Università degli Studi di Trieste
Dipartimento di Matematica, Informatica e Geoscienze

Corso di Laurea Magistrale in
Data Science and Scientific Computing
TESI DI LAUREA MAGISTRALE

A Dialectic Pipeline for Improving LLM Robustness

Relatore:
Prof. Luca Bortolussi
Correlatore:
Dott. Gabriele Sarti

Candidata:
Sara Candussio

Anno Accademico 2023/2024

Contents

1 Introduction **1**

2 Literature review **3**
2.1 Transformer models . . . 3
2.1.1 The Self-Attention mechanism . . . 3
2.1.2 Multi-Head Self-Attention . . . 5
2.1.3 The Transformer block . . . 6
2.1.4 The Language Model Head . . . 7
2.1.5 Embedding words into numerical vectors . . . 8
2.1.6 Causal self-attention in *decoder-only* Transformers . . . 9
2.2 Improvements in modern Large Language Models . . . 11
2.2.1 The RoPE embedding . . . 11
2.2.2 Pre-normalization and post-normalization architecture . . . 12
2.2.3 Sliding Window Attention (SWA) . . . 13
2.2.4 Multi-Head Attention, Multi-Query Attention and Grouped Query Attention . . . 14
2.2.5 Decoding strategies . . . 14
2.3 Large Language Models employed . . . 15
2.3.1 `Gemma-2` family . . . 16
2.3.2 `Phi-3` family . . . 17
2.3.3 `LlaMa-3.1` family . . . 18
2.3.4 Models summary . . . 19
2.4 Supervised Fine-Tuning and RAG strategies . . . 19
2.5 In-context learning or "few-shot" prompting . . . 21
2.6 Chain-of-Thought (CoT) prompting . . . 22
2.7 Emulating or *understanding* patterns . . . 24
2.8 Grounding answers in a *selected* context . . . 26
2.8.1 RE-RAG . . . 26
2.8.2 Using NLI verifiers . . . 26
2.8.3 System 2 Attention (S2A) . . . 28
2.9 Self-refinement approaches . . . 29
2.9.1 `SELF-REFINE` algorithm . . . 29
2.9.2 `SELF-CORRECTION` algorithm . . . 30
2.9.3 Reflexion . . . 32
2.10 Reasoning on the context . . . 33
2.10.1 Rethinking with retrieval . . . 33
2.10.2 Better Multi-Hop Reasoners . . . 34
2.10.3 MIRAGE . . . 35
2.11 Literature connections with our method . . . 38

3 Data details **39**
3.1 HotpotQA . . . 40
3.1.1 Dataset description . . . 40
3.1.2 Data processing . . . 41
3.2 WikiHop . . . 45
3.2.1 Dataset description . . . 45
3.2.2 Data processing . . . 46
3.3 Datasets' summary statistics . . . 48

4 Methods **49**
4.1 Reasoning or simply imitating? . . . 50
4.2 A dialectic pipeline . . . 52
4.3 Answering given the context . . . 53
4.3.1 WikiHop context summarization . . . 54
4.3.2 MIRAGE context filtering . . . 55
4.4 Answers format: the `guidance` framework . . . 56
4.5 Thesis . . . 58
4.6 Antithesis . . . 58

4.6.1 The importance of questioning previous statements . . . 58
4.6.2 The influence of the given examples . . . 60
4.7 Synthesis . . . 63
4.8 Assessing performances . . . 65
4.9 Comparison with Chain-of-Thought prompting . . . 66

5 Results **68**
5.1 Does the dialectic pipeline work? . . . 68
5.2 How models' architectures and number of parameters impact on pipeline performances . 68
5.3 Robustness with respect to different datasets . . . 69
5.4 Pipeline variations . . . 72
5.5 *Overthinking* can be harmful (even for LLMs) . . . 76
5.6 Chain-of-Thought prompting comparison . . . 77
5.7 Context filtering and summarization . . . 79

6 Conclusions and future directions **83**

A PECoRe invocation for context filtering **i**

B Usage of `guidance` framework for multiple-choice questions **ii**

C Thesis **iii**

D Antithesis **iii**

E Synthesis **iv**

F Relative improvements between different pipeline steps **v**

L'inferno dei viventi non è qualcosa che sarà; se ce n'è uno, è quello che è già qui, l'inferno che abitiamo tutti i giorni, che formiamo stando insieme.
Due modi ci sono per non soffrirne.
Il primo riesce facile a molti: accettare l'inferno e diventarne parte fino al punto di non vederlo più.
Il secondo è rischioso ed esige attenzione e apprendimento continui:
cercare e saper riconoscere chi e cosa, in mezzo all'inferno, non è inferno, e farlo durare, e dargli spazio.

Le Città Invisibili, Italo Calvino

Abstract

Assessing ways in which Language Models can reduce their hallucinations and improve the outputs' quality is crucial to ensure their large-scale use.

However, methods such as fine-tuning on domain-specific data or the training of a separate *ad hoc* verifier require demanding computational resources (not feasible for many user applications) and constrain the models to specific fields of knowledge.

In this thesis, we propose a dialectic pipeline that preserves LLMs' generalization abilities while improving the quality of its answer via self-dialogue, enabling it to reflect upon and correct tentative wrong answers.

We experimented with different pipeline settings, testing our proposed method on different datasets and on different families of models. All the pipeline stages are enriched with the relevant context (in an oracle-RAG setting) and a study on the impact of its summarization or its filtering is conducted.

