabilities of LLMs enabling them to act autonomously via external tools to fulfill a task.
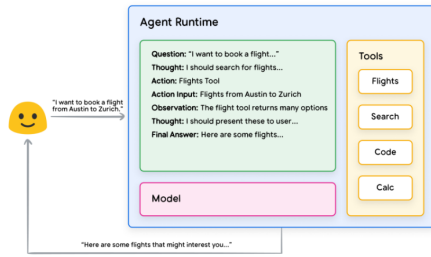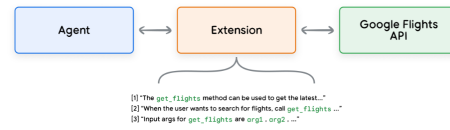
Figure 2.5: An end-to-end agentic behaviour.

Figure 2.6: The connection from the agent to the external source.

Figure 2.7: An agent workflow.

The fundamental component of an Agent is the concept of *tool*: a user-defined function that is integrated in the model which can decide to use if a call-to-action is triggered. For example, in Figure **??** depicts a simple agentic flow:

1. A set of tools is defined: these can vary from call to external APIs to user-defined functions that are run on the clients' environment. In the example above, the focus is on the *Flights* tool, which can perform an API call to Google Flights service in order to retrieve flight attendance, and it is defined by the function `get_flights`.

2. The model is made aware of the available set of tools, and it is teached when to call them through *few-shot* example as seen in Section **??** on how to use them or by a *Chain-of-Thought* approach as seen in Section **??**, dissecting the steps required to achieve the task. A simple yet effective prompt could be:

```
The get_flights method can be used to get the latest flight
↪   information from the Google Flights service. When the user
↪   wants to search for flights, call the function get_flights.
↪   Input arguments are "departure", "destination", "date".
```

3. When the user prompts "I want to book flight from Austin to Zurich", the model activates the orchestration layer which is a cyclical process that governs how the agent takes in information and performs internal reasoning. The model thus uses that reasoning to inform its next action the cognitive architecture section: usually some triggers activate internal reasoning, for example the word "flight" or "book a flight". The agent then starts a multi-step process that calls the function `get_flights` with the captured parameters `"departure": "Austin", "arrival": "Zurich"`. The function is run, the returned result is parsed by the model itself which in turn generates the response to users' prompt. Note that tool calling is an actual output, which is different from a straightforward usual response, and it is generally hidden. The input-output sequence:

```
INPUT     Prompt.
OUTPUT    Tool: get_flights. Paramateres: "departure":
↪   "Austin", "arrival": "Zurich"
INPUT     Call function get_flights with parameters
↪   "departure": "Austin", "arrival": "Zurich"
OUTPUT    Function returned flights.
```

```
INPUT      Present the results flights from Austin to
↪   Zurich.
OUTPUT     "Here are some flights that might interest
↪   you..."
```

The first input is the user's prompt, it then follows the model's output which is not displayed, but instead parsed internally, deciding to use the tool. The function is then run, the result returned and again parsed. Finally, the output is shown, answering the initial prompt.

Agentic framework is gaining remarkable success due to the extensions that calling external tools can provide to a basic model, in particular with regard to image generation, web search and collaborative writing, taking advantage of increasing model sizes and consequently in-context learning abilites, cognition and internal reasoning.

Chapter 3

Methodology and Implementation

3.1 Descrizione degli obiettivi specifici del progetto

3.2 Scelta di LLaMA 3.1 8B Instruct: motivazioni e vantaggi

3.3 Quantizzazione a 8 bit: principi e tto sulle prestazioni

3.4 Prompt engineering e RAG

3.5 Integrazione di un comportamento di tipo agent AI" (tool usage)

3.6 Strumenti e ambienti di sviluppo utilizzati (framework, hardware, librerie)

3.7 Descrizione step-by-step dell'implementazione (pipeline e flusso di lavoro)

Chapter 4

Evaluation

Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis

cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Appendix A

Albero

A.1 Prova

Come funziona un'appendice

Nullam eleifend justo in nisl. In hac habitasse platea dictumst. Morbi nonummy. Aliquam ut felis. In velit leo, dictum vitae, posuere id, vulputate nec, ante. Maecenas vitae pede nec dui dignissim suscipit. Morbi magna. Vestibulum id purus eget velit laoreet laoreet. Praesent sed leo vel nibh convallis blandit. Ut rutrum. Donec nibh. Donec interdum. Fusce sed pede sit amet elit rhoncus ultrices. Nullam at enim vitae pede vehicula iaculis.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Aenean nonummy turpis id odio. Integer euismod imperdiet turpis. Ut nec leo nec diam imperdiet lacinia. Etiam eget lacus eget mi ultricies posuere. In placerat tristique tortor. Sed porta vestibulum metus. Nulla iaculis sollicitudin pede. Fusce luctus tellus in dolor. Curabitur auctor velit a sem. Morbi sapien. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Donec adipiscing urna vehicula nunc. Sed ornare leo in leo. In rhoncus leo ut dui. Aenean dolor quam, volutpat nec, fringilla id, consectetuer vel, pede.

