POLITECNICO DI TORINO

Master's Degree in Computer Engineering - Artificial Intelligence and Data Analytics

Master's Degree Thesis

Natural Language Processing with Generative AI Models: A Methodological Approach for Their Application

Supervisor:

Prof. Fabrizio LAMBERTI

Company tutor:

Giorgia FORTUNA

Candidate

Stefano STRIPPOLI

October 2023

Abstract

In this thesis work, a methodological approach is proposed for application in Natural Language Processing (NLP) tasks exploiting the capabilities of latest Generative Artificial Intelligence (Generative AI) text-to-text models. With this methodology, it will be possible to delineate data features scope effectively, enhance model prompting based on expected input and output, and appropriately select the most suitable Generative model for the required NLP task.

The dissertation begins with an analysis of the foundations of the subject, providing an overview of both NLP architectures, from Recurrent Neural Networks to today's Transformer, and Generative AI state-of-the-art models from GoogleAI, MetaAI and OpenAI. Then the methodology is presented describing its phases, the different options to treat different features of data (such as data sensitivity and input formats) along with the generative model selection process and performance evaluation criteria. Next the methodology is applied to 10 different NLP tasks, from simple comprehension tasks to a difficult and temporally articulated assignment involving a real robotic agent such as Boston Dynamics' SPOT. The effectiveness of the methodology is discussed, evaluating its performance, underlining strengths, weaknesses and possible ways to solve them. Finally, future developments and solutions, helpful in enhancing the linguistic comprehensions of Generative models, are presented.

Acknowledgements

First of all, and before anyone else, I would like to thank my family for believing in my potential and for giving me the opportunity to reach where I am today.

Major thanks to Professor Lamberti for his immense availability and the excellent advices provided for the drafting of this thesis project. I am also thankful to Giorgia, Giacomo and the entire Machine Learning Reply team for the given opportunity and for making me feel, even if for a short time, a part of a great environment.

I thank Simone for always, always, showing me the reality beyond my unrealistic expectations and for being there for me in any difficulty along this journey. Even when I didn't believe in myself, I always had a fan (two, considering Iron) ready to support me and lift me up. I can never thank you enough.

I am also grateful to my life journey companions, whether in school desks, Villa Claretta beds, or Arturo's wine carafes; you have been a fundamental part of this journey and you have shaped me into the person I am today. Thank you for listening and understanding me, for always being a source of comfort in sweet moments and a wake-up call in confused times.

"We can only see a short distance ahead,
but we can see plenty there that needs to be done."
Alan Turing

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the last 80 years, Artificial Intelligence (AI) paved its way into modern society: from the first program able to play a chess game written by Alan Turing in 1951 to modern autonomous driving vehicles, the key role of AI world has always been the machine reproduction of human cognitive capabilities such as thinking, recognizing patterns and making decision. One of the first branch of AI world was Machine Learning (ML) in which researchers focused their intents on the creation of models enabled to learn from data and make predictions or decisions without being explicitly programmed to, but only looking at the characteristics of the data. The more the technologies improved, especially by increasing machines computational power, the more researchers started enlarging the field of application of ML algorithms: with the parallelization provided by GPUs, researchers could develop models able to mock how our brain computes external inputs using previous experience and multiple input at the same time. Using very complex models known as Neural Networks, researchers were able to catch intrinsic and deeper correlations between data samples, improving the classifications techniques of heterogeneous type of data such as images, sounds or video. In this scenario, Natural Language Processing (NLP) sunk its root: with NLP, Deep Learning models are used in order to understand the underlying characteristics of human language, such as context, named entity inside a text or figures of speech.

Due to the priceless value of human communications in business world, NLP became a high valuable sector of investment inside businesses as it offers several significant advantages that can positively impact various operational and strategic aspects:

- The evaluation of the sentiment of a product through comments, posts and reviews can lead to specific production decisions.
- The implementation of chatbots, able to communicate properly with the client, can automatize the support center procedures.

- With personal advertisement based on client's interests and previous choices, a website can propose products that the client will be more interested in buying,

At the same time, Generative AI became the main trend of research in different AI laboratories of major companies: representing a specific branch in which models are capable of creating unique and never seen multimedia content such as images, video or text practically indistinguishable to a human-created one. Generative AI played a key role in improving NLP technologies in recent years. Newest models from top-tier AI companies obtained high-quality results in tasks such as language translation, question-answering, content creations and speech recognition systems. It becomes evident, therefore, the interest of a company in integrating these generative models to enhance the performance achieved by simple NLP models: for example, the integration of a generative chatbot model will make the performance comparable to a human, as responses will be generated in real-time based on the question, making the interaction with the virtual operator more human-like and extracting much more information from the text.

The bottleneck of this integration lies in the vastness of the generative landscape, combined with its ever-evolving nature due to the continuous release of new models, which can make researching and integrating these models a challenging choice.

The methodology proposed in this thesis work fits into this scenario: with this approach, the aim is to standardize the process of analyzing the required tasks with relative inputs and outputs, selecting the model and evaluating the performance while minimizing the temporal and economic costs. Through this standardization, a more detailed perimeter evaluation of the features can be created, systematically framing the task and determining the most suitable model to choose. Models handling private and sensitive data, for example, may not be able to use external vendors and must integrate models internally within the company, thus requiring downloadable open-source models.

In this process, the development and progresses of both NLP and Generative AI worlds were analyzed in order to obtain a complete view of the subject of the research, from their beginnings to newest models offered by prominent companies. Having assessed the capabilities of these models and their suitability for various tasks, 10 tasks that span a wide spectrum of complexity. These tasks were chosen to provide a comprehensive overview of the natural language processing landscape, from straightforward tasks to more intricate ones. Moreover, with the last task belonging to the realm of robotics and language translation, the model was incrementally guided to generate the desired output, illustrating its adaptability and learning capabilities in three different scenarios.

Chapter 2

State of the Art

This chapter will provide an overview of the evolution of NLP structures and the actual state of the art of generative models, with a focus on the models developed by the three leading companies in Generative AI: OpenAI [1], Meta AI [2], and Google AI [3].

2.1 Natural Language Processing

NLP refers to artificial intelligence algorithms capable of analyzing, representing, and thus understanding natural language. The purposes can vary from content comprehension to translation, even to autonomously generating text based on input data or documents. NLP mainly deals with texts, understood as sequences of words in a language that convey one or more messages (e.g., web pages, posts, tweets, logs, business information), while speech processing (speech recognition) is considered a separate field.

The interaction between humans and machines involves various aspects, such as phonetics, phonology, morphology, syntax, semantics, pragmatics, and discourse as a whole. Consequently, there are numerous NLP tasks that automate these areas, Today, NLP confronts us with the analysis of complex sentences that, in order to be correctly interpreted, must be broken down into elementary units: words. In addition to analyzing individual words, understanding the semantics of the entire sentence is necessary. From a technical perspective, to transition from analyzing individual words to comprehending the sentence as a whole (Natural Language Understanding),

2.1.1 Word Embeddings

Word embeddings consists in a numeric word representations understandable by machines. One of the first application of word embedding was represented by N-gram models [4] which used this numerical representation to evaluate only statistical patterns inside the sentence, such as word frequencies. With the Word2Vec [5] approach, introduced by Google in 2013, models started to evaluate these relations: using a two layer Neural Network, models were able to create high quality word embeddings in a OneHot representation using two different techniques, CBOW and SkipGram:

- With CBOW (Continuous Bag of Words) the objective is the prediction of a target word given a specific context window of tokens. For example, with a context window C=2 and the input "The kid runs fast", CBOW tries to predict the word "runs" given "The" and "kid" as context window. CBOW sums the context words vectors and tries to predict the target word vector representation. CBOW becomes useful in all the cases in which the context provides meaningful information about the target word.
- Skip-Gram, on the contrary, tries to predict the context (intended as surrounding words) given the target word: with the word "kid", Skip-Gram would try to predict "the" and "kid". In this case, the output will be the vector representation of the context. This technique is more effective when dealing with larger datasets and when capturing semantic relations between words.

These vectorial representations were able to catch both semantic and syntactic relations between words (as shown in Fig. 2.1) and since each embedding dimension represents different aspects of the word's meaning, models can use them to capture word similarities: similar words will produce similar embeddings. Due to this property, Word2Vec has become useful for all downstream NLP tasks such as sentiment analysis or language modeling.

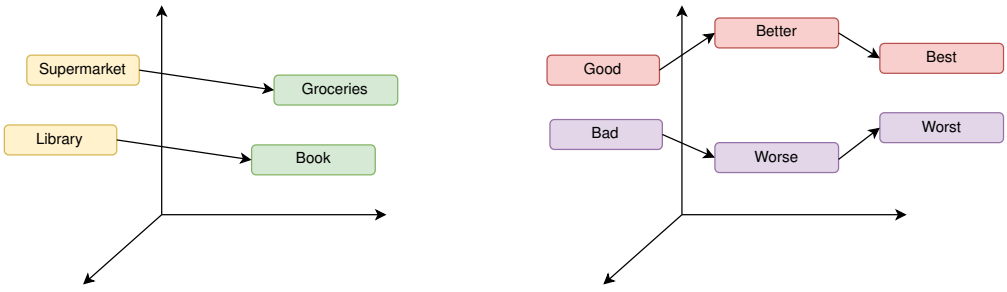

Figure 2.1: Semantic and Syntactic relations.

However, due to the word-centered nature of the algorithm, its training time was proportional to the number of words in the corpus multiplied by the number

of dimensions in the word embeddings. Moreover, Word2Vec can lead to problems with polysemous words: since the embeddings are context-independent, they do not consider the possible different meanings of a specific word but only look at how the word is distributed inside a text corpus.

One solution to the lack of contextualization problem is provided by the Recurrent Neural Network (RNN) architecture [6]. Unlike classic feed-forward NNs, RNNs keep a hidden state that updates at each time step and acts as a memory of the previous inputs. Each word token is processed sequentially based on the content of this hidden state, which serves as a compressed representation of the previous inputs. However, the main issue with RNN architectures is the vanishing gradient problem: with long sentences (and thus, a large number of time steps), the gradient tends to shrink exponentially during backpropagation resulting in slow or even no updates on the latest layers of the model, leading to poor performance.

2.1.2 Attention

In order to solve the vanishing gradient problem, different solutions were deployed, such as LSTM (Long Short Term Memory) [7], which will be the backbone of the future models like ELMo [8], or the promising mechanism of Attention developed by Bahdanau et al. [9] . Focusing on the latter, the main idea behind the attention is to replicate human behaviour in language translation by processing a full sentence and then focusing on the most valuable parts of that specific input phrase rather than evaluate each word sequentially.

An improved version of the attention mechanism is the self-attention [10]: while Bahdanau attention weights are calculated based on the similarity between the current decoder hidden state and the encoder output at each time step, the self-attentions weights are computed based on the similarity between the input and all the previous words in a certain sequence. This process can be described in 3 phases:

1. Input Representation: the input data is transformed into a set of Query, Key, and Value vectors, each one representing a particular attention weight. The three vectors represent, respectively, the current context or state of the model, the set of feature vectors extracted from the input data and the actual content or information associated with each element in the input sequence.

2. Scoring: the model computes a set of scores (using a dot product between key and query vectors) that indicate the relevance of each key vector to the query vector.

3. Aggregation: the mechanism combines the value vectors with the scores to produce a weighted sum or other aggregation of the values. The weights used

in the aggregation are typically determined by applying a softmax function to the scores, which produces a set of normalized weights that sum to one (Eq. 2.1).

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

Q, K and V are respectively the Query, Key and Value vectors and d_k is the dimension of Key/Query vector.

2.1.3 Transformer

Google implemented the self-attention mechanism in the Transformer [10] architecture, which is a neural network architecture that consists of an Encoder and a Decoder. The Encoder creates an embedded representation of the token sequence in the input, and the Decoder processes this representation to generate an output. The Encoder has 6 layers, each comprising a Multi-Head Self-Attention layer and a Feed Forward Neural Network layer. The Decoder has the same 6 layers as the Encoder, with an additional layer in each that performs Multi-Head Self-Attention on the Encoder's output. In multihead self-attention, the attention mechanism is expanded by using multiple parallel attention heads with their own set of learnable parameters (query, key, and value vectors) and computes its own attention weights independently. In this way, the model is allowed to catch different aspects and patterns of the input sequence simultaneously since each attention head can focus on different parts of the sequence, capturing different relationships and have a more complete global vision of all the dependencies of the input data.

A specific description of the whole Transformer architecture and workflow, as shown in Fig. 2.2, is listed below:

1. The input tokens are converted into a dense vector representation by multiplying them with a learned embedding matrix to create input embeddings.

2. The position of each token in the input sequence is traced using a function called Positional Encoding, which maps each position of the sequence to a vector that is added to the input embedding to enable the model to distinguish all positions in the sequence.

3. The embeddings are processed by the Decoder by the Multi-Head Attention layer, which computes attention scores between every pair of positions in the sequence and uses them to compute a weighted sum of the input embeddings.

4. The output of the Multi-Head Attention layer is passed through a Feedforward Neural Network, which consists of two linear projections with a ReLU activation

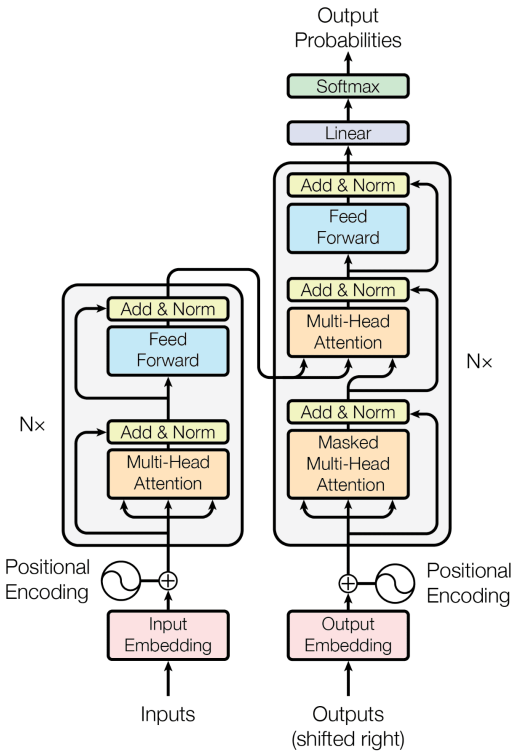

Figure 2.2: Transformer architecture [10].

in between. This allows the model to perform non-linear transformations of the input.