We find that our proposed dialectic pipeline is able to outperform by significative margins the standard model answers and that it consistently achieves higher performances than Chain-of-Thought only prompting.

Italian abstract

Al fine di garantire l'uso su larga scala del Language Models, è fondamentale trovare delle strategie con cui è possibile ridurre le loro allucinazioni e quindi migliorare la qualità dei loro outputs.

Tuttavia, metodi come il fine-tuning su dati specifici o l'addestramento di verifiers *ad hoc* richiedono risorse computazionali elevate (non accessibili per molte applicazioni utili all'utente) e limitano i modelli ad alcuni precisi ambiti di conoscenza.

In questa tesi, proponiamo una pipeline dialettica che preserva le capacità di generalizzazione dei LLMs migliorando al contempo la qualità delle loro risposte attraverso l'auto-dialogo, consentendo al modello di riflettere e correggere risposte inizialmente errate.

Abbiamo sperimentato diverse configurazioni di auto-dialogo, testando il nostro metodo su vari dataset e su diverse famiglie di Language Models. Tutte le fasi di questa pipeline dialettica sono state arricchite con il contesto rilevante per rispondere al prompt (in un setting oracle-RAG) ed è stato condotto uno studio sull'impatto del riassunto o del filtraggio di questo.

Abbiamo riscontrato che la pipeline dialettica proposta è in grado di superare con margini significativi le risposte standard del modello e che riesce ad ottenere in maniera consistente prestazioni superiori rispetto al solo prompting con Chain-of-Thought.

1 Introduction

In 2017, the field of Natural Language Processing had been revolutionized by Vaswani et al. [66], leading to a mass-scale interest towards neural networks applied to text generation. The era of Large Language Models (LLMs) began with the intuition that a recurrent structure to is not mandatory to face the given task; what is necessary is just the *attention mechanism* (and a supporting complex model structure, as we will show in section 2.1.3).

Many efforts were made by the researchers' community from that moment on in order to improve the quality of the produced text: these models were often object of the so-called *hallucinations*, i.e. the generation of false or misleading information. This phenomenon is due to many reasons, often caused by data issues: incomplete, noisy, biased or not updated training data tend to drive the auto-regressive towards the wrong output. Additionally, an unusual prompting method or an imprecise instruction could confuse the model. Different strategies are used nowadays to reduce the *hallucinations* frequency in LLMs: curated pre-training datasets and fine-tuning, RAG-powered applications (Section 2.4), human-based alignment, Chain-of-Thought prompting (2.6) are all aimed at improving the models' prediction quality.

We propose a new solution to face this problem, consisting in prompting the model to reason multiple times on the answer to a question before definitively choosing the correct option. Our approach aims at improving the quality of the *spontaneous* answer by incrementally checking whether and why it is correct. This approach constructs a *dialectic pipeline* for generating the final output, where the candidate answer is checked and examined by two steps before being effectively chosen. The term *dialectic* is due to the fact that the model is dialoguing with itself in three different stages: the candidate answer production, the first comment on the correctness of it, and the final decision on which should be the final answer. We will refer to the first stage as the *thesis*, to the second as *antithesis* and to the latter as *synthesis*. Differently from the Hegelian dialectic [75], the *antithesis* step is not forced to refute the *thesis*' one, neither is the *synthesis* to compromise between the other two. However, we find these names as representative for the role that they have in the process: the *thesis* makes a first guess, the *antithesis* has to check whether or not it is correct (and why), and the *synthesis* further merges these opinions into a final, reasoned answer.

We are going to test this method on *multi-hop* question-answering datasets, namely on tasks that require to properly merge multiple knowledge sources to answer the question. This kind of problems requires both content extraction and reasoning abilities, thus are more challenging than standard RAG-problems. We are going to consider different sub-tasks inside this broader class, consisting of multiple passages (3.3) to exploit in slightly different ways, for example by comparing them or by constructing links between them (3.1.1). In all the pipeline steps, we allow the model to access the relevant context in order to properly answer to the question while dealing with (implicit) challenging context pre-processing.

The research questions for which we search an answer through our experiments are oriented to assess whether or not this *dialectic* pipeline can compete against well-established other methods in terms of accuracy and reliability (5.6). We want to test whether the pipeline is able to reach good accuracy values and whether this is maintained when we consider different models inside the pipeline. We test both the robustness across different families of models (2.3.1, 2.3.2, 2.3.3) and with respect to the same model but with different sizes (5.2).

We also try this pipeline on different datasets to assess the non-specificity of the proposed method: we will consider two *multi-hop* datasets with various numbers of *hops* and requiring different reasoning strategies to merge them (5.3). We are going to define some pipeline *variations* that induce the models to behave differently when asked to solve a task (4.6.2).

We check whether the *synthesis* step is necessary to improve the models' performances and by which margin; this is due to the idea that the *antithesis* step could be misleaded by the context or simply wrong. Consequently, we run experiments comparing the *antithesis*' proposed answer and the *synthesis*' ones, in order to assess whether the latter is in fact useful in prediction terms (2).

Finally, we test whether the *way* in which the context is provided to the pipeline affects the answers' quality: works such as S2A (2.8.3) and RE-RAG (2.8.1) spot a light on the beneficial effect of carefully

selecting only the relevant parts of it and use only these to generate the output text for a given prompt. Consequently, we compare the answers obtained by passing the original, the summarized (4.3.1) and the filtered context to the pipeline. The filtering approach exploits a gradient-based attribution method (2.10.3) that is used to select the passages' sentences that are found to be influential for the output generation (4.3.2).