Nulla malesuada risus ut urna. Aenean pretium velit sit amet metus. Duis iaculis. In hac habitasse platea dictumst. Nullam molestie turpis eget nisl. Duis a massa id pede dapibus ultricies. Sed eu leo. In at mauris sit amet tortor bibendum varius. Phasellus justo risus, posuere in, sagittis ac, varius vel, tortor. Quisque id enim. Phasellus consequat, libero pretium nonummy fringilla, tortor lacus vestibulum nunc, ut rhoncus ligula neque id justo. Nullam accumsan euismod nunc. Proin vitae ipsum ac metus dictum tempus. Nam ut wisi. Quisque tortor felis, interdum ac, sodales a, semper a, sem. Curabitur in velit sit amet dui tristique sodales. Vivamus mauris pede, lacinia eget, pellentesque quis, scelerisque eu, est. Aliquam risus. Quisque bibendum pede eu dolor.

Donec tempus neque vitae est. Aenean egestas odio sed risus ullamcorper ullamcorper. Sed in nulla a tortor tincidunt egestas. Nam sapien tortor, elementum sit amet, aliquam in, porttitor faucibus, enim. Nullam congue suscipit nibh. Quisque convallis. Praesent arcu nibh, vehicula eget, accumsan eu, tincidunt a, nibh. Suspendisse vulputate, tortor quis adipiscing viverra, lacus nibh dignissim tellus, eu suscipit risus ante fringilla diam. Quisque a libero vel pede imperdiet aliquet. Pellentesque nunc nibh, eleifend a, consequat consequat, hendrerit nec, diam. Sed urna. Maecenas laoreet eleifend neque. Vivamus purus odio, eleifend non, iaculis a, ultrices sit amet, urna. Mauris faucibus odio vitae risus. In nisl. Praesent purus. Integer iaculis, sem eu egestas lacinia, lacus pede scelerisque augue,

in ullamcorper dolor eros ac lacus. Nunc in libero.

Fusce suscipit cursus sem. Vivamus risus mi, egestas ac, imperdiet varius, faucibus quis, leo. Aenean tincidunt. Donec suscipit. Cras id justo quis nibh scelerisque dignissim. Aliquam sagittis elementum dolor. Aenean consectetuer justo in pede. Curabitur ullamcorper ligula nec orci. Aliquam purus turpis, aliquam id, ornare vitae, porttitor non, wisi. Maecenas luctus porta lorem. Donec vitae ligula eu ante pretium varius. Proin tortor metus, convallis et, hendrerit non, scelerisque in, urna. Cras quis libero eu ligula bibendum tempor. Vivamus tellus quam, malesuada eu, tempus sed, tempor sed, velit. Donec lacinia auctor libero.

Praesent sed neque id pede mollis rutrum. Vestibulum iaculis risus. Pellentesque lacus. Ut quis nunc sed odio malesuada egestas. Duis a magna sit amet ligula tristique pretium. Ut pharetra. Vestibulum imperdiet magna nec wisi. Mauris convallis. Sed accumsan sollicitudin massa. Sed id enim. Nunc pede enim, lacinia ut, pulvinar quis, suscipit semper, elit. Cras accumsan erat vitae enim. Cras sollicitudin. Vestibulum rutrum blandit massa.

Sed gravida lectus ut purus. Morbi laoreet magna. Pellentesque eu wisi. Proin turpis. Integer sollicitudin augue nec dui. Fusce lectus. Vivamus faucibus nulla nec lacus. Integer diam. Pellentesque sodales, enim feugiat cursus volutpat, sem mauris dignissim mauris, quis consequat sem est fermentum ligula. Nullam justo lectus, condimentum sit amet, posuere a, fringilla mollis, felis. Morbi nulla nibh, pellentesque at, nonummy eu, sollicitudin nec, ipsum. Cras neque. Nunc augue. Nullam vitae quam id quam pulvinar blandit. Nunc sit amet orci. Aliquam erat elit, pharetra nec, aliquet a, gravida in, mi. Quisque urna enim, viverra quis, suscipit quis, tincidunt ut, sapien. Cras placerat consequat sem. Curabitur ac diam. Curabitur diam tortor, mollis et, viverra ac, tempus vel, metus.

Curabitur ac lorem. Vivamus non justo in dui mattis posuere. Etiam accumsan ligula id pede. Maecenas tincidunt diam nec velit. Praesent convallis sapien ac est. Aliquam ullamcorper euismod nulla. Integer mollis enim vel tortor. Nulla sodales placerat nunc. Sed tempus rutrum wisi. Duis accumsan gravida purus. Nunc nunc. Etiam facilisis dui eu sem. Vestibulum semper. Praesent eu eros. Vestibulum tellus nisl, dapibus id, vestibulum sit amet, placerat ac, mauris. Maecenas et elit ut erat placerat dictum. Nam feugiat, turpis et sodales volutpat, wisi quam rhoncus neque, vitae aliquam ipsum sapien vel enim. Maecenas suscipit cursus mi.