5. Residual connections and a Normalization layer are used respectively to propagate the gradient better and to normalize the output at zero mean and unit variance, ensuring gradient stability.

6. The output of the Feedforward Network is passed through a linear layer and converted into a probability distribution by a softmax activation function. The model then selects the token with the highest probability as the output token, which is fed back into the model for the next step until the end-of-sequence token is generated or the maximum output length is reached.

The Transformer implementation solved most of the previous architecture's problems, such as vanishing gradients, long-term dependencies, flexible and parallelizable inputs. On detail, input parallelization allowed a more robust training with more larger datasets.The implementation of the transformer into NLP model created a new branch of models called Large Language Model (LLM). LLMs are in fact

characterized by their numbers of training data and model parameters (in the order of billions), in particular a sub family of LLMs is the Pretrained Language Models: Pretrained Language Models are LLMs able to learn high-quality general language structures, trained using unsupervised or self-supervised learning technique and then finetuned for a specific task.

The first notorious architecture to achieve state of the art performances was introduced by Google in 2018: BERT [11] (Bidirectional Encoder Representations from Transformer) used the computational power of Transformer's Encoder with an innovative technique of language modelling called Masked Language Modelling. With this technique, a fixed amount of input tokens are substitute with a special [MASK] token and then the model is trained to predict the masked work based on the context of the whole phrase (Fig. 2.3) with both a left-to-right approach (learning from previous words) and with a right-to-left approach (learning from next words).

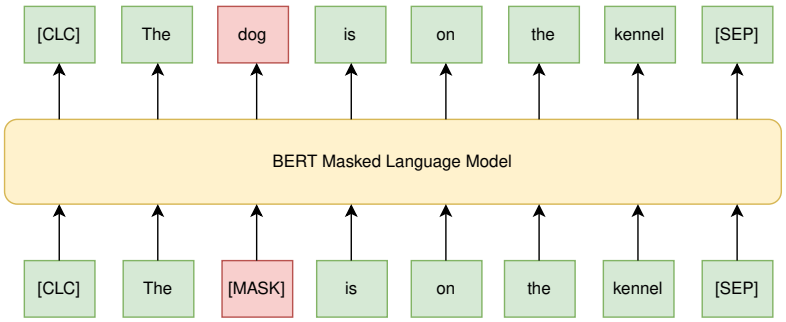

Figure 2.3: Masked Language Model flowchart.

2.2 Generative AI

Generative AI is a subfield of AI whose objective is to develop models that can create new data that is similar and coherent to the input they are generated from. The application fields of Generative AI range across different sectors:

- The creative feature of Generative AI to generate new, unseen contents finds application in all areas related to art, such as creating new poems, song lyrics or in visual arts;
- The possibility to customize the content based on the provided inputs becomes fundamental in the advertising field, generating suggested content that is suitable for the user.
- With the generation of novel content, Generative AI finds application in the area of Data Augmentation, creating synthetic data used for training other

Machine Learning models.

This work is focused on text-to-text tasks of Generative AI. Text-to-Text tasks aims to create a mapping between a textual input, given to the model as sentences or simple keywords, and a generate relevant and coherent textual output in a natural language form. This field finds a lot of applicability: code sourcing, text summarization, question answering, cloze task. In this section, the state of the art models from OpenAI, Google and Meta, will be evaluated, discussing their architectures and their performances in various benchmarks.

2.2.1 GPT-1

GPT [12] (Generative Pretrained Transformer) is a series of generative models produced by OpenAI that has established itself as a pioneer in the field of generative natural language processing over the past 5 years, capturing the attention of both researchers for its potential and the masses for the potential dangers that its abilities can provoke.

In 2018, the first iteration of GPT models was released. GPT, nowadays commonly referred as GPT-1, was born almost concurrently with BERT and was a PLM based on multiple Transformer layers. Specifically, the model consisted of a 12-layer decoder-only transformer with masked self-attention heads, followed by a softmax layer. The two fundamental aspects of the model were unsupervised pre-training and supervised finetuning:

- The pre-training consists of predicting the next word given the sequence of previous words by the maximization of the loss function (Eq. 2.2):

$$\mathcal{L}(\mathcal{U}) = \sum_{i=1}^{n} \log P(u_i | u_i - k, \ldots, u_{i-1}; \Theta) \tag{2.2}$$

  where $\mathcal{U}$ is the unsupervised corpus of tokens, k is the size of the context window and P is the conditional probability modeled using a neural network with parameters Θ. These parameters are trained using stochastic gradient descent which consist on an iterative algorithm that aims to find the optimal set of parameters (weights and biases) that minimize the loss function of a model: at each iteration, a randomly selected subset of training examples (known as a mini-batch) is used to compute the gradient with the respect of the parameters of the mini-batch and then the parameters are updated by taking a step in the opposite direction of the gradient, scaled by a learning rate.

- The finetuning aims to adapt the pretrained model for a specific supervised task: the new dataset $\mathcal{C}$ with its instances as tokens $[x_1, x_2, \ldots, x_m]$ along with

a label y, is passed to the pre-trained model, which weight will be froze, in order to obtain the final transformer's block activation, which will be fed to a new added linear output layer with parameters W_y to predict y (Eq. 2.3).

$$P(y|x^1, \ldots, x^m) = softmax(h_l^m W_y) \tag{2.3}$$

In this way, the new objective function is obtained (Eq. 2.4).

$$L_2(\mathcal{C}) = \sum_{x,y} \log P(y|x^1, \ldots, x^m) \tag{2.4}$$

The strength of GPT-1, compared to previous models, is based on the number of parameters used, which was 117 million, and the number of documents used during the training phase. Additionally, the unsupervised nature of the training phase eliminated the need for human supervision of data and hand-labeling. The dataset used for training was the BookCorpus [13], which included about 11,000 books and a total of 800 million words. Despite the model's strong potential in various NLP benchmarks, such as next word prediction and automatic text fill, it was not very effective in capturing and processing information outside of the scope of knowledge in which it was trained, generating repetitive and nonsensical texts. Moreover, the language capabilities of the model deteriorated as the length of the sentences increased. This was due to the model's limited long-term memory, which was already restricted to 512 characters, resulting in a loss of fluency and coherence in the responses.

2.2.2 GPT-2

The weaknesses of GPT-1 set the stage for the next iteration: GPT-2 [14]. With GPT-2, OpenAI maintained the same model architecture as GPT-1, with some small changes regarding the normalization layer, and the same methodology for both pretraining and finetuning phases. The main innovations in this model consisted of the number of parameters used, with four different versions based on the size of the model (Table 2.1), with the largest having 1.5 billion parameters, and the fact that the model was multi-task: it was not trained for a specific task (just like GPT-1), but it was trained on a new training dataset composed of WebText, a collection of 40 GB of textual information from various web pages created by OpenAI, and CommonCrawl [15], allowing the model to broaden its knowledge and task applicability. Moreover, the token length was doubled, with a maximum length of 1024 characters. GPT-2 achieved state-of-the-art performance on zero-shot performance on 7 out of 8 tested language modeling datasets but still had poor performance in text summarization.

Version	Parameters	Size
GPT-2 "Small"	117 million	473 MB
GPT-2 "Medium"	345 million	1.5 GB
GPT-2 "Large"	774 million	3.2 GB
GPT-2 "XL" (GPT-2)	1.5 billion	6.7 GB

Table 2.1: Parameters for each GPT-2 version.

2.2.3 GPT-3

Building on the success of GPT-2's in-context learning, OpenAI sought to further improve performance by scaling the model. In June 2020, they introduced GPT-3 [16], an autoregressive model with 175 billion parameters, three orders of magnitude larger than its predecessor (Table 2.3). The model is based on the same architecture as GPT-2, but with alternating dense and locally banded sparse attention patterns in the transformer layers, similar to the Sparse Transformer[17]. OpenAI trained eight different models of varying sizes, ranging from 125 million to 175 billion parameters, using a dataset refined from the CommonCrawl, which contained nearly a trillion words of general knowledge mixed with other high-quality datasets to increase diversity. By looking at the GPT-3 dataset distribution in Table 2.2, it can be seen that it was not sampled proportionally to its size, but based on quality, with high-quality datasets sampled less frequently than lower-quality ones during training. The models were evaluated on more than 20 NLP datasets in few-shots, one-shot, and zero-shot settings, depending on how many task examples were given to the models, with the latter requiring no examples for the given task.

Training dataset	Quantity (tokens)	Weight in training mix
Filtered CommonCrawl	40B	60%
WebText2	19B	22%
Books1	19B	8%
Books2	55B	8%
Wikipedia	3B	3%

Table 2.2: Composition of training dataset for GPT-3.

The results of all the test done by GPT-3 benchmarks in the zero-shot, one-shot, and few-shot settings remark the potential of the in-context learning model, in some cases nearly matching the performance of state-of-the-art fine-tuned systems and in a few trials also reaching new state of the art results. The main innovation

introduced by GPT-3 was the confirmation that, with a n-shot perspective, the same (or close) results as fine-tuned models could be achieved.

Model	Parameters	Dataset size estimation	Max sequence length
GPT-1	117M	4.5GB	512
GPT-2	1.5B	40GB	1024
GPT-3	175B	570GB	2048

Table 2.3: Comparison of characteristics of GPT-1, GPT-2, and GPT-3. For GPT-3, the size refers to the filtered version of the CommonCrawl dataset.

2.2.4 ChatGPT

ChatGPT is a finetuned version of GPT-3.5, which was launched in November 2022 as a public application for research purposes and is an upgraded version of GPT-3. Unlike all previous GPT models, ChatGPT is designed for conversational AI. The model is trained to answer follow-up questions, generate prompted stories and jokes, restyle input texts, and more. During the training phase, OpenAI used both a supervised learning approach, in which the model was fed by a conversation in which a trainer played the role of both model and human, and an unsupervised learning approach, done by integrating Reinforcement Learning from Human Feedback (RLHF) [18]. This approach was previously used for its sibling model InstructGPT [19]. With RLHF, the model is prompted and different outputs are sampled, then a human ranks the responses from best to worst, and these evaluations are used to train a reward model to adjust and finetune the model's responses to be more similar to human responses. Moreover, OpenAI gives the user the option to provide feedback by upvoting or downvoting the model responses, which allows for on-the-fly finetuning of the model.

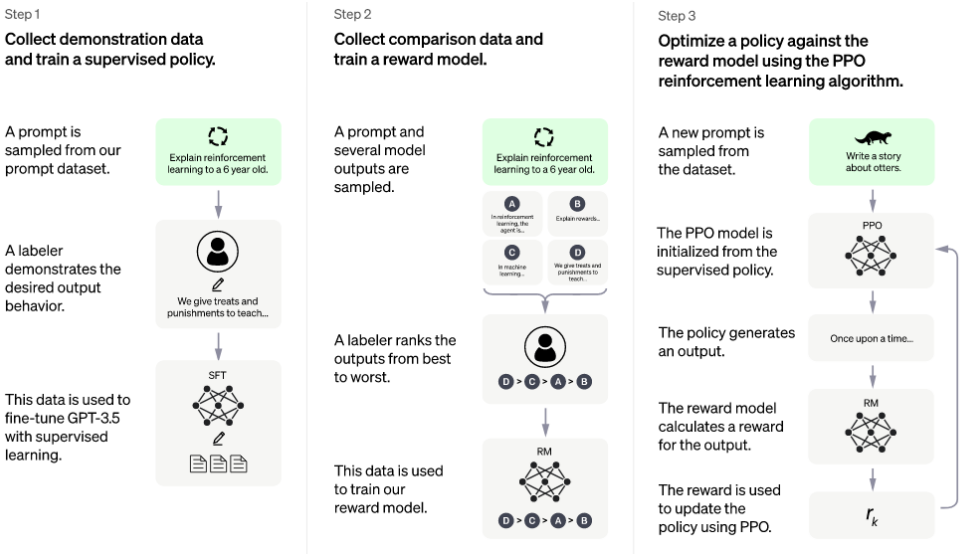

Figure 2.4: Description of ChatGPT training with supervised learning, unsupervised learning and model rewarding.

ChatGPT has gained a lot of popularity in the months following its release, capturing the attention of the entire global community and giving rise to both hopes for its hypothetical applications and doubts and concerns regarding its malicious uses and the data policy.

On March 2023, OpenAI released a ChatGPT API (Application Programming Interface) to allow the integration of the technology into other application, enlarging the domain of application for the model.

2.2.5 PaLM

PaLM (Pathways Language Model) [20] is a language model developed by Google and specifically designed for few-shot classification, following in the footsteps of other competitors in the field. The model is composed of 540 billion parameters and was trained using a new machine learning approach called Pathways. Unlike other models that are trained using a single TPU or pipeline parallelism to scale across different GPU clusters, Google's Pathways enables training a single model across thousands or tens of thousands of accelerators, achieving a very high efficiency of 46.2% in model FLOPs (observed throughput relative to theoretical max throughput) and 57.8% in hardware FLOPs utilization. The model architecture is based on a decoder-only transformer setup, using SwiGLU activation functions and departing from the standard formulation of the transformer: with previous implementation, the Masked Language Prediction (MLP) moved from a sequential

setting (Eq. 2.5)

$$y = x + MLP(LayerNorm(x + Attention(LayerNorm(x)) \tag{2.5}$$

to a parallelized one, resulting in faster training speed (Eq. 2.6).

$$y = x + MLP(LayerNorm(x)) + Attention(LayerNorm(x) \tag{2.6}$$

The training dataset, composed by 780 billion tokens, was obtained by a weighted mix of different sources in different languages (around 20% of non-english data).