Our work is structured as follows.

- The first section (2) is dedicated to a literature overview. It delves into the technical details of Transformer models and explains the differences among those we will consider in our experiments. Then it focuses on the existing approaches to overcome the problem of *hallucinations*, such as SFT and RAG (2.4), training logical verifiers to check the correctness of the answer before outputting it (2.8.2) and prompting strategies aimed at showing the expected behaviour of the model before executing it. We consequently introduce possible alternatives to our approach, such as Chain-of-Thought prompting (2.6) and self-refinement methods (2.9). Finally, we inspect ways in which the context can improve the answers' quality (2.8, 2.10).
- We then present the data (3) and highlight the pre-processing used to make these *multi-hop* datasets also multiple choice: this is done to avoid approximate, LLM-based evaluations on the correctness of the answer.
- The methods section (4) explains largely how we constructed the pipeline (4.2), how practically we filtered (4.3.2) and summarized (4.3.1) the context and how we assessed performances (4.4, 4.8).
- Finally, we run answer the research question described before in section (5), decreeing whether or not the proposed method works and in which cases it performs better or worse. Final conclusions are drawn in section (6).

2 Literature review

In this section we go through a set of relevant works for our study. We begin by studying how the models that we use are structured, architecturally speaking; we then highlight the differences between the classes of LLMs exploited for our experiments.

Subsequently, we provide an overview of methods aimed at improving the output quality in terms of factuality, style and consistency with the given examples.

Finally, we highlight how these discoveries and proposed methods are related to our new approach, and how much it owes to other researchers' work.

2.1 Transformer models

Language Models (LMs) are inherently sequential, processing input data in a step-by-step manner. Early LMs leveraged this sequential structure through Recurrent Neural Network (RNN) architectures, which were designed to capture dependencies in data sequences. However, RNNs faced limitations in scalability due to their reliance on sequential processing and their fixed-size context representations. To address these challenges, the *attention mechanism* was introduced, providing a more flexible way to handle context by allowing models to focus on different parts of the input sequence as needed.

2.1.1 The Self-Attention mechanism

The *attention mechanism*, originally proposed by Bahdanau, Cho and Bengio [3] to improve machine traslation with RNNs, has assumed a central role when Vaswani et al. [66] published a new Language Models architecture based on it.
From that moment on, this approach has been exploited massively, displacing RNNs from the state-of-the-art techniques and becoming the new paradigm for text generation tasks.

The key point is to represent each token in the sequence with respect of the other tokens present in it, i.e. produce a meaningful *contextual representation* of the words. By comparing an item of interest to a collection of other items, we can reveal their *relevance* in the context in which they are placed. The results of these comparisons are then used to compute an output sequence for the current input sequence.

Diving into mathematical aspects, the comparison is performed by considering the dot product between two word vector representations and produces a score representing the relevance of one with respect to the other:

$$\text{score}(x_i, x_j) = x_i \cdot x_j$$

and this quantity takes values in the $\Big[-|x_i| \cdot |x_j|, |x_i| \cdot |x_j|\Big]$ interval[1], where the magnitude of the score is greater when the relevance of the tokens is greater.

Once these scores are computed, we are interested in translating them in relative relevance scores, i.e. finding a function that maps $\mathbb{R} \to [0, 1]$. One of the most popular choices in the machine learning field is the softmax$(\cdot)$ function:

$$\alpha_{ij} = \text{softmax}(\text{score}(x_i, x_j)) = \frac{\exp(\text{score}(x_i, x_j))}{\sum_j \exp(\text{score}(x_i, x_j))}$$

where the normalization is performed with respect to all the elements x_j in the sequence. Thus each word x_i will have an array of vector weights $\alpha^i = [\alpha_{i1}, ..., \alpha_{iJ}]$ associated to it.

Now that we have the proportional relevance of each word in the sequence *with respect to* x_i, we can compute a linear combination between each proportional relevance α_{ij} and each word x_j in the sequence, resulting in the relevance of word x_i in the context in which is immersed:

$$a_i = \sum_j \alpha_{ij} \cdot x_j$$

[1] If $x_i, x_j \in \mathbb{R}^n$ (i.e. they are finite-dimensional), the bounds are finite. Otherwise, $\text{score}(x_i, x_j) \in [-\infty, +\infty]$.

this is the so-called *attention weights* associated to word x_i.

The process may appear non-trivial due to the shift of focus that happens in the last step. Firstly we focus on one word, consider the relevance of the other words with respect to it; then we use this information to compose a relevance score for the considered word.

The big implicit step in this procedure is the fact that if x_j is a sequence of characters from a certain alphabet we need a further step that embeds each token into a numerical vector, such that it is possible to perform the previous computations. We will not dig deeper than this on the embedding details in this paragraph, but we will talk about this in section (2.1.5).

However, word embedding is not the only transformation imposed to tokens. The *self-attention mechanism* represents differently the same word with respect to the *role* that it plays in the algorithm. The three roles that each word may have are:

- the **query**: the current focus of attention, i.e. the word compared to the others;
- the **key**: the *other* word compared to the current focus of attention to produce the *proportional relevance weights*;
- the **value**: the *other* word used to compute each *attention weight* for the current word.