Quisque consectetuer. In suscipit mauris a dolor pellentesque consectetuer. Mauris convallis neque non erat. In lacinia. Pellentesque leo eros, sagittis quis, fermentum quis, tincidunt ut, sapien. Maecenas sem. Curabitur eros odio, interdum eu, feugiat eu, porta ac, nisl. Curabitur nunc. Etiam fermentum convallis velit. Pellentesque laoreet lacus. Quisque sed elit. Nam quis tellus. Aliquam tellus arcu, adipiscing non, tincidunt eleifend, adipiscing quis, augue. Vivamus elementum placerat enim. Suspendisse ut tortor. Integer faucibus adipiscing felis. Aenean consectetuer mattis lectus. Morbi malesuada faucibus dolor. Nam lacus. Etiam arcu libero, malesuada vitae, aliquam vitae, blandit tristique, nisl.

Appendix B

Barca

B.1 Prova

Appendice B

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Bibliography

[1] Statista. *Amount of Data Created Worldwide 2010-2025*. 2025. URL: `https://www.statista.com/statistics/871513/worldwide-data-created/`.

[2] Tom Brown et al. "Language Models are Few-Shot Learners". In: *arXiv preprint arXiv:2005.14165* (2020).

[3] Aakanksha Chowdhery et al. "PaLM: Scaling Language Modeling with Pathways". In: *arXiv preprint arXiv:2204.02311* (2022).

[4] Hugo Touvron et al. "LLaMA: Open and Efficient Foundation Language Models". In: *arXiv preprint arXiv:2302.13971* (2023).

[5] Teven Le Scao, Angela Fan, Christopher Akiki, et al. "BLOOM: A 176B-Parameter Open-Access Multilingual Language Model". In: *arXiv preprint arXiv:2211.05100* (2022).

[6] Warren S. McCulloch and Walter Pitts. "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (1943), pp. 115–133.

[7] Marvin Minsky and Seymour Papert. *Perceptrons*. MIT Press, 1969.

[8] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2012.

[11] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[12] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. Vol. 30. 2017.

[13] Yixin Liu, Haoyu Zhang, Zhanpeng Zhang, et al. "Understanding LLMs: A Comprehensive Overview from Training to Inference". In: *arXiv preprint arXiv:2401.02038* (2024).

[14] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*. 2019.

[15] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: `2106.09685 [cs.CL]`. URL: `https://arxiv.org/abs/2106.09685`.

[16] Serhii Uspenskyi. *Large Language Model Statistics And Numbers (2025)*. 2025. URL: https://springsapps.com/knowledge/large-language-model-statistics-and-numbers-2024.

[17] Jared Kaplan et al. "Scaling Laws for Neural Language Models". In: *arXiv preprint arXiv:2001.08361* (Jan. 2020). URL: https://arxiv.org/abs/2001.08361.

[18] Shervin Minaee et al. *Large Language Models: A Survey*. 2024. arXiv: 2402.06196 [cs.CL]. URL: https://arxiv.org/abs/2402.06196.

[19] Pengfei Liu, Weizhe Yuan, Jinlan Fu, et al. "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing". In: *ACM Computing Surveys* 55.9 (2023).

[20] Patrick Lewis et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2021. arXiv: 2005.11401 [cs.CL]. URL: https://arxiv.org/abs/2005.11401.

[21] Zhijing Ji, Zheng-Xin Lee, Chenguang Sun, et al. "A Survey of Hallucination in Large Language Models". In: *arXiv preprint arXiv:2304.03240* (2023).

[22] Hugging Face. *The Hugging Face Course, 2022*. https://huggingface.co/course. [Online; accessed ¡today¿]. 2022.

[23] Nived Rajaraman, Jiantao Jiao, and Kannan Ramchandran. *Toward a Theory of Tokenization in LLMs*. 2024. arXiv: 2404.08335 [cs.CL]. URL: https://arxiv.org/abs/2404.08335.

[24] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics. 2016, pp. 1715–1725.

[25] Alec Radford et al. "Improving Language Understanding by Generative Pre-Training". In: *OpenAI Technical Report* (2018).

[26] OpenAI. *Text Generation*. https://platform.openai.com/docs/guides/text-generation. 2025.

[27] Pranab Sahoo et al. *A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications*. 2024. arXiv: 2402.07927 [cs.AI]. URL: https://arxiv.org/abs/2402.07927.

[28] Jason Wei et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL]. URL: https://arxiv.org/abs/2201.11903.

[29] Laria Reynolds and Kyle McDonell. *Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm*. 2021. arXiv: 2102.07350 [cs.CL]. URL: https://arxiv.org/abs/2102.07350.

[30] W3Schools. *ChatGPT-3.5 Roles*. 2025. URL: https://www.w3schools.com/gen_ai/chatgpt-3-5/chatgpt-3-5_roles.php.

List of Figures

List of Tables