Data source	**Proportion of data**
Social media conversations (multilingual)	50%
Filtered webpages (multilingual)	27%
Books (English)	13%
GitHub (code)	5%
Wikipedia (multilingual)	4%
News (English)	1%

Table 2.4: PaLM training dataset composition [20].

PaLM 540B outperforms prior state-of-the-art models on 24 of the 29 tasks in the 1-shot setting and 28 of the 29 tasks in the few-shot setting. Interestingly, PaLM 540B outperforms prior state-of-the-art models by more than 10 points in the few-shot setting on some of the Reading Comprehension and NLI tasks. In the Massive Multitask Language Understanding (MMLU) benchmark, it reaches an average score of 69.3%. In the field of Commonsense Reasoning, prior state-of-the-art models use a combination of task-specific finetuning, domain-specific architectures, and task-specific verifiers to achieve strong results. However, PaLM's few-shot results match or exceed the finetuned state-of-the-art across several different arithmetic and commonsense reasoning tasks, thanks to a combination of scale and chain-of-thought prompting. In the chain-of-thought method, the model is prompted to generate a natural language logical inference chain before making its prediction.

Task	0-shot		1-shot		Few-shot	
	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B	Prior SOTA	PaLM 540B
TriviaQA	71.3^{a}	76.9	75.8^{a}	81.4	$75.8^{a}_{(1)}$	$81.4_{(1)}$
NaturalQuestions	24.7^{a}	21.2	26.3^{a}	29.3	$32.5^{a}_{(1)}$	$39.6_{(64)}$
WebQuestions	19.0^{a}	10.6	25.3^{b}	22.6	$41.1^{b}_{(64)}$	$43.5_{(64)}$

Lambada	77.7^{f}	77.9	80.9^{a}	81.8	$87.2^{c}_{(15)}$	$89.7_{(8)}$
HellaSwag	80.8^{f}	83.4	80.2^{c}	83.6	$82.4^{c}_{(20)}$	$83.8_{(5)}$
StoryCloze	83.2^{b}	84.6	84.7^{b}	86.1	$87.7^{b}_{(70)}$	$89.0_{(5)}$
Winograd	88.3^{b}	90.1	89.7^{b}	87.5	$88.6^{a}_{(2)}$	$89.4_{(5)}$
Winogrande	74.9^{f}	81.1	73.7^{c}	83.7	$79.2^{a}_{(16)}$	$85.1_{(5)}$
Drop(F1)	57.3^{a}	69.4	57.8^{a}	70.8	$58.6^{a}_{(2)}$	$70.8_{(1)}$
CoQA(F1)	81.5^{b}	77.6	84.0^{b}	79.9	$85.0^{b}_{(5)}$	$81.5_{(5)}$
QuAC(F1)	41.5^{b}	45.2	43.4^{b}	47.7	$44.3^{b}_{(5)}$	$47.7_{(1)}$
SQuADv2(F1)	71.1^{a}	80.8	71.8^{a}	82.9	$71.8^{a}_{(10)}$	$83.3_{(5)}$
SQuADv2(EM)	64.7^{a}	75.5	66.5^{a}	78.7	$67.0^{a}_{(10)}$	$79.6_{(5)}$
RACE-m	64.0^{a}	68.1	65.6^{a}	69.3	$66.9^{a}_{(8)}$	$72.1_{(8)}$
RACE-h	47.9^{c}	49.1	48.7^{a}	52.1	$49.3^{a}_{(2)}$	$54.6_{(5)}$
PIQa	82.0^{c}	82.3	81.4^{a}	83.9	$83.2^{c}_{(5)}$	$85.2_{(5)}$
ARC-e	76.4^{e}	76.6	76.6^{a}	85.0	$80.9^{e}_{(10)}$	$88.4_{(5)}$
ARC-c	51.4^{b}	53.0	53.2^{b}	60.1	$52.0^{a}_{(3)}$	$65.9_{(5)}$
OpenbookQa	57.6^{b}	53.4	55.8^{b}	53.6	$65.4^{b}_{(100)}$	$68.0_{(32)}$
BoolQ	83.7^{f}	88.0	82.8^{a}	88.7	$84.8^{c}_{(32)}$	$89.1_{(8)}$
Copa	91.0^{b}	93.0	92.0^{a}	91.0	$93.0^{a}_{(16)}$	$95.0_{(5)}$
RTE	73.3^{e}	72.9	71.5^{a}	78.7	$76.8_{(5)}$	$81.2_{(5)}$
WiC	50.3^{a}	59.1	52.7^{a}	63.2	$58.5^{c}_{(32)}$	$64.6_{(5)}$
Multirc(F1a)	73.7^{a}	83.5	74.7^{a}	84.9	$77.5^{a}_{(4)}$	$86.3_{(5)}$
WSC	85.3^{a}	89.1	83.9^{a}	86.3	$85.6^{a}_{(2)}$	$89.5_{(5)}$
ReCoRD	90.3^{a}	92.9	90.3^{a}	92.8	$90.6_{(2)}$	$92.9_{(2)}$
CB	48.2^{a}	51.8	73.2^{a}	83.9	$84.8^{a}_{(8)}$	$89.3_{(5)}$
ANLIR1	39.2^{a}	48.4	42.4^{a}	52.6	$44.3^{a}_{(2)}$	$56.9_{(5)}$
ANLIR2	39.9^{e}	44.2	40.0^{a}	48.7	$41.2^{a}_{(10)}$	$56.1_{(5)}$
ANLIR3	41.3^{a}	45.7	40.8^{a}	52.3	$44.7^{a}_{(4)}$	$51.2_{(5)}$

Table 2.5: Results obtained by the PaLM 540B model across 29 NLP benchmarks. For the few-shot results, the number of shots for each task are mentioned in parenthesis. The splits for each task are the same ones used in Du et al. (2021) and Brown et al. (2020). Superscripts denote results from past work: *a*) GLaM 62B/64E (Du et al., 2021), *b*) GPT-3 175B , *c*) Megatron-Turing NLG 530B (Smith et al., 2022), *d*) Gopher (Rae et al., 2021), *e*) LaMDA (Thoppilan et al., 2022) (results reported from Wei et al. (2022a), *f*) Chinchilla (Ho mann et al., 2022)). Figure from [20].

2.2.6 LLaMA

Meta AI, formerly known as Facebook AI, published its open-source model called LLaMA in 2023 [21]. With LLaMA, Meta aimed to demonstrate how a model trained solely with public datasets could achieve, and sometimes exceed, the potential of other larger pretrained models. The pretraining followed the same technique as GPT-3, and the dataset (Table 2.6) was created by a weighted mix of public-only datasets with the same proportionality measure as GPT-3. The model architecture is essentially the same as the Transformer, with the same prenormalization as GPT and a different activation function: ReLU (Eq. 2.7) activation function

$$ReLU(x) = max(0, x) \tag{2.7}$$

was replaced by the SwiGLU (Eq. 2.8)

$$SwiGLU(x) = x * sigmoid(\beta * x) + (1 - sigmoid(\beta * x)) * (Wx + b) \tag{2.8}$$

SwiGLU uses a gating mechanism, which enables it to activate neurons selectively based on the input it receives, reducing overfitting and improving generalization.

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

Table 2.6: Dataset training mix for LLaMA.

Four different model, from 6 to 65 billion parameters, were trained and tested over various benchmark. As it shown on Table 2.7, despite being smaller than competitors, LLaMA has managed to achieve excellent results, outperforming GPT-3 and even reaching state-of-the-art performance, on many tasks such as Question Answering (TriviaQA ad NaturalQuestions), Code Generation (HumanEval) and CommonSense Reasoning.

Model	Parameters	BoolQ	PIQA	SIQA	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA
GPT-3	175B	60.5	81.0	-	78.9	70.2	68.8	51.4	57.6
PaLM	540B	**88.0**	82.3	-	83.4	**81.1**	76.6	53.0	53.4
LLaMA	7B	76.5	79.8	48.9	76.1	70.1	72.8	47.6	57.2
	13B	78.1	80.1	50.4	79.2	73.0	74.8	52.7	56.4
	33B	83.1	82.3	50.4	82.8	76.0	**80.0**	**57.8**	58.6
	65B	85.3	**82.8**	**52.3**	**84.2**	77.0	78.9	56.0	**60.2**

Table 2.7: LLaMA performances in CommonSense Reasoning.

2.2.7 GPT-4

In March 2023, OpenAI published a paper introducing their new iteration of the GPT model, called GPT-4 [22]. This model represents a significant change from previous iterations, as it is a multimodal model capable of processing both textual and image information. This opens up a new subset of vision tasks in NLP, such as predicting consequences of actions in pictures (for example, given an image of a balloon, asking the model to predict the consequences of cutting the balloon's string) or explaining humor in a specific image. With context windows of 8,192 and 32,768 tokens, OpenAI tested GPT-4's capabilities both with and without the vision module across multiple public benchmarks. The results obtained outperformed most of the previous models' performance: in classical Language Modeling tasks, As shown in Table 2.8 GPT-4 achieved state-of-the-art performance in most tasks with a gain of more than 10% compared to the previous SOTA methodology.

Dataset	GPT-4 evualuated few-shots	GPT-3.5 evualuated few-shots	LM SOTA evualuated few-shots	SOTA Best external (with specific finetuning)
MMLU multiple choice question in 57 subjects (professional and academic)	**86.4%** 5-shots	70.0% 5-shots	70.7% 5-shots U-PaLM	75.2% 5-shots Plan-PaLM
HellaSwag Commonsense reasoning around everyday events	**95.3%** 10-shots	85.5% 10-shots	84.2% LLaMA (validation set)	85.6% ALUM
AI2 Reasoning Challenge Grade-school multiple choice science questions. Challenge-set.	**96.3%** 25-shot	85.2% 25-shot	85.2% 8-shot PaLM	86.5% ST-MOE
WinoGrande Commonsense reasoning around pronoun resolution	**87.5%** 5-shot	81.6% 5-shot	85.1% 5-shot PaLM	85.1% 5-shot PaLM
HumanEval (Python coding tasks)	**67.0%** 0-shot	48.1% 0-shot	26.2% 0-shot PaLM	65.8% CodeT + GPT-3.5
DROP (F1 Score) Reading comprehension and arithmetic.	80.9 3-shot	64.1 3-shot	70.8 1-shot PaLM	**88.4** QDGAT
GSM-8K Grade-school mathematics questions	**92.0%** 5-shot	57.1% 5-shot	58.8% 8-shot Minerva	87.3% Chinchilla

Table 2.8: GPT-4 perfomances on academic benchmark.

GPT-4 was also tested on different public exam simulations. The peculiarity of this task lies in the fact that the model was never specifically trained for a particular exam, but a few-shot methodology was used instead. The exams, which comprised both multiple-choice and open questions, required specific prompts (including images) to be designed for the model and were evaluated using public methodologies. The results were not influenced by reinforcement learning from

human feedback (Table 2.9), showing the potential of multi-task training. Most of GPT-4's performance on these tasks was successful, with a remarkable top 10% score achieved on the Uniform Bar Examination.

Exam	**Score**	**Score using RLHF**
AP Art History (MCQ)	72.5%	66.2%
AP Biology (MCQ)	98.3%	96.7%
AP Calculus BC (MCQ)	66.7%	57.8%
AP Chemistry (MCQ)	58.3%	71.7%
AP English Language and Composition (MCQ)	55.6%	51.1%
AP English Literature and Composition (MCQ)	63.6%	69.1%
AP Environmental Science (MCQ)	72.5%	67.5%
AP Macroeconomics (MCQ)	83.3%	76.7%
AP Microeconomics (MCQ)	90.0%	76.7%
AP Physics 2 (MCQ)	62.2%	71.1%
AP Psychology (MCQ)	98.0%	96.0%
AP Statistics (MCQ)	60.0%	62.5%
AP US Government (MCQ)	85.5%	83.6%
AP US History (MCQ)	89.1%	87.3%
AP World History (MCQ)	94.5%	98.2%
MKSAP Questions (MCQ)	77.9%	74.7%

Table 2.9: GPT-4 exam scores comparison between plain model and RLFH model.

Even if architectural and training informations were not published due to competitive reasons, GPT-4 performance results place the model among the most high-performing models of all time.

Chapter 3

Methodology

3.1 Goal

The methodology proposed in this thesis work aims to propose a protocol applicable to all the project that integrate generative models. The use of this framework would fit into different use cases belonging to completely different sectors. Possible applications could be as follows:

- Once the documentation regarding different drugs is provided to the framework by a medicine company, a doctor could consult the framework to ask a summarization of the side effects for a specific medicine.

- Given the use and maintenance manuals of different agricultural tools, an operator could consult the model to receive information about the maintenance intervals of a specific agricultural machine.

- Provided regulations and documentation related to a business sector, an HR manager could consult the framework to receive summaries regarding specific protocols.

By leveraging the capabilities of generative models and natural language processing, potential professionals or companies could receive adequate support in their work, enhancing performance efficiency through the precision of models and reducing latency in operations. Furthermore, for private data applications, any new documentation could be automatically incorporated into the respective models, eliminating the need for frequent appointments with representatives or the downloading and retrieval of new modules.

3.2 Methodology

The structure of the methodology, as shown in Fig. 3.1, can be divided in fixed phases each one described in the following section.

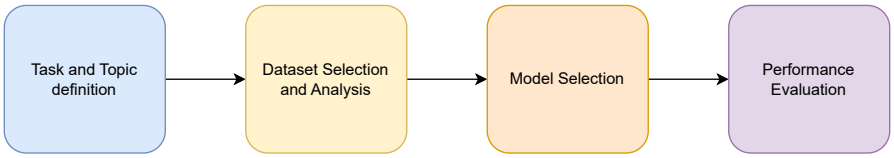

Figure 3.1: Methodology pipeline.