To produce these representations of the same word x_j, it is necessary to apply three different transformations that can be represented as matrices:

$$q_i = x_i \cdot W_Q \,; \quad k_j = x_j \cdot W_K \,; \quad v_j = x_j \cdot W_V$$

being the query $W_Q \in \mathbb{R}^{d \times d_k}$, key $W_K \in \mathbb{R}^{d \times d_k}$ and value $W_V \in \mathbb{R}^{d \times d_v}$ matrices.

We need to transform x_i and x_j using the matrices W^Q, W^K and W^V since, if we do not account for these projections, the model would be forced to use the same hidden vector for these three different tasks. But the query, the key and the value roles require that the same token performs different tasks: consequently, the vectors have to be transformed in order to let the matrices W^Q, W^K and W^K absorb this role in their place.

With this in mind, we can rephrase the previous self-attention mechanism as:

$$\text{score}(x_i, x_j) = q_i \cdot k_j \,; \quad a_i = \sum_j \alpha_{ij} \cdot v_j$$

A final precaution has to be made: the dot product between two vectors can assume arbitrarily large values, leading to numerical issues and hindering the effective propagation of the gradients during the training. To avoid this scenario, a re-scaling of the dot product is performed:

$$\text{score}(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

where d_k is the dimensionality of the query and the key vectors.

Transformers are able to take in input, as preceding context, a maximum number of tokens. This quantity is called *context length* and currently takes values 4K or 8K, even though many modern models are more capacious.
Previously, we considered one token in the sequence at a time; more precisely, x_i is the embedding of a word in the sequence, and it has dimension $1 \times d$, where d is the embedding dimensionality. We can now assume that the input of the self-attention operation is the matrix $X \in \mathbb{R}^{N \times d}$, containing the d-dimensionality embeddings of all the N words in the context. In the self-attention operation of the first model layer, this input has to be multiplied by the key, query and value matrices to produce the following matrices:

$$Q = XW_Q \quad K = XW_K \quad V = XW_V$$

where $Q \in \mathbb{R}^{N \times d_k}$, $K \in \mathbb{R}^{N \times d_k}$ and $V \in \mathbb{R}^{N \times d_v}$. The entire process described above can be summarized as follows

$$X' = \text{SelfAttention}(Q,\ K,\ V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where X' is the contextualized representation for the sequence produced in the first Self-Attention operation. X' will undergo further refinement and contextualization in successive layers of the Transformer model.

The great advantage of the attention mechanism is each token's the attention weight can be computed independently of the others, given the weight matrices. This means that the self-attention computation can be performed in parallel, reducing by a large factor the training and the inference time with respect to the RNNs alternatives. In practice, this means that the loss for a full sequence can be computed in a single forward pass, rather than one pass per word in the sequence like in RNNs.

2.1.2 Multi-Head Self-Attention

It may be difficult for a single triplet of weight matrices $(W_Q,\ W_K,\ W_V)$ "to capture all of the different kinds of parallel relations among its inputs"[33].

The solution proposed to this limitation is to create a set of triplets $(W_Q^h\, W_K^h,\ W_V^h)$, $h \in \{1, ..., H\}$, each of them referred to as $head^h$.
The self-attention computation is performed independently in parallel for every attention *head* (placed at the same depth in the model), and results are finally aggregated in a unified output for further processing. The reason to have so many parameters is that each head will project the input in a different representation highlighting some characteristics allowing to perform a more varied analysis.

This choice implies that each head will output a $N \times d_v$ matrix[2], thus we will have H matrices of that shape. In order to obtain an output of the same dimension of the input matrix $X \in \mathbb{R}^{N \times d_v}$, we have to project the outputs of all the heads into a single $\mathbb{R}^{N \times d_v}$ matrix.
This is obtained by first concatenating the outputs of each head (in a $\mathbb{R}^{N \times H \cdot d_v}$ matrix) and then by projecting it using an output matrix $W_O \in \mathbb{R}^{H \cdot d_v \times d}$:

$$(head_1 \oplus head_2 \oplus ... \oplus head_H) \in \mathbb{R}^{N \times H \cdot d_v}\ ; \quad (head_1 \oplus head_2 \oplus ... \oplus head_H)W_O \in \mathbb{R}^{N \times d_v}$$

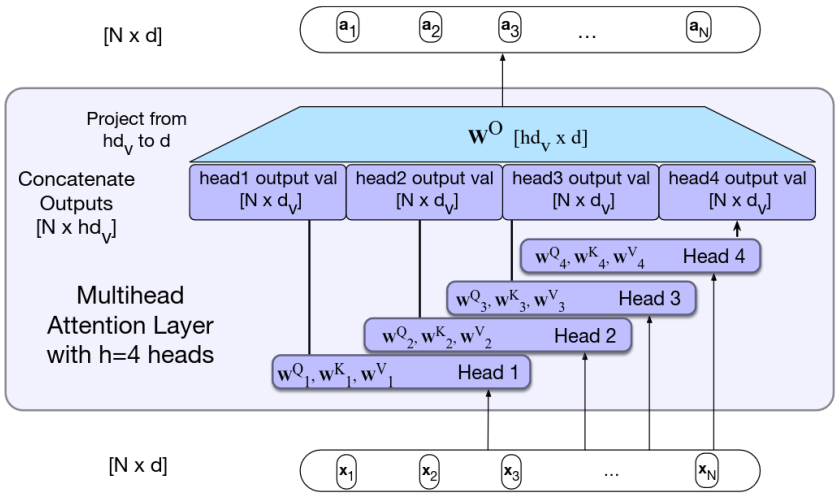


Figure 1: A schematic representation of Multi-Head Attention (MHA); each *head* is provided with its own set of key, query and value matrices. The outputs of all the heads are first concatenated and consequently projected to W_O (taken from [33]).