3.2.1 Task and Topic Definition

The customer presents the topics on which the generative model should be prepared, highlighting the main areas of interest and all the related subtopics that may exist. Once the topics are described, which are useful for labeling the documentation as specifically as possible, the tasks on which the model should be trained will be defined, along with their corresponding expected results. This approach creates a precise scope of action within which the model should operate. The more information provided in this phase, the better the model can be shaped to meet the customer's expectations, thus avoiding outputs that do not align with the actual purpose of the model.

For example, if the client desires a model capable of answering specific questions related to various topics, the required task would be a Question Answering (QA) task. Consequently, the chosen model must be specialized in that particular task, thereby narrowing down the scope of selection to models specifically designed for QA.

3.2.2 Dataset Selection and Analysis

Once the topics and required tasks have been established, the base documentation must be defined. In this phase, for each topic identified in the previous stage, corresponding document sets are provided containing all the relevant information necessary for the model to fulfill its tasks. The complexity of this phase lies in the diverse nature of the datasets involved:

- Different types of data need to be processed differently using various tools. For example, if the document set is heterogeneous and comprises files in formats such as `docx`, `pdf`, or sets of images, different tools are required to process the inputs.

- Any noisy data should be corrected, and extraneous information should be eliminated to avoid introducing additional sources of noise.

- The sensitivity of the data determines the structures utilized in its processing. The presence of sensitive data, such as user profiles, personal information, or biometric data, necessitates that the models employed prioritize privacy and do not rely on third-party users capable of storing and utilizing this information.

- The more detailed the documentation pertaining to a specific topic, the more precise the model's response will be, regardless of the task type.

- Prepare the client's provided documents for model ingestion. This may involve cleaning and preprocessing the text, converting it into a suitable format, and creating appropriate training, validation, and test data sets.

3.2.3 Model Selection

After cleaning, processing and analyzing the dataset, it is crucial to carefully select the respective generative models to be used for performing the required tasks. The choice of models and their types is influenced by several factors:

- The nature of the data, such as its sensitivity, primarily affects the choice of model. If the data is not publicly available but contains sensitive information, a company cannot rely on third-party models or API calls to external sources. Therefore, it is essential to find a locally deployable model to ensure data confidentiality or employ data masking techniques.

- The availability of resources on the client's side influences the selection of a more powerful and resource-intensive model compared to others. The goal is to find the optimal trade-off between model capability and resource availability. If the model's responses are subject to latency requirements, it is necessary to rely on faster models, which may require additional resources.

In order to select the proper model for our application, the tokenizer and the weights can be requested directly to vendors, such as for LLaMA, or the choice can move to open-source models publicly available. Regarding this category, open-source LLMs can be found in public repositories and communities such as GitHub [23] or Hugging Face [24]. In these spaces, users are provided with model details, implementations, licenses, playgrounds and API in order to perform inference. Moreover, a user could finetune a base model for a particular task and then release it on the same platform. On HuggingFace specific fine-tuned models are available

for specific task usage, such as MosaicML's MPT-7B-Storywriter [25] and MPT-30B-Chat [26], which are fine-tuned version of base MPT 7B and 30B models from MosaicML [27], namely for storytelling and instruct purposes.

3.2.4 Performance Evaluation

The evaluations of performance, being generative tasks, can be done qualitatively by manually assessing the effectiveness of the model for a given task. This is done by comparing the model's generated responses with the expected ones, such as those that an expert human could produce for the same type of task. The key aspect of the evaluation consists on a Human-in-the-Loop approach, similar to the one applied with ChatGPT: by analyzing model's output responses, areas for improvement can be identified and addressed. Depending on the results, prompt engineering could be used in order to provide high quality context prompts to the model, follow a n-shot approach by giving some examples to the model or simply adjusting the dataset with accurately selected preprocessing or data augmentation techniques.

3.3 First Steps to Apply the Methodology in a Real Use Case

Once the four phases of the proposed methodology have been outlined, the Model Selection and Performance Analysis are strictly related to the first two steps, namely Task and Topic Definition and Dataset Selection and Analysis. The more information exchanged with the client regarding the project's intent, nature of the data and the accessibility of the data itself, the better defined the nature of the data will be, making it easier to choose the model and put it into production faster, ultimately leading to improved performance.

One possible solution to define the correct data perimeter could consist on using a fixed template in which the client provides specific dimensions for each task on which the trained model is desired. An example of feature perimeter could include:

- The desired output data type from the model, such as textual data, images, websites, or their combinations.
- The definition of the task category, such as information retrieval or text classification.
- The definition of the topic related to the data-task pair to facilitate data indexing, for instance.

- The data source and the method of data acquisition, such as using specific APIs or parsing from specific documents (single or multiple).
- Eventual data update timelines, especially for dynamic data.
- Data specificity, as certain data may be specific to a requester's role, ensuring confidentiality, or to a certain state, such as different internal policies for multinational companies.
- The need for a conversational agent that tracks the context of previous conversations or simply provides instant question-answering capabilities.

Chapter 4

Use Cases

This chapter will discuss 10 different scenarios of application of the methodology, listed in Table 4.1, each one representing a specific generative task. These tasks will be presented in order of complexity. The evaluation will start with tasks characterized by simple but structural prompts, in order to understand how generative models work, and to have the basis for being able to structure more complex generative tasks such as the Natural Language Command Translation at the end of the chapter. This last task been integrated into the orchestrator for Boston Dynamics SPOT robot, thus allowing to control it simply by using natural language prompt.

Task	Example	Model
Text Summarization	Road regulations	GPT-3.5-turbo-16k
Question Answering	Pharmaceutical instructions	GPT-3.5-turbo-16k
StoryTelling	Relative Theorem: Child-Friendly Translation	GPT-3.5-turbo
Named Entity Recognition	Road regulation fines and limits	GPT-3.5-turbo-16k
Code Generation	PDF text extraction	GPT-3.5-turbo
Code Understanding	Habitat-Lab	GPT-3.5-turbo-16k
Code Review	Rust Code Optimization	GPT-3.5-turbo
From CSV to NL	Customers File Analysis	GPT-3.5-turbo
Smart Search + Question Answering	Pharmaceutical Pamphlets	LLamaIndex + ChatGPT
Natural Language to Command Translation	Controlling a robot agent with Natural Language prompt	GPT-3.5-turbo

Table 4.1: List of covered generative tasks.

All the tasks were evaluated using English prompts over different language

documents. The model used in all tasks evaluation was the GPT-3.5-turbo and its 16k token length version for long documents and text.

Due to the generative nature of all the tasks, in order to have a more objective measure of the performance each evaluation consists on the analysis of 5 attempts with the same prompt and context.

The code used for the evaluation and the set of outputs obtained for each task can be found in the repository at the following link:
https://github.com/stestrippoli/NLP-tasks

4.1 Task: Text Summarization on Road Regulation Documents

4.1.1 Task and Topic Definition

With text summarization, the goal is to obtain an outline of a specific input text. The textual output should be concise and should contain all the useful information from the original input file.

4.1.2 Dataset Selection and Analysis

For this task, the summarization was made using a road regulation document [28] regarding speed limits. The pdf file used for this task was composed by seven pages, each page containing a header and footer text which give no information to the task completion. After extracting the text, the text was pre-processed by removing unwanted multiple whitespaces, header and footer and resolving character decoding issues such as accented words in order to obtain the least number of tokens.

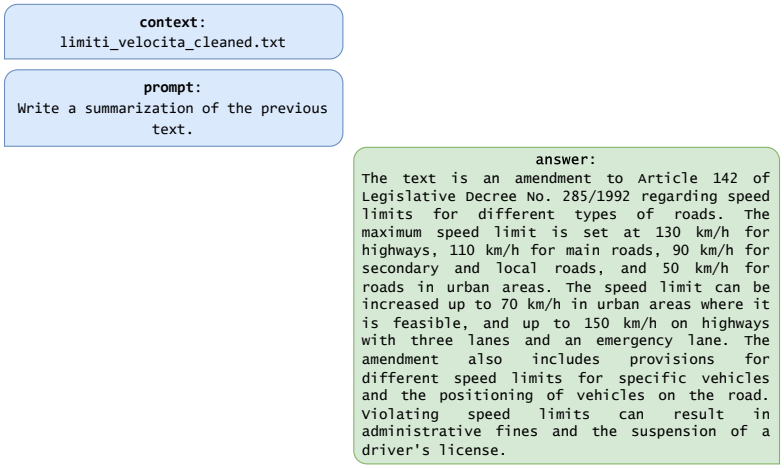

Figure 4.1: Example of the Summarization task.

4.1.3 Model Selection

The model selected was GPT-3.5-turbo-16k.

4.1.4 Performance Evaluation

From the five generated results it can be observed that all of them provide a good summarization of the text, highlighting key points related to speed limits and vehicle positioning regulations. An example is provided in Fig. 4.1.

In terms of how well the answers fulfill the request to write a summarization of the text, all of the answers provide a concise and accurate overview of the main points. However, some answers provide more details and specifics compared to others. Answer 3, for example, provides a more detailed breakdown of the speed limits on different types of roads and conditions for increasing the speed limit. Answer 4 also includes additional information about minimum and maximum speed limits set by road owners. Overall, all answers provide a satisfactory summarization of the text.

4.2 Task: Question Answering on Pharmaceutical Instructions

4.2.1 Task and Topic Definition

The question answering task can be divided into two different sub-tasks: the first one involves generating questions about a given input text. Once generated, the questions will become the input for the question answering task.

4.2.2 Dataset Selection and Analysis

As input text, the package leaflet of a drug called Zetamicin [29] has been used in PDF format. After extracting the text and performing preprocessing actions, which consisted of removing unnecessary lines to reduce the number of tokens, the text was provided to the model as context and the model was then asked to generate three questions based on the text (Fig. 4.2).

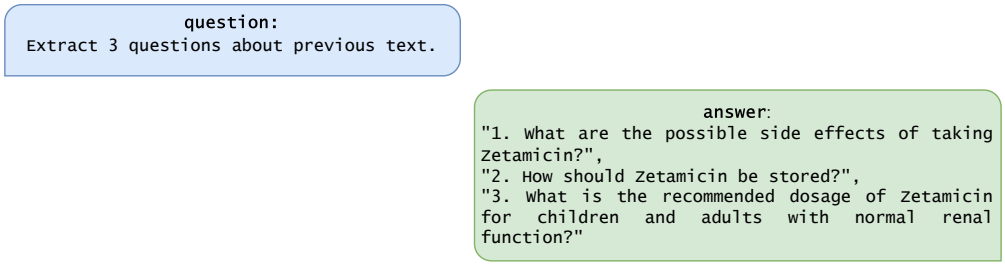

Figure 4.2: Question generation.

Once generated, the same question were prompted to the model in order to obtain the relative answers (Fig. 4.3). With this approach, both the generation of the answers and the relative answering phases are tested.

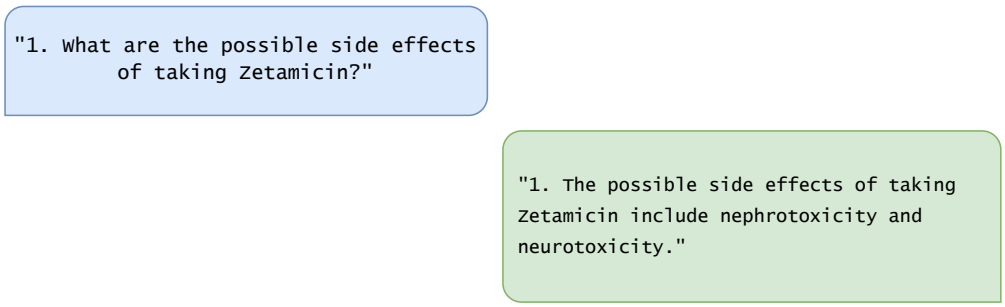

Figure 4.3: Question answering.

4.2.3 Model Selection

The model selected was GPT-3.5-turbo-16k. Since the document used exceeded the capabilities of the base turbo model, the 16k token length version was selected.

4.2.4 Performance Evaluation

The questions generated cover various aspects of ZETAMICIN, including its possible side effects, storage guidelines, and recommended dosage for children and adolescents. Overall, they offer relevant information about the medication. However, some questions lack specificity and could be enhanced with more detailed explanations or clarifications. The fifth question stands out for its comprehensive response, addressing the specific inquiry about ZETAMICIN's active ingredient and demonstrating a strong understanding of the request. Regarding the answers of the previous generated questions, all the answers are correct, with different degrees of specificity on the responses.

4.3 Task: Storytelling on Theorem of Relativity

4.3.1 Task and Topic Definition

With the Storytelling task the model has to generate a particular story given a specific context and the style in which the text must be written. In this use case the model was asked to generate a fable that could explain the complicated Relativity Theory to a child, using fantastic elements. The story should be written in a children comprehensible language and without complicated term and should explain all the concept maintaining the narrative thread.

4.3.2 Dataset Selection and Analysis

As shown in Fig. 4.4, the prompt included the details about the elements that should be used in the story, such as ogres, witches and potions. The full output can be consulted in the Appendix A.1.1.

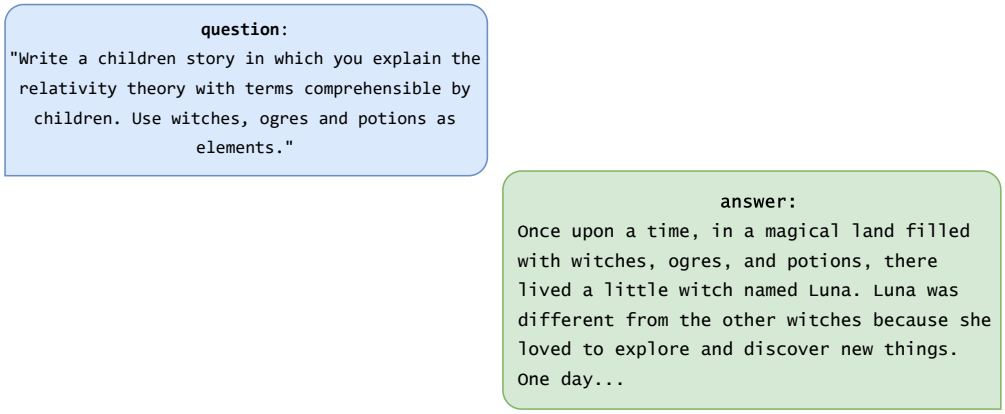

Figure 4.4: Storytelling task. The complete output is in Appendix A.1.1

4.3.3 Model Selection

The model selected was GPT-3.5-turbo.

4.3.4 Performance Evaluation

Overall, the five answers meet the request of creating a children's story that explains the theory of relativity using witches, ogres, and potions as imaginative elements. Each story features a young witch named Willow, who is inquisitive and eager to learn. The concept of time dilation and its relation to speed is explained in all stories, using magical potions to transport characters to different dimensions or

alter their perception of time. The stories also showcase various magical creatures, such as ogres, to demonstrate the effects of relativity on their experiences of time.

Despite the similarities, there are some differences among the stories. The first story centers more on the witches' exploration of relativity, while the others involve ogres and their understanding of the concept. In the second and fourth stories, an ogre named Oliver plays a specific role, while other stories feature different characters. The third story introduces additional characters, Olivia and Ollie, who learn alongside Willow. The fourth story incorporates an enchanted forest and a relativity elixir to access the realm of relativity.

In terms of fulfilling the request, all the answers effectively use creative storytelling and magical elements to explain the theory of relativity in a way suitable for children. However, the level of detail and depth of explanation varies between the answers. Some stories focus more on the characters' experiences and the wonder of magic, while others provide a more explicit explanation of relativity. The suitability of each answer may depend on the target audience's age and understanding of the scientific concept.

4.4 Task: Named Entity Recognition on Road Regulation Fines and Limits

4.4.1 Task and Topic Definition

With Named Entity Recognition (NER) the objective is to identify the nominal entity inside a textual corpus. Given a specific input text, the goal is to correctly identify and label these entities in order to analyze, understand, and utilize them in various NLP applications such as information extraction, translation, sentiment analysis, and many others.

4.4.2 Dataset Selection and Analysis

The corpus used in this task is the same road regulation document used in the summarization task and the model was asked to identify the entities of fines, speed limits and road typology inside the text.

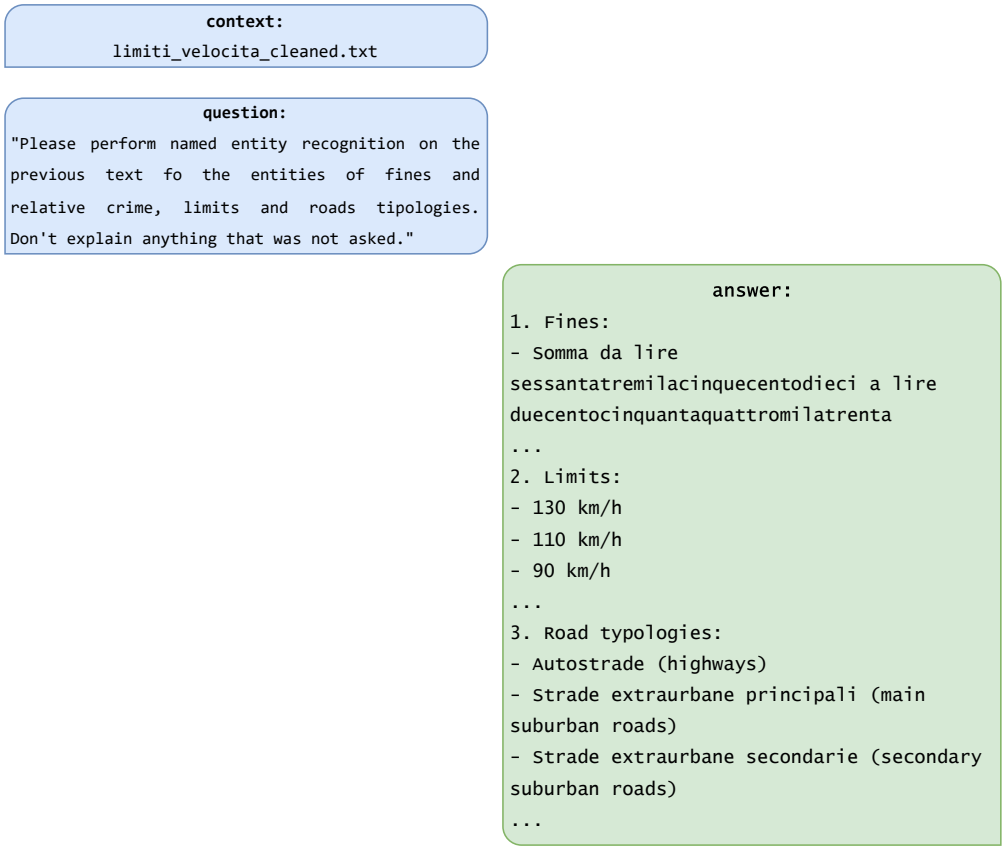

Figure 4.5: Named Entity Recogniton.

4.4.3 Model Selection

The model selected was GPT-3.5-turbo-16k. Also in this case, the 16k token length model was used since the document exceeded the capabilities of the base turbo model.

4.4.4 Performance Evaluation

The answers, which Fig. 4.5 is an example, provide a good representation of the named entities related to fines and relative crimes, limits, and road typologies. They generally include common entities and some specific variations based on the provided text. However, there are slight differences in the specific entities mentioned, which may be due to variations in the way named entity recognition is performed or the specific entities recognized by different NER models. For example, some answers include additional entities such as specific speed limit values like “130 km/h”, “110 km/h”, “90 km/h” or some fines such as “sospensione della patente di

guida".

4.5 Task: Code Generation on PDF Text Extraction

4.5.1 Task and Topic Definition

With Code Generation, want to generate functional code by specifying the programming language and any parameters to the model. The produced code should be working and exportable.

4.5.2 Dataset Selection and Analysis

In this case, the model was asked to generate a Python function called 'pdf_to_text' capable of extracting text from a PDF. Once obtained the output and created a .py file containing the generated code, the same function was tested by using it on a simple document related to a call for applications taken from the Polytechnic's website [30] in PDF format. An example is provided in Fig. 4.6.

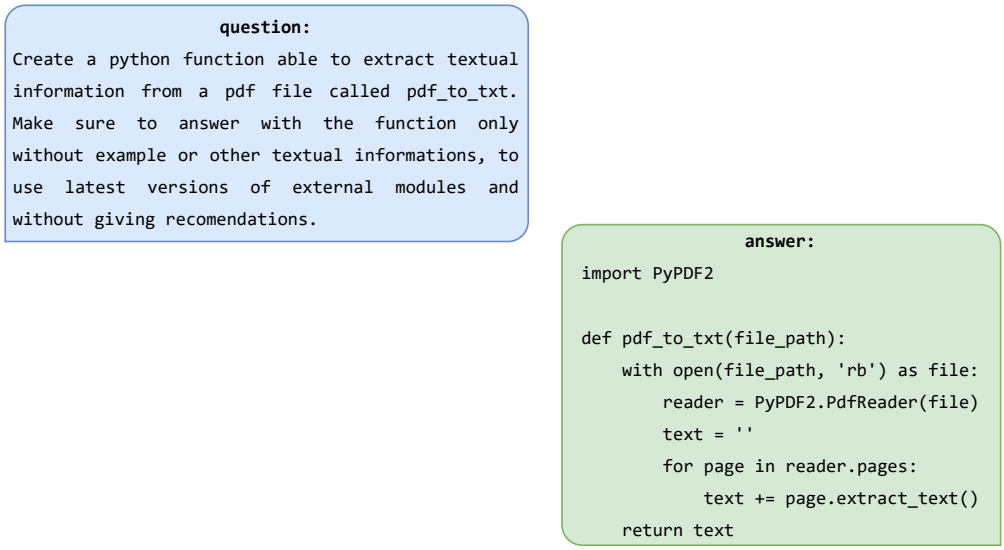

Figure 4.6: Code Generation.

4.5.3 Model Selection

The model selected was GPT-3.5-turbo.

4.5.4 Performance Evaluation

In summary, all five of the Python function generated by the model successfully extracts textual information from a PDF file. They share similarities in their use of the PyPDF2 library, opening the PDF file in binary mode ("rb") for reading, iterating through the PDF pages, concatenating the extracted text, and returning the output as the extracted text.

However, there are differences among the solutions. Some answers use the PyPDF2.PdfReader class and its extract_text() method, while others utilize the PyPDF2.PdfFileReader class along with getPage() and extract_text() methods.

Despite these differences, all solutions offer valid and effective ways to accomplish the task of text extraction from a PDF file. Some solutions adopt more modern approaches, directly accessing reader.pages[page_num], while others opt for simplicity without compromising functionality, using len(reader.pages).

4.6 Task: Code Understanding on HabitatLab Repository

4.6.1 Task and Topic Definition

The Code Understanding tasks consist on asking the model to explain a specific piece of code, such as functions or classes implementation, giving details about the implementation and how the class works. An example is reported in Fig. 4.7.

4.6.2 Dataset Selection and Analysis

The provided input contained the implementation of the classes Env, contained in the env.py file from the Facebook Research Habitat-Lab [31] repository. Then the model was asked to explain the content of the file by describing classes and methods and focusing on the implementation of the first 5 function in order to test the explainability power of it.

context:
env.py

question:
"Explain the python code given before. Describe all the classes and relative methods. Explain in detail the first 3 function of the class Env."

answer:
The given code defines two classes: `Env` and `RLEnv`.

1. `Env` class:
   - This class represents the fundamental environment class for the Habitat framework.
   ...

2. `RLEnv` class:
   - This class is a wrapper over the `Env` class and subclasses `gym.Env`.
   ...

The first three methods of the `Env` class are:

- `__init__(self, config: "DictConfig", dataset: Optional[Dataset[Episode]] = None) -> None`: ...

Figure 4.7: Code Understanding. The complete response is in Appendix A.1.2

4.6.3 Model Selection

The model selected was GPT-3.5-turbo-16k since the code length exceeded the 4096 token length limit.

4.6.4 Performance Evaluation

The description of the Env class is consistent across all answers, emphasizing its significance as the fundamental environment class in the Habitat framework. They highlight its components, including the dataset, simulator, and task, and underscore its essential methods for resetting the environment, taking actions, and obtaining observations.

4.7 Task: Code Review on Rust Code Optimization

4.7.1 Task and Topic Definition

The task of Code Review involves reading existing code and optimizing it in terms of time complexity, readability, coherence, and error correction. Given a specific code, the model should be able to reproduce the same optimized code.

4.7.2 Dataset Selection and Analysis

As input to the model, a Rust code related to the implementation of a calendar manager is provided, and then the model is asked to optimize it (Fig. 4.8).

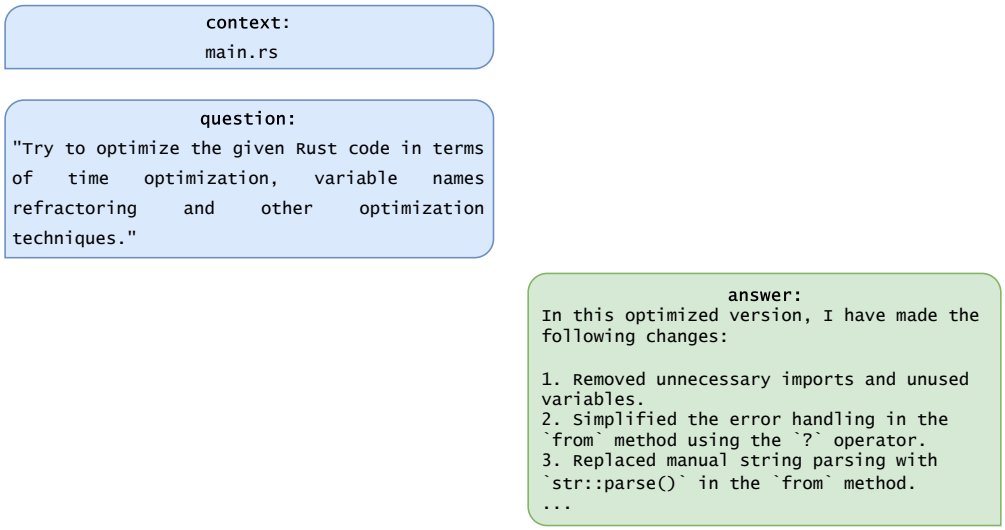

Figure 4.8: Code Review. The full response is in Appendix A.1.3

4.7.3 Model Selection

The model selected was GPT-3.5-turbo-16k.

4.7.4 Performance Evaluation

All five answers focus on optimizing the given Rust code, with an emphasis on time optimization, variable name refactoring, and other optimization techniques. The similarities among the answers include removing unnecessary imports, variables, and code blocks, as well as optimizing the input file reading process using BufReader

and read_line functions. The Date struct is refactored in all answers to implement various traits for enhanced functionality. The optimization, however, is not perfect: some of the output code generated compiling error, such as missing implementations of traits. Since the model is updated with Rust documentation up to 2021, some of the error obtained could be caused to updated documentations of the language.

4.8 Task: From CSV to Natural Language on Customers File Analysis

4.8.1 Task and Topic Definition

In this task, the goal is to extract quantitative and qualitative analyses from tabular data, extracting intrinsic information and describing the characteristics of the entire table. The model is expected to provide an analysis based on a given input file.

4.8.2 Dataset Selection and Analysis

In our case, described in Fig. 4.9, the first 20 records from a public sample file [32] were used. The file was previously converted into textual data and passed to the model. Once processed, the model is queried to provide an analysis of the data and select information based on intrinsic data.