[2]This is due to the fact that, given $Q^h \in \mathbb{R}^{N \times d_k}$, $K^h \in \mathbb{R}^{N \times d_k}$ and $V^{\in} \mathbb{R}^{N \times d_v}$, we perform the following operations:

$$\text{softmax}\left(\frac{Q^h {K^h}^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{d_k \times d_k}\ ; \quad \text{softmax}\left(\frac{Q^l {K^l}^T}{\sqrt{d_k}}\right)V \in \mathbb{R}^{d_k \times d_k} \times \mathbb{R}^{N \times d_v} \in \mathbb{R}^{N \times d_v}$$

In matrix terms, the Multi-Head Self-Attention mechanism can be expressed as:

$$Q^h = XW_Q^h \quad K^h = XW_K^h \quad V^h = XW_V^h$$

$$head^h = \text{SelfAttention}(Q^h, K^h, V^h)$$

$$A = \text{MultiHeadAttention}(X) = (head_1 \oplus head_2 \oplus ... \oplus head_H)W_O$$

2.1.3 The Transformer block

In this section we are going to describe the structure of the Transformer unit, exploiting the Multi-Head Self-Attention operation introduced in the previous paragraphs (2.1.2).
The Transformer block is schematically represented here:

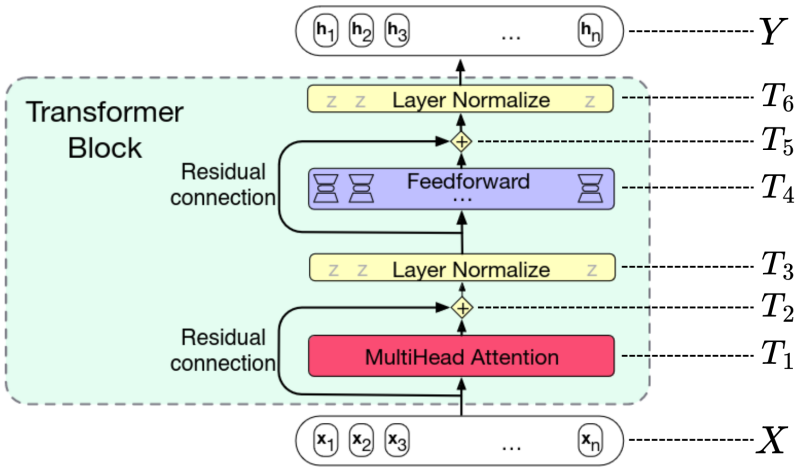


Figure 2: taken from [33]

Breaking down the process with one equation for each computation component, we obtain:

$$T_1 = \text{MultiHeadAttention}(X)\,; \quad T_2 = X + T_1\,; \quad T_3 = \text{LayerNorm}(T_2)$$

$$T_4 = \text{FFN}(T_3)\,; \quad T_5 = T_3 + T_4\,; \quad Y = \text{LayerNorm}(T_5)$$

Going in detail, the residual connections present before the Normalization layers have some benefits. By allowing information from the activation to go forward and, complementary, the gradient to go backwards skipping a layer, they improve learning and give higher-level layers direct access to information residing in lower-level ones [25].

The Normalization layer is useful to improve the training performances by keeping the hidden layer values' in a range that facilitates gradient-based training [33], [78].
The vector components x are *row-normalized* as:

$$\hat{x} = \frac{x - \mu}{\sigma}$$

where μ and σ are respectively:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i\,; \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

In addition to this, the standard implementation of layer normalization includes also two learnable parameters, γ (the gain value) and β (the offset value):

$$\text{LayerNorm}(x) = \gamma \cdot \hat{x} + \beta = \gamma \cdot \frac{x - \mu}{\sigma} + \beta$$

Finally, the Feed-Forward layer contains N fully connected two-layer networks[3] (i.e. composed pf one hidden layer and two weight matrices) that usually has $d = 512$ input, $d_h = 2048$ hidden and again $d = 512$ output neurons. The activation functions applied between these two linear layers are both Rectified Linear Unit (ReLU) functions, defined as $\text{ReLU}(x) = max(0, x)$.

The operations within the FFN can be mathematically described by the following equation:

$$\text{FFN} = max(0, max(0, x \cdot W_1 + b_1) \cdot W_2 + b_2)$$

where W_1 and W_2 are the weight matrices and b_1 and b_2 are the biases for the first and the second layer respectively.

Each of the N units in FFN is also called position-wise network, since the weights of the two matrices involved in the transformation are the same for all the processed tokens, i.e. independent of the token position.

2.1.4 The Language Model Head

The Large Language Model is composed of a certain number L of Transformer blocks stacked one after the other. The output of the last Transformer block is a $N \times d$ matrix (one $1 \times d$ state per input token). For inference, only the last state (of dimension $1 \times d$) is used to predict the next token, while the previous $(N-1) \times d$ ones are ignored[4].

In order to come out with a probability distribution over the vocabulary tokens, it is necessary to *unembed* the output of the with a $d \times |V|$ matrix. This linear layer can be learned, but more commonly we tie this matrix to (the transpose of) the embedding matrix E.