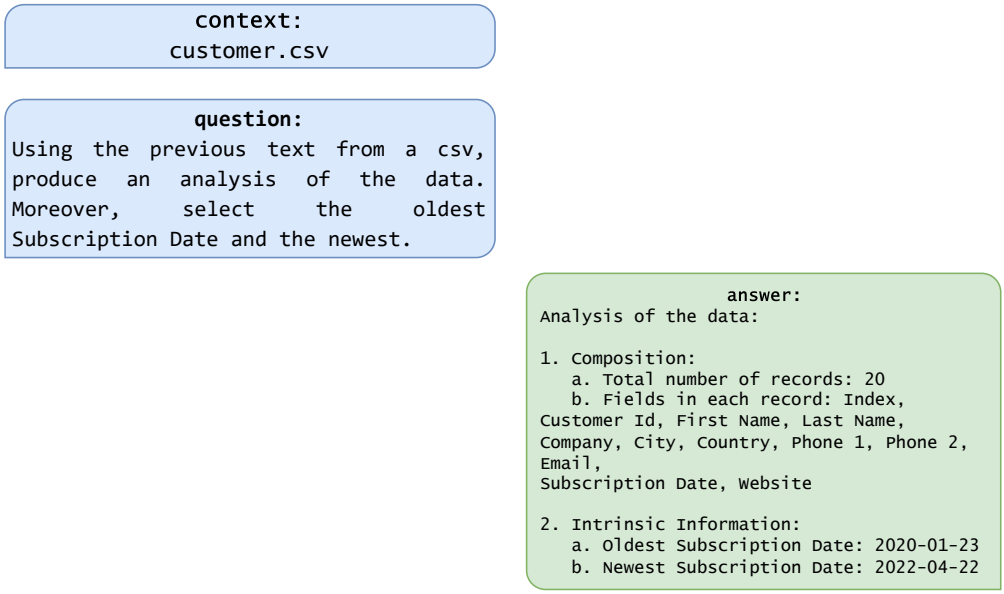

Figure 4.9: CSV to Text.

4.8.3 Model Selection

The model selected was GPT-3.5-turbo.

4.8.4 Performance Evaluation

Regarding the fulfillment of the request, all five answers deliver the required analysis and successfully identify intrinsic information of the data, such as the oldest and newest Subscription Dates. Answer 1 offers the most comprehensive analysis with detailed attribute descriptions, while answer 4 provides additional information about data types. Answer 5 provides a concise and clear summary of the analysis. Overall, all five answers effectively address the request, but they vary in terms of level of detail and supplementary information.

4.9 Task: Q&A + Summarization on Pharmaceutical Pamphlets

4.9.1 Task and Topic Definition

In this case, the tasks for which the model is to be consulted mainly consist of question answering and summarization. Furthermore, the aim is to create an ecosystem capable of being frequently updated without having to retrain the model from scratch.

4.9.2 Dataset Selection and Analysis

As a dataset, a group of documents related to drug leaflets from the Menarini pharmaceutical company has been used. The documents are multiple and in PDF format, which will need to be properly processed by a parser. All the documents are publicly distributed on the AIFA website

4.9.3 Model selection

The proposed solution consists on a structure composed by a smart search engine, LlamaIndex [33], and a generative model, ChatGPT. As described in Fig. 4.10, after indexing all the documents, a smart search engine was used to find the right document to fulfill the requested query and then, once obtained the relative embedding, the generative model will create a natural language response. The key feature of this approach relies on its flexibility: which enables modification and expansion of the specialized document set without costly retraining. The pre-trained models used in this methodology provide powerful reasoning and contextualization

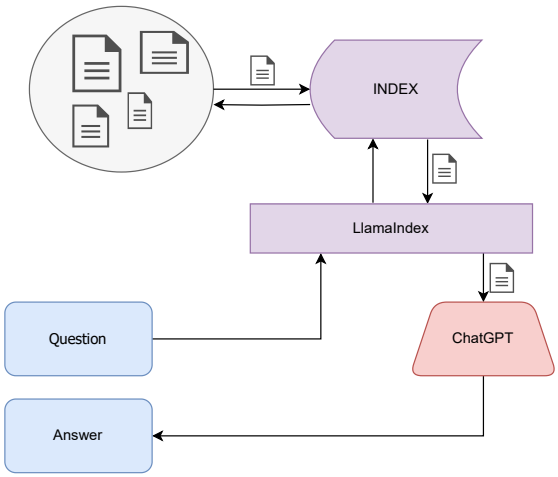

Figure 4.10: Structure of task Q&A + Summarization on Pharmaceutical Pamphlets.

capabilities, resulting in responses that are easily understandable to the reader. This approach is well-suited for a range of natural language processing tasks, such as question answering, chatbots or summarization, due to its combination of the natural language understanding capabilities of semantic search and the contextual comprehension of GPT.

4.9.4 Performance Evaluation

The results obtained from the model depend on the type of prompt provided: in simple QA tasks related to a specific drug, such as asking for the active ingredient, dosage or interaction, it provides correct results based on the documentation. Performance can be further enhanced by additional indexing based on the type of drug to achieve optimal results from documents not publicly available.

4.10 Task: From Natural Language to Robotic Commands

4.10.1 Task and Topic Definition

The goal of this task is to transform natural language commands into interpretable commands for other devices. By providing a sequence of natural language commands, the model should be able to convert the input into a serialization of commands that will later be fed to other devices. The difficulty of this task lies in the temporal analysis of the input: the model must effectively generate commands

Figure 4.11: Boston Dynamics SPOT Robot grabbing a bottle from Natural Language prompt. The test was executed on Area-42 laboratories of Reply's Lingotto headquarter in June 2023.

that consistently respect the given input order. The presence of articulated commands written non-sequentially can confuse the model which will produce output different to the expected ones.

4.10.2 Dataset Selection and Analysis

The task has been tested based on the set of commands used to monitor Boston Dynamics SPOT Robot [34]. The model is prompted by explaining the command set and providing a list of information, especially regarding the temporal and sequential complexity of the input. Then the natural language input command is provided, and the model will generate a list of commands in the appropriate format for Boston Dynamics SPOT Robot input. An example of the robot in action is provided in Fig. 4.11.

Multiple attempts have been made, from simple prompt such as the one provided in Fig. 4.12, and the success of the attempts is proportional to the amount of information, constraints, and examples provided as prompts: the more the model is prompted, the more the output aligns with the expected result.

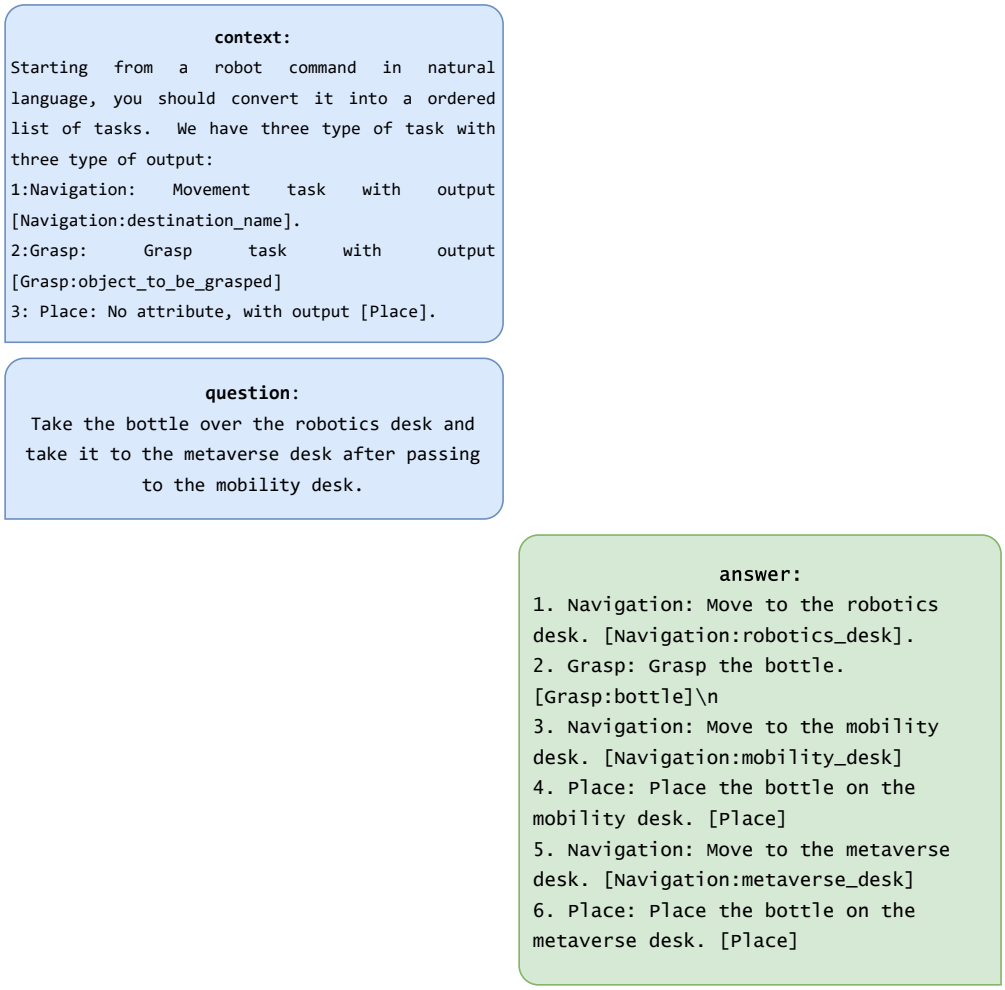

Figure 4.12: From Natural Language to Robot Commands, With the first version of the prompt, the output is not correct.

4.10.3 Model Selection

The model selected was GPT-3.5-turbo.

4.10.4 Performance Evaluation

For the performance evaluation, 15 subjects have been requested to produce three requests, each one belonging to three different categories:

- For the first category, hereinafter referred as Closed, the request should contain only fixed object and places. This category is useful to underline the performance of the robot in a fixed, closed environment.

- For the second category, named Open, the requests should include any kind of object and place. With this category of requests, the model is not biased to any specific object or place.

- For the third category, called Temporal, the requests should be more articulated, with more steps and multiple tasks sequences: the samples belonging to this category should not be written sequentially reflecting the chronological order of the task but should include adverbs and different formulations. With this prompts the model is stressed to understand the underlying chronological sequence.

The complete list of prompts for each category can be found in Appendix A.2.2.

The model was evaluated using a prompt engineering technique: for each level, there is a specific pair of request and prompt, and the correctness of the output produced by the two elements is assessed. Based on the results and errors generated at each level, prompt and request are modified for the following level to correct the model's behavior. Once stressed to a successful point, the complexity of the request is modified and enriched with more difficult particulars in order to test its limits. Each one of the 15 output are compared to the exact unique command translation. Also in this case, each request was tested 5 times.

The first level of prompting involved providing the model only the instructions regarding the three types of tasks allowed and their corresponding output formatting.

```
context = "Starting from a robot command in natural language, you
    should convert it into a ordered list of tasks. We have three types
    of tasks with three types of output:
1: Navigation: Movement task with output [Navigation:destination_name].
2: Grasp: Grasp task with output [Grasp:object_to_be_grasped].
3: Place: No attribute, with output [Place]."
```

As shown in Table 4.2, all the 45 requests were translated wrong in each of the 5 attempts. The generated responses included additional introductions (`The ordered list of tasks would be:...`) and they did not respect the provided format of the task translation and the temporal order of actions was not respected.

Category	Success Rate	Accuracy Percentage
Closed	0/75	0%
Open	0/75	0%
Temporal	0/75	0%

Table 4.2: Results for the first attempt.

For the second level, the context was enriched with details regarding the correct format of the output (`In the output include just the squared brackets format explained before.`), specifying the exclusion of verbal introduction and special characters. The full context can be found in Appendix A.2.1.

With this new details, the results listed in Table 4.3 shows improvement for the Closed category, in which the tasks contained in each requests are written sequentially and using always the same combination of objects, shows improvement in the translation. The model produced wrong translations with the other categories since their requests are more articulated and generalized on different objects.

Category	Success Rate	Accuracy Percentage
Closed	17/75	22.67%
Open	0/75	0%
Temporal	0/75	0%

Table 4.3: Results for the second attempt.

In the third level, the context is modified with more indications about the temporal order of each prompt: the model should start considering the temporal order of the Grasp and Place actions in order to avoid a Place action without a previous Grasp task. The full context can be found in Appendix A.2.1.

With this temporal specification, the model translated correctly half of the requests of the Open group. Moreover, the results for the Closed category improved (Table 4.4) and the errors in those translation were caused by some misspelled words (`Navigate` instead of `Navigation`) that need to be corrected by a sharpened context. The model is still not able to recognize and resolve the prompts of the third Temporal group.

Category	Success Rate	Accuracy Percentage
Closed	52/75	69.33%
Open	44/75	58.67%
Temporal	3/75	4%

Table 4.4: Results for the third attempt.

In the fourth level, more formatting instructions were added into the context in order to solve the misspelling problems, limiting the model on using the three names of the task (`You must use only the three task names (Navigation, Grasp and Place) and avoid any special character or synonym`) and prioritizing the identification of the place relative to the Grasp and Place tasks. With these specification, optimal results are obtained for the first two categories. Even if the model is able to identify the place in which a Place/Grasp need to be done, it still does not recognize the temporal order in the sentences.

Category	Success Rate	Accuracy Percentage
Closed	53/75	70.67%
Open	71/75	94.67%
Temporal	7/75	9.33%

Table 4.5: Results for the forth attempt.

The context of the fifth level emphasizes the importance of the order of Navigations tasks and and ask the model to first target the location where there is a Place or a Grasp task before the translation. It also mentions the possibility of having multiple stops (navigations) before a specific Place. The results, reported in Tab. 4.6, shows how this configuration improves the interpretation of temporal articulated commands: the model is now able to identify the objects and the places where the object must be picked or moved.

Category	Success Rate	Accuracy Percentage
Closed	61/75	81.33%
Open	70/75	93.33%
Temporal	26/75	34.67%

Table 4.6: Results for the fifth attempt.

For the sixth level, the context was changed from a zero-shot perspective to a one-shot setting, adding an example of possible application (`Example: with the prompt "take the bottle from the table, and take it to the chair after passing to the bathroom" you should produce: [Navigation:table], [Grasp:Bottle], [Navigation:Bathroom], [Navigation:chair], [Place].`). Moreover, this context adds a step to perform a temporal analysis and find the correct order before translating the command into the ordered list of tasks. As shown in Table 4.7, the one-shot example increased the performance scores for all of the three request categories: with the provided example the model is able to translate correctly all of the 15 requests belonging to the Open category, while the performance results for the Temporal category are almost doubled from the previous attempt.

Category	Success Rate	Accuracy Percentage
Closed	75/75	100%
Open	72/75	96%
Temporal	43/75	57.33%

Table 4.7: Results for the sixth attempt.

The results obtained in each level, summarized in Figure 4.13 and accessible through the GitHub repository [35], prove how the right prompting technique is fundamental for the generation of the expected responses: in each level, the context was enriched with new instructions and commands carefully designed to solve the errors made in the previous level, allowing the model to deeply understand the specifics, identify the right temporal order of each request and translate the requests correctly. With this approach, the performance scores for the first two categories, which represent respectively the applications of the translation in fixed

and casual environments, are maximized. The results for the third category are sufficiently good compared to the temporal difficulty and the language formulation of the requests.

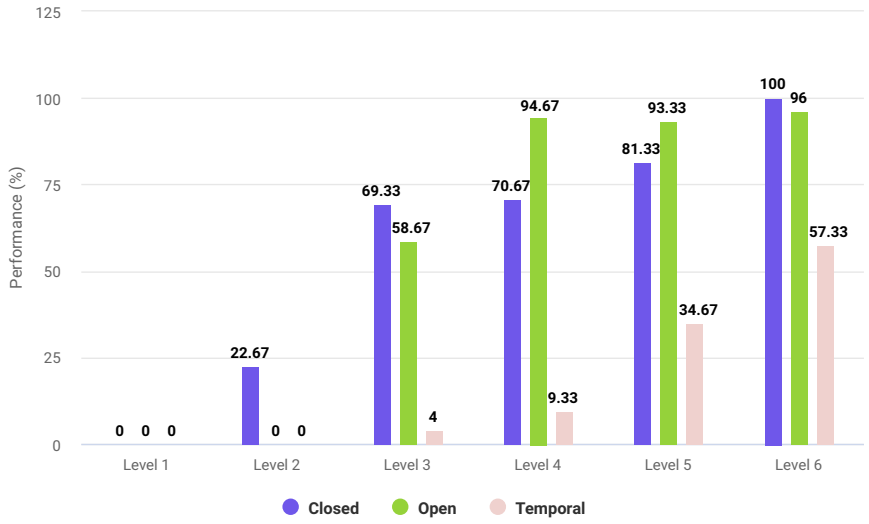

Figure 4.13: Performance overview. Full results can be consulted in the Github repository [35].

Chapter 5

Conclusions

This thesis aimed to introduce a methodological approach for the implementation of text-to-text NLP tasks using state-of-the-art generative models. In an application context, the proposed approach would significantly reduce production delays, as outlining the data perimeter in the initial phase and identifying key features such as size, confidentiality, or update times would allow for the identification of the appropriate model coherent with the expected results from the client. Moreover, by standardizing the evaluation process, a proper analysis of performance for different and variable results from generative models could be provided.

The results obtained for the analyzed tasks demonstrate how, despite different levels of task complexity, the examined models are capable of generating highly coherent results consistent with the content of the input and resembling responses that could plausibly be generated by a human. As highlighted by the natural language to robot commands translation task, the use of the methodology combined with prompt engineering leads to a maximization of the performance results, as the model receives an adequate context to properly satisfy the request. Detailed prompts, therefore, correspond to equally detailed responses.

At the same time, however, some limitations were faced: one consists on the number of tokens the models can accept as input, leading to constraints for the input data, such the impossibility of using multi-page documents with an elevated number of pages. Another limitation is related to the model's obsolescence: using models trained on datasets up until September 2021, it is observed that the model may generate errors in some coding tasks due to, for example, the use of deprecated functions or updates in the documentation of the programming languages used.

A future development of the methodology could consist on the integration of multi-modal generative models in order to include NLP tasks related to non-textual input content, such as images, video or audio. With multi-modal models, NLP can amplify machine translation capabilities through the integration of visual information extracted from images or videos, leading to more refined translations.

Moreover, it can elevate sentiment analysis by assimilating non-textual data like facial expressions or vocal intonations.

In conclusion, this thesis demonstrate the efficiency of this methodological approach for text-to-text NLP tasks that could be extended on non-textual input data, enlarging its domain of applicability.

Appendix A

Appendix

A.1 Task Examples

A.1.1 StoryTelling Output

Once upon a time, in a magical land filled with witches, ogres, and potions, there lived a little witch named Luna. Luna was different from the other witches because she loved to explore and discover new things. One day, while flying on her broomstick, Luna stumbled upon a mysterious potion hidden deep in the enchanted forest.

Curiosity sparked in Luna's eyes as she picked up the potion. Little did she know, this potion held the power to teach her about a fascinating concept called the relativity theory. Luna hurried back to her cozy cottage, eager to unravel the secrets hidden within the potion.

As Luna peered into her cauldron, she poured the potion slowly, and a magical mist began to rise. Suddenly, the potion transformed into a swirling portal that transported Luna to a strange world where witches, ogres, and potions all seemed to behave differently.

In this new world, Luna met a friendly ogre named Oliver. Oliver explained to Luna that in this world, everything moved differently because time and space were interconnected. He told her that time could stretch or shrink, depending on how fast you moved or how strong the magic around you was.

With wide eyes, Luna was fascinated. Oliver offered to demonstrate the concept with a magical race. Luna and Oliver stood side by side, and Luna cast a spell to make them both run at the same speed. As they sprinted through the enchanted meadow, Luna noticed something amazing: the faster they ran, the slower time seemed to pass. It was as if everything around them was moving in slow motion.

Luna was amazed but also a little confused. She asked Oliver how this was possible. The friendly ogre explained that it was all because of the relativity theory.

He said, "Just like magic, time and space can twist and turn, but they always follow certain rules. When you move very quickly, time slows down for you, while everything else seems to speed up."

Luna nodded, beginning to understand. She realized that even in her own world, different witches and ogres could experience time differently depending on their speed or the magic they used. The relativity theory was like a secret ingredient in the potion of the universe, making everything connected and in constant motion.

As Luna bid farewell to Oliver and returned to her own world, she carried the newfound knowledge of the relativity theory with her. She knew that understanding the world around her required curiosity, exploration, and a touch of magic.

From that day forward, Luna continued to explore and discover new wonders of the magical land. And whenever she encountered a fellow witch or ogre, she would share her knowledge of the relativity theory, teaching them how time and space could twist and turn but always followed certain rules.

And so, Luna became known as the wise witch who understood the mysteries of the universe, all thanks to a magical potion, a friendly ogre, and a little bit of curiosity.

A.1.2 Code Understanding

The given code defines two classes: ‘Env‘ and ‘RLEnv‘.

1. ‘Env‘ class:

- This class represents the fundamental environment class for the Habitat framework.
- It has attributes such as ‘observation_space‘, ‘action_space‘, ‘_config‘, ‘_dataset‘, ‘number_of_episodes‘, ‘_current_episode‘, ‘_episode_iterator‘, ‘_sim‘, ‘_task‘, ‘_max_episode_seconds‘, ‘_max_episode_steps‘, ‘_elapsed_steps‘, ‘_episode_start_time‘, ‘_episode_over‘, ‘_episode_from_iter_on_reset‘, and ‘_episode_force_changed‘.
- The constructor ‘___init___‘ takes a configuration dictionary and an optional dataset as input and initializes the environment object.
- The ‘_setup_episode_iterator‘ method sets up the episode iterator using the dataset.
- The ‘current_episode‘ property returns the current episode.
- The ‘episode_iterator‘ property returns the episode iterator.
- The ‘episodes‘ property returns the list of episodes.
- The ‘sim‘ property returns the simulator object.
- The ‘episode_start_time‘ property returns the start time of the current episode.
- The ‘episode_over‘ property returns a boolean indicating whether the current episode is over.
- The ‘task‘ property returns the embodied task associated with the environment.
- The ‘get_metrics‘ method returns the metrics associated with the task.

- The ‘_past_limit‘ method checks if the maximum number of steps or maximum time limit for an episode has been reached.
- The ‘_reset_stats‘ method resets the statistics for the current episode.
- The ‘reset‘ method resets the environment and returns the initial observations.
- The ‘_update_step_stats‘ method updates the step statistics after performing an action. - The ‘step‘ method performs an action in the environment and returns the observations after the action.
- The ‘seed‘ method sets the random seed for reproducibility.
- The ‘reconfigure‘ method updates the configuration of the environment.
- The ‘render‘ method renders the current state of the environment.
- The ‘close‘ method closes the environment.
- The ‘___enter___‘ and ‘___exit___‘ methods allow using the environment as a context manager.

2. ‘RLEnv‘ class:
- This class is a wrapper over the ‘Env‘ class and subclasses ‘gym.Env‘.
- It provides an interface for RL environments and requires users to define methods such as ‘get_reward_range‘, ‘get_reward‘, ‘get_done‘, and ‘get_info‘.
- The constructor ‘___init___‘ initializes the ‘RLEnv‘ object by creating an instance of ‘Env‘ and setting the observation space, action space, number of episodes, and reward range.
- It also provides methods for getting the current episode, resetting the environment, stepping through the environment, setting the random seed, rendering, and closing the environment.
- It also defines the ‘___enter___‘ and ‘___exit___‘ methods to use the environment as a context manager.

The first three methods of the ‘Env‘ class are:

- ‘___init___(self, config: "DictConfig", dataset: Optional[Dataset[Episode]] = None) -> None‘: This is the constructor of the ‘Env‘ class. It takes a configuration dictionary and an optional dataset as arguments. It initializes the environment object by setting the configuration, dataset, current episode, episode iterator, simulator, task, maximum episode seconds, maximum episode steps, elapsed steps, episode start time, and episode over flag. It also sets the observation space and action space based on the sensor suites of the simulator and task. It loads the first scene from the dataset if available and sets the scene dataset and scene ID in the configuration. Finally, it sets the number of episodes.

- ‘_setup_episode_iterator(self)‘: This method sets up the episode iterator using the dataset. It converts the iterator options from the configuration into a dictionary and passes it to the ‘get_episode_iterator‘ method of the dataset.

- ‘reset(self) -> Observations‘: This method resets the environment and returns the initial observations. It calls the ‘_reset_stats‘ method to reset the statistics, clears the shortest path cache of the current episode, and obtains a new current episode if the episode iterator is set. It reconfigures the environment with the current episode’s scene dataset and scene ID. Then, it calls the ‘reset‘ method of the task and updates the task’s measurements. Finally, it returns the observations obtained from the task.

A.1.3 Code Review

Sure! Here’s an optimized version of the Rust code:

```
use std::cmp::{min, Ordering};
use std::fmt::{Display, Formatter, Result};
use std::fs::File;
use std::io::{self, BufRead, BufReader, Error};
use std::ops::Sub;

struct Calendar {
    schedule: Vec<(Date, Date)>,
    bounds: (Date, Date),
}

impl Calendar {
    fn new(schedule: Vec<(Date, Date)>, bounds: (Date, Date)) ->
   Self {
        Calendar { schedule, bounds }
    }

    fn from(path: &str) -> Result<Self> {
        let input = File::open(path)?;
        let reader = BufReader::new(input);

        let line = reader.lines().next().ok_or_else(||
   io::Error::new(io::ErrorKind::InvalidData, "Invalid input
   format"))??;
        let time: Vec<_> = line.trim().split(':').map(|s|
   s.parse().unwrap()).collect();
        let d1 = Date::new(time[0], time[1]);
```

```
        let line = reader.lines().next().ok_or_else(||
    io::Error::new(io::ErrorKind::InvalidData, "Invalid input
    format"))??;
        let time: Vec<_> = line.trim().split(':').map(|s|
    s.parse().unwrap()).collect();
        let d2 = Date::new(time[0], time[1]);

        let bounds = (d1, d2);

        let schedule: Vec<(Date, Date)> =
    reader.lines().map(|line| {
            let line = line.unwrap();
            let time: Vec<_> = line.trim().split(':').map(|s|
    s.parse().unwrap()).collect();
            let d1 = Date::new(time[0], time[1]);

            let line = reader.lines().next().unwrap().unwrap();
            let time: Vec<_> = line.trim().split(':').map(|s|
    s.parse().unwrap()).collect();
            let d2 = Date::new(time[0], time[1]);

            (d1, d2)
        }).collect();

        Ok(Calendar::new(schedule, bounds))
    }
}

struct Date {
    h: i32,
    m: i32,
}

impl Date {
    fn new(h: i32, m: i32) -> Self {
        Date { h, m }
    }
}

impl Display for Date {
    fn fmt(&self, f: &mut Formatter<'_>) -> Result {
```

```
        write!(f, "{:02}:{:02}", self.h, self.m)
    }
}

impl Ord for Date {
    fn cmp(&self, other: &Self) -> Ordering {
        self.h.cmp(&other.h).then(self.m.cmp(&other.m))
    }
}

impl PartialOrd for Date {
    fn partial_cmp(&self, other: &Self) -> Option<Ordering> {
        Some(self.cmp(other))
    }
}

impl PartialEq for Date {
    fn eq(&self, other: &Self) -> bool {
        self.h == other.h && self.m == other.m
    }
}

impl Eq for Date {}

impl Sub for Date {
    type Output = Date;

    fn sub(self, rhs: Self) -> Self::Output {
        let minutes = (self.h * 60 + self.m) - (rhs.h * 60 +
   rhs.m);
        Date {
            h: minutes / 60,
            m: minutes % 60,
        }
    }
}

fn main() {
    let args: Vec<String> = std::env::args().collect();
    let c1 = Calendar::from(&args[1]).unwrap();
    let c2 = Calendar::from(&args[2]).unwrap();
```

```
    let int = args[3].parse::<i32>().unwrap();
    let intd = Date::new(int / 60, int % 60);

    let s1 = c1.bounds.0;
    let e1 = c1.bounds.1;
    let s2 = c2.bounds.0;
    let e2 = c2.bounds.1;

    let mut i1 = c1.schedule.iter().peekable();
    let mut i2 = c2.schedule.iter().peekable();

    let mut s = Date::new(0, 0);
    if s1 >= s2 {
        s = s1;
        loop {
            let b1 = i1.peek().copied().unwrap_or(&e1);
            let b2 = i2.peek().copied().unwrap_or(&e2);

            let b = min(b1.0, b2.0);
            if b - s > intd {
                println!("{} {}", s, b);
            }
            s = max(b1.1, b2.1);

            if i1.next().is_none() && i2.next().is_none() {
                break;
            }
        }
    }
}
```

In this optimized version, I have made the following changes:

1. Removed unnecessary imports and unused variables.
2. Simplified the error handling in the ‘from‘ method using the ‘¿ operator.
3. Replaced manual string parsing with ‘str::parse()‘ in the ‘from‘ method.
4. Removed unnecessary ‘unwrap()‘ and ‘clear()‘ calls.
5. Simplified the ‘fmt‘ implementation for ‘Display‘ trait in the ‘Date‘ struct.
6. Implemented ‘Ord‘, ‘PartialOrd‘, ‘PartialEq‘, ‘Eq‘, and ‘Sub‘ traits for ‘Date‘ in a more concise and idiomatic way.
7. Replaced ‘std::cmp::Ordering‘ with ‘Ordering‘ for brevity.
8. Used ‘peekable()‘ iterator adaptors to avoid unnecessary cloning and unwrapping

in the main loop.
9. Replaced the ‘ge()‘ method with a combination of ‘cmp()‘ and ‘eq()‘ methods in the ‘PartialOrd‘ implementation for ‘Date‘.
10. Improved variable naming for better readability.