The output of this process is called *logit* or *score vector* u, since it has a score for each of the $|V|$ possible words in the vocabulary V. These scores are transformed into probabilities through the usage of a softmax function. These two operations composed the so-called *Language Model Head* of the model:

$$u = h_N^L \cdot E^T \quad \text{and} \quad y = \text{softmax}(u)$$

where $h_N^L \in \mathbb{R}^{1 \times d}$, $E^T \in \mathbb{R}^{d \times |V|}$ and $u \in \mathbb{R}^{1 \times |V|}$ (consequently also $y \in \mathbb{R}^{1 \times |V|}$).

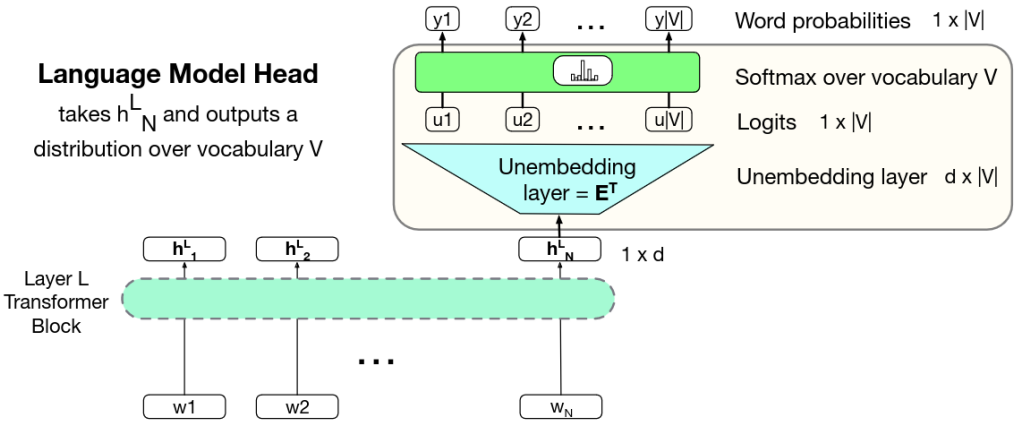


Figure 3: The Language Model Head: the circuit at the top of the last Transformer block maps the output embedding for token N from the last layer h_N^L to a probability distribution over words of the vocabulary V (taken from [33]).

The multi-layer architecture that we showed in the previous sections is referred to as *decoder-only* LLMs and it was first introduced by `GPT-1` [53] and subsequently adopted in other model architectures.
Before `GPT-1`, the original Transformer architecture (as proposed by Vaswani et al. [66]) was employed, consisting in both an *encoder* and a *decoder* part.

[3]We recall that N is the context length; thus, the Feed-Forward layer processes in parallel each item that comes out from the MultiHeadAttention and LayerNorm operations.

[4]On the opposite, for the training, each state is *projected* (as described in this section) to the vocabulary and a loss is computed in parallel as the average over the full predicted sequence against the gold sequence.

2.1.5 Embedding words into numerical vectors

The first, fundamental step that preceds all the others is the transformation of the words inside the sequence into numerical vectors on which we can perform the already explained *sequence of relevant mathematical operations*, leading to information extraction and allowing us to make inference.

Given a sequence of N words, we define the *embedding* of a sequence as the matrix $X \in \mathbb{R}^{N \times d}$, where d is the pre-defined dimension of each embedding vector.

We can imagine to define a vocabulary V of words present in the sequence of interest. Given this dictionary, we can map each word into a *one-hot encoded vector*, in which the only element which is not zeroed out is the one corresponding to the token.
Think for example of the sequence *May the Force be with you*, then the vocabulary will be made by: $V = \{$ *be*, *Force*, *May*, *the*, *you*, *with* $\}$. We can represent the word *May* as $[1, 0, 0, 0, 0, 0]$, the word *the* as $[0, 1, 0, 0, 0, 0]$ and so on. Note that all the vectors are encoded into vectors having $|V|$ elements, and that the entire sequence can be represented by a matrix $W \in \mathbb{R}^{N \times |V|}$.

To *smooth* by a certain degree the word representation, we apply to the one-hot encoded matrix a transformation $E \in \mathbb{R}^{|V| \times d}$ called *embedding matrix*:

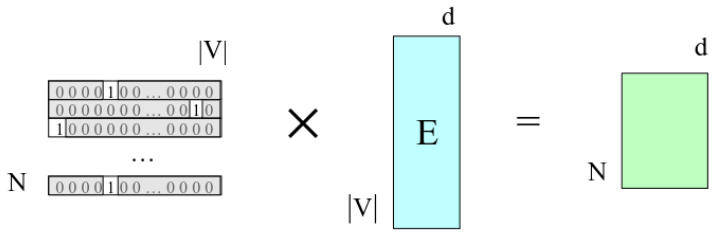


Figure 4: Creating the word embedded matrix $W \in \mathbb{R}^{N \times d}$ for the input sequence of tokens by multiplying a one-hot matrix of dimensions $N \times |V|$ by the embedding matrix $E \in \mathbb{R}^{|V| \times d}$ (taken from [33]).

The initial $N \times |V|$ words representation is referred to as *sparse*, in opposition to the *dense* one $W \in \mathbb{R}^{N \times d}$ of much lower dimensions (typically in the range of $d = 50$ to 300 dimensions) produced by the application of an embedding matrix E.

The benefit of *dense* embeddings over *sparse* one is that, due to the training objective of the embedding matrix E, these kind of transformations push similar words to be close in the *dense* vector space defined by W; this idea is inspired from the *distributional hypothesis - you shall know a word by the company it keeps* [17]. Words embeddings are used in all *Neural-Networks based* Language Models, including in RNNs.