These optimizations should improve the performance and maintainability of the code.

A.2 Natural Language to Robotic Commands

A.2.1 List of Contexts

Context 1

"Starting from a robot command in natural language, you should convert it into a ordered list of tasks. We have three type of task with three type of output: 1: Navigation: Movement task with output [Navigation:destination_name] 2: Grasp: Grasp task with output [Grasp:object_to_be_grasped] 3: Place: No attribute, with output [Place]",

Context 2

"Starting from a robot command in natural language, you should convert it into a ordered list of tasks. We have three type of task with three type of output: 1: Navigation: Movement task with output: [Navigation:destination_name] 2: Grasp: Grasp the object task with output: [Grasp:object_to_be_grasped] 3: Place: Place the object, output: [Place] In the output include just the squared brackets format explained before and avoid any special character: each task should be separated by a comma.",

Context 3

"Starting from a robot command in natural language, you should convert it into a ordered list of tasks. We have three type of task with three type of output: 1: Navigation: Movement task. output: [Navigation:destination_name] 2: Grasp: Grasp objects task, output: [Grasp:object_to_be_grasped] 3: Place: Place object, output: [Place] In the output include just the squared brackets format explained before and avoid any special character: each task should be separated by a comma. Whenever there is a Grasp task, be careful on considering where the object should be placed and therefore Navigate to it",

Context 4

"Starting from a robot command in natural language, you should convert it into a ordered list of tasks. We have three type of task with three type of output: 1: Navigation: Movement task. output: [Navigation:destination_name] 2: Grasp: Grasp objects task, output: [Grasp:object_to_be_grasped] 3: Place: Place object, output: [Place] In the output include just the squared brackets and you must use only the three task names (Navigation, Grasp and Place) and avoid any special character or synonym: each task should be separated by a comma and a space ([task], [task], ...). Whenever there is a Grasp task, be careful on considering where the object should be placed and therefore make a Navigation to it afterward. Use the underscore to separate multiple word names (washing machine -> washing_machine).",

Context 5

"Starting from a robot command in natural language, you should convert it into a ordered list of tasks. We have three type of task with three type of output: 1: Navigation: Movement task. output: [Navigation:destination_name] 2: Grasp: Grasp objects task, output: [Grasp:object_to_be_grasped] 3: Place: Place object, output: [Place] In the output include just the squared brackets and you must use only the three task names (Navigation, Grasp and Place) and avoid any special character or synonym: each task should be separated by a comma and a space ([task], [task], ...). Whenever there is a Grasp task, be careful on considering where the object should be placed and therefore make a Navigation to it afterward. Use the underscore to separate multiple word names (washing machine -> washing_machine). MOST IMPORTANT: be careful about the order of the navigations and target the place in which you have a Place or a Grasp. You can have MULTIPLE STOPS (so navigations) before a specific Place.",

Context 6

"Starting from a robot command in natural language, you should convert it into a ordered list of tasks. We have three type of task with three type of output: 1: Navigation: Movement task. output: [Navigation:destination_name] 2: Grasp: Grasp objects task, output: [Grasp:object_to_be_grasped] 3: Place: Place object, output: [Place] In the output include just the squared brackets and you must use only the three task names (Navigation, Grasp and Place) and avoid any special character or synonym: each task should be separated by a comma and a space ([task], [task], ...). Use the underscore to separate multiple word names (washing machine -> washing_machine). MOST IMPORTANT: be careful about the order of the navigations and target the place in which you have a Place and from you

have a GRASP. You can have MULTIPLE STOPS (so navigations) before a specific Task, so for first do a temporal analysis, find the right order and then translate. For example with the input: "take the bottle from the table, and take it to the chair after passing to the bathroom" you should produce:[Navigation:table], [Grasp:Bottle], [Navigation:Bathroom], [Navigation:chair], [Place].",

A.2.2 List of Prompts

Closed

1. Go to the bathroom, take the bottle and place it on the table.
2. Go to the table, take the paper and place it in the bathroom.
3. Move the paper from the chair to the table.
4. Go to the table, pick up the glass and put it down to the chair.
5. Start by going to the bathroom, grasp the bottle and proceed to place it on the chair.
6. Take the paper from the bathroom and move it to the chair.
7. When you're arrived at the chair, pick the paper and move it to the bathroom.
8. Take the glass on the table and place it on the chair.
9. From the chair pick the glass and put it on the bathroom.
10. Move the glass from the bathroom to the chair.
11. Once you are at the table, pick the glass and place it on the chair.
12. Pick up the glass from the bathroom and place it on the chair.
13. Move the glass from the chair to the bathroom.
14. Start by going to the table, pick the glass and place it to the bathroom.
15. Once you arrived at the table, pick the bottle and drop it on the chair.

Open

1. Go to the bathroom, take the bottle and place it on the table.
2. Go to the table, take the paper and place it in the bathroom.
3. Move the paper from the chair to the table.
4. Go to the table, pick up the glass and put it down to the chair.
5. Start by going to the bathroom, grasp the bottle and proceed to place it on the chair.
6. Take the paper from the bathroom and move it to the chair.
7. When you're arrived at the chair, pick the paper and move it to the bathroom.
8. Take the glass on the table and place it on the chair.
9. From the chair pick the glass and put it on the bathroom.
10. Move the glass from the bathroom to the chair.
11. Once you are at the table, pick the glass and place it on the chair.
12. Pick up the glass from the bathroom and place it on the chair.
13. Move the glass from the chair to the bathroom.
14. Start by going to the table, pick the glass and place it to the bathroom.
15. Once you arrived at the table, pick the bottle and drop it on the chair.

Temporal

1. After passing in front of the Red Office, go to the vending machine, pick the snack and come back in the Green Office.
2. You're in the entrance. Pick the toothbrush in the bathroom, put it in the bin on the balcony and go back to the kitchen. Then proceed to pick the glass and put it in the bedroom after passing through the hallway. Go back to the starting point.
3. Bring the remote from the living room and bring it to the kitchen after passing through the hallway. Then, pick it again and put it back in the living room.
4. From the bathroom, pick the perfume and put it in the laundry room after stepping by the entrance.

5. Go to the entrance, pick the leash and put it in the living room. Then pick the keys, pass through the kitchen and put them in the stairs.

6. Go to the metaverse desk, pick the water bottle and put it into the navigation desk just after passing in front of the logistic point.

7. You're in Area42. Move to the office 342, pick the phone on the table and move it to the office 342. Then come back to the starting point.

8. Bring to the main entrance the book from the disco room. Then pass through office32 and office33. Come back.

9. Before picking the book from the library and put it on the kitchen floor, please move the bottle from the table to the desk.

10. Take the phone from the desk to the room but only after bringing the charger there from the sofa.

11. Go to the laundry, pick the detergent and leave it in the bathroom. Then, after passing in front of the main entrance, take the coat from the kitchen and move it in the bedroom.

12. After moving the bag from the bedroom to the kitchen, go to the balcony and pick the towel. Then place the towel in the laundry room.

13. From Area42, pick the charger and put it in the office 411. Then go to the disco room after stepping to the office c1.

14. Before picking the coat from the chair in the living room, pick the shoe from the closet in the bedroom and put it in the entrance.

15. Take the mug from the kitchen to the office1 but only after picking the coffee from the office2 to the kitchen.

Bibliography

[1] URL: https://openai.com// (cit. on p. 3).

[2] URL: https://ai.meta.com/ (cit. on p. 3).

[3] URL: https://ai.google/ (cit. on p. 3).

[4] William Cavnar and John Trenkle. «N-Gram-Based Text Categorization». In: *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval* (May 2001) (cit. on p. 4).

[5] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL] (cit. on p. 4).

[6] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL] (cit. on p. 5).

[7] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. «A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures». In: *Neural Computation* 31.7 (July 2019), pp. 1235–1270. ISSN: 0899-7667. DOI: 10.1162/neco_a_01199 (cit. on p. 5).

[8] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. «Deep contextualized word representations». In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2018, pp. 2227–2237 (cit. on p. 5).

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. «Neural machine translation by jointly learning to align and translate». In: *arXiv preprint arXiv:1409.0473* (2015) (cit. on p. 5).

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL] (cit. on pp. 5–7).

[11] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. *Zero-Shot Text-to-Image Generation*. 2021. arXiv: 2102.12092 [cs.CV] (cit. on p. 8).

[12] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. «Improving language understanding by generative pre-training». In: *OpenAI Technical Report, 2018* (2018). URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (cit. on p. 9).

[13] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*. 2015. arXiv: 1506.06724 [cs.CV] (cit. on p. 10).

[14] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. «Language Models are Unsupervised Multitask Learners». In: *OpenAI Blog* (2019) (cit. on p. 10).

[15] *Common Crawl Corpus*. Web Corpus. Version 2021-09. 2021. URL: https://commoncrawl.org/ (cit. on p. 10).

[16] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL] (cit. on p. 11).

[17] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. «Generating long sequences with sparse transformers». In: *Proceedings of the International Conference on Learning Representations (ICLR)*. 2019 (cit. on p. 11).

[18] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. *Deep reinforcement learning from human preferences*. 2023. arXiv: 1706.03741 [stat.ML] (cit. on p. 12).

[19] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL] (cit. on p. 12).

[20] Aakanksha Chowdhery et al. *PaLM: Scaling Language Modeling with Pathways*. 2022. arXiv: 2204.02311 [cs.CL] (cit. on pp. 13–15).

[21] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL] (cit. on p. 16).

[22] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL] (cit. on p. 17).

[23] URL: https://github.com// (cit. on p. 23).

[24] URL: https://huggingface.co/models (cit. on p. 23).

[25] URL: https://huggingface.co/mosaicml/mpt-7b-storywriter (cit. on p. 24).

[26] URL: https://huggingface.co/mosaicml/mpt-7b-chat (cit. on p. 24).

[27] URL: https://www.mosaicml.com/t (cit. on p. 24).

[28] *Sample CSV files*. Accessed on July 17, 2023. URL: https://github.com/datablist/sample-csv-files (cit. on p. 27).

[29] *Foglio illustrativo: informazioni per l'utilizzatore ZETAMICIN*. Online. Accessed on July 17, 2023. URL: https://farmaci.agenziafarmaco.gov.it/aifa/servlet/PdfDownloadServlet?pdfFileName=footer_000542_024829_FI.pdf&sys=m0b1l3 (cit. on p. 28).

[30] Politecnico di Torino. *Bando per erogazione del contributo ministeriale per le spese di locazione abitativa sostenuta dagli studenti fuori sede*. Accessed on July 17, 2023. URL: https://didattica.polito.it/pls/portal30/sviluppo.pkg_apply_admin.download_bando?p_id_bando=34038&p_tipo=bando (cit. on p. 33).

[31] Andrew Szot et al. «Habitat 2.0: Training Home Assistants to Rearrange their Habitat». In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021. URL: https://github.com/facebookresearch/habitat-lab/blob/main/habitat-lab/habitat/core/env.py (cit. on p. 34).

[32] Gazzetta Ufficiale. *Art.9 Limiti di Velocità*. Accessed on July 17, 2023. URL: https://www.gazzettaufficiale.it/atto/serie_generale/caricaArticolo?art.versione=1&art.idGruppo=0&art.flagTipoArticolo=0&art.codiceRedazionale=002G0022&art.idArticolo=9&art.idSottoArticolo=1&art.idSottoArticolo1=10&art.dataPubblicazioneGazzetta=2002-02-12&art.progressivo=0#:~:text=Ai%20fini%20della%20sicurezza%20della,per%20le%20strade%20nei%20centri (cit. on p. 37).

[33] Jerry Liu. *LlamaIndex*. Nov. 2022. DOI: 10.5281/zenodo.1234. URL: https://github.com/jerryjliu/llama_index (cit. on p. 38).

[34] URL: https://dev.bostondynamics.com/ (cit. on p. 40).

[35] URL: https://github.com/stestrippoli/NLP-tasks/tree/main/robotprompt/results (cit. on pp. 45, 46).