However, Transformer-based models for text generation, differently from RNNs, do not have a sequential inductive bias by design.
In this kind of models, word embeddings are position-independent. In order to keep track of this information, we have to add to the "standard" *word embedding* $w_i \in \mathbb{R}^{1 \times d}$ (w_i is the i-th row of *words embedded* matrix $W \in \mathbb{R}^{N \times d}$) its corresponding *position embedding*, defined as a function mapping each position i into an embedding $P[i] \in \mathbb{R}^{1 \times d}$.
In this setting, the final representation of the input, the matrix $X \in \mathbb{R}^{N \times d}$ is such that each row i is the representation of the i-th token in the input, computed by adding the embedding w_i of the token that occurred at position i, to $P[i]$, the positional embedding of position i.

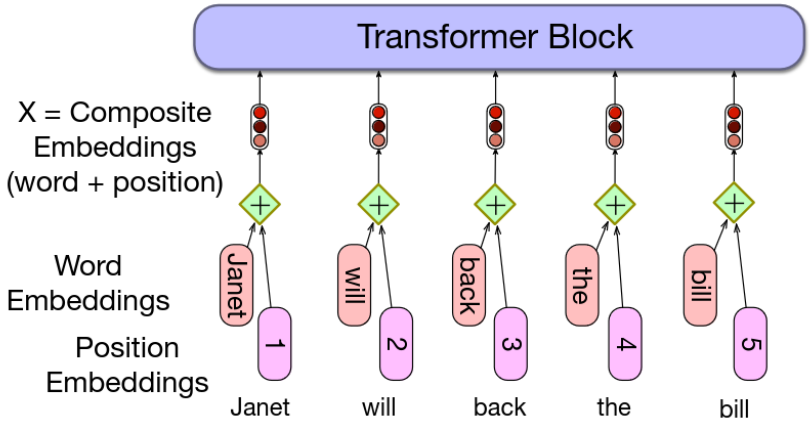


Figure 5: A simple way to model position: add an embedding of the absolute position to the token embedding to produce a new embedding of the same dimensionality (taken from [33]).

The positional embedding method proposed by Vaswani et al. [66] aims at ensuring that each sentence position $i \in 1, ..., I$ has an unique representation. Their idea is to exploit sin and cos function to achieve this goal without the need to train a separate positional embedding matrix P. do is apply to each position i to a set of d functions in order to obtain a $1 \times d$ vector for each sequence position.

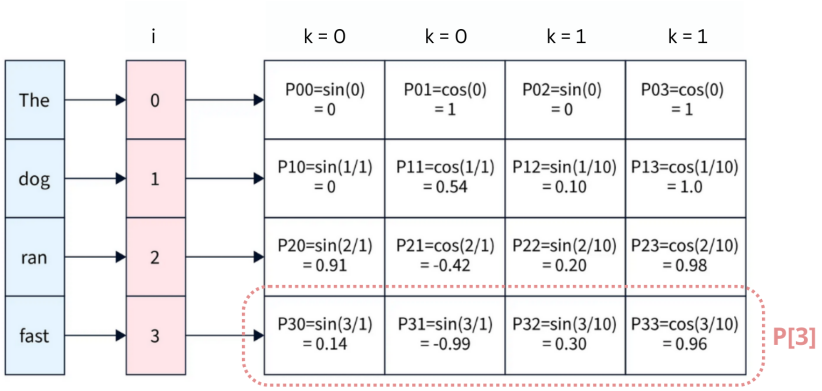


Figure 6: Toy example of Positional Encoding (with $d = 4$), inspired from [40]. Note that what matters is not the term (e.g. `fast`) the position i (e.g. $i = 3$) that is appropriately transformed through sin and cos functions.

2.1.6 Causal self-attention in *decoder-only* Transformers

The *self-attention mechanism* presented in section 2.1.1 implies that the *attention weight* associated with a token will not only depend on the past ones (i.e. the preceding context), but also on the upcoming ones. The issue that this *bidirectional attention* carries is that, in some sense, it *incorporates* the future information in the representation of the current word.

This approach is unsuited for tasks like text generation, since we expect the model to correctly predict the *next* word, and this task is trivial if the model already knows the following ones. Practically speaking, by using *bidirectional self-attention* Transformer in a "guess which is the next word" problem we are allowing the model to cheat.

Due to the *inductive bias* of auto-regressive language modeling, a **causal mask** is used to hide future tokens from the context mixing operation of the attention block. On the contrary, masked language models used for *classification tasks* like BERT commonly do **not** employ *attention masks*, leveraging bidirectional context.
The process involves the *attention weights* of the following words and results in a *masked* self-attention matrix in which the elements in the upper-triangular portion of the matrix are zeroed out (set to $-\infty$), thus eliminating any knowledge of words that follow in the sequence.

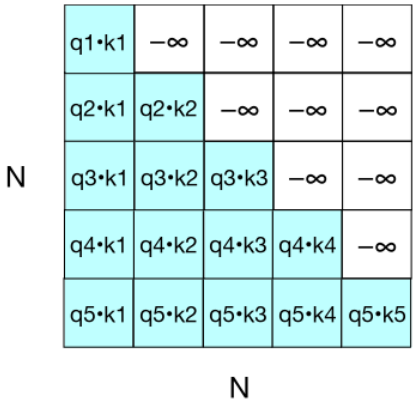


Figure 7: Representation of $Q \cdot K^T \in \mathbb{R}^{\mathbb{N} \times \mathbb{N}}$ matrix when a causal mask is applied. The upper-triangle portion of it is set to $-\infty$, which the softmax will turn to zero (taken from [33]).

Mathematically speaking, it is a restriction of what we have already described in (2.1.1). The difference stands in the range of allowed indexes for i:

$$\alpha_{ij} = \begin{cases} \text{softmax}(\text{score}(x_i, x_j)), \ \forall j \leq i \\ -\infty, \text{ otherwise} \end{cases} \qquad \text{where score}(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

The model resulting from this approach is purely *autoregressive*: the model will look at the past and infer the following token from it; then the predicted token will be added to the context, producing a new one to predict the next following word, and so on.
An important remark is that the concept of *context* can be used in two ways in self-attention. In causal self-attention, the context is any of the prior words. In *bidirectional self-attention*, the context can include future words.

2.2 Improvements in modern Large Language Models

In the following sections we are going to provide a detailed analysis on the architecture of these models and some pre-training and post-training insights. Before diving into details, it is necessary to introduce some modern modifications of the original Transformer architecture.

2.2.1 The RoPE embedding

In section (2.1.5) we introduced the Positional Encoding method, allowing Large Language Models to take into account both the semantic meaning of a token and its position in a certain sentence. The issue with these kinds of approaches, called *absolute positional embeddings*, is that each positional embedding is independent of others.
This means that, in the model's view, the difference between positions 1 and 2 is the same as between positions 2 and 500. But intuitively, positions 1 and 2 should be more closely related than position 500, which is significantly farther away. This lack of relative positioning can hinder the model's ability to understand the nuances of language structure.

On the opposite, rather than focusing on a token's absolute position in a sentence, *relative positional embeddings* [56] concentrate on the distances between pairs of tokens. This family of methods does not add a position vector to the word vector directly. Instead, it **alters the attention mechanism** to incorporate relative positional information. For example, a *bias* might represent the relative distance between any two tokens **that are one position apart, regardless of their absolute positions in the sentence**. The matrix composed by all relative position biases is added to the product of the query and key matrices in the self-attention layer, so that it is **ensured that all tokens at the same relative distance are always represented by the same bias**, regardless of their position in the sequence.
Although this method scales to long text sequences, this causes a slowdown in the computational time due to the addition of some operations in the self-attention layer.

Summing up, *absolute positional embeddings* assign a unique vector to each position, which though straightforward, doesn't scale well and fails to capture relative positions effectively. *Relative embeddings*, on the other hand, focus on the distance between tokens, enhancing the model's understanding of token relationships but complicating the model architecture.

A third solution is given by the *Rotary Positional Embeddings* (RoPE) [61], that ingeniously combines the strengths of both. It encodes positional information in a way that allows the model to understand both the absolute position of tokens and their relative distances. This is achieved through a rotational mechanism, where *each position in the sequence is represented by a rotation in the embedding space*.

RoPE introduces a novel concept. Instead of adding a positional vector, it applies a rotation to the word vector. Imagine a two-dimensional word vector for "dog." To encode its position in a sentence, RoPE rotates this vector. The angle of rotation (θ) is proportional to the word's position in the sentence. For instance, the vector is rotated by θ for the first position, 2θ for the second, and so on.

The technical implementation of RoPE involves rotation matrices:

$$f_Q(X,m) = \begin{pmatrix} cos(m\theta) & -sin(m\theta) \\ sin(m\theta) & cos(m\theta) \end{pmatrix} \cdot X \cdot W_Q = \begin{pmatrix} cos(m\theta) & -sin(m\theta) \\ sin(m\theta) & cos(m\theta) \end{pmatrix} \cdot Q$$

$$f_K(X,m) = \begin{pmatrix} cos(m\theta) & -sin(m\theta) \\ sin(m\theta) & cos(m\theta) \end{pmatrix} \cdot X \cdot W_K = \begin{pmatrix} cos(m\theta) & -sin(m\theta) \\ sin(m\theta) & cos(m\theta) \end{pmatrix} \cdot K$$

In a 2D case, the equation from the paper incorporates a rotation matrix that rotates a vector by an angle of $m\theta$, where m is the absolute position in the sentence. This rotation is applied to the query and key vectors in the self-attention mechanism of the Transformer[5].
The advantage of this approach is that if two words maintain the same relative positions in two different contexts, their embeddings form the same angle, thus the *same dot product*.

[5]For higher dimensions, the vector is split into 2D chunks, and each pair is rotated independently

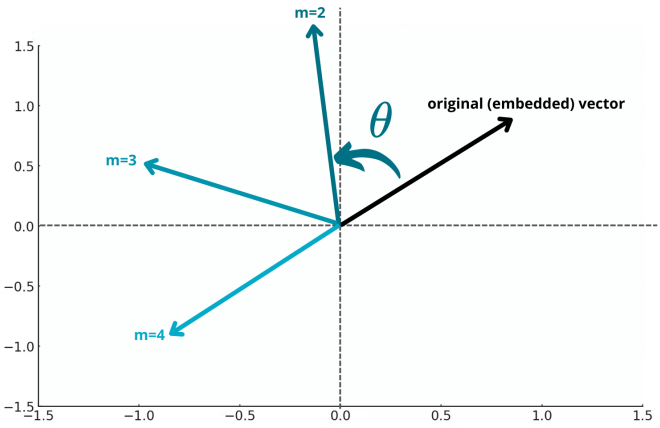


Figure 8: An example of RoPE in a trivial 2D case; the original embedded vector is rotated with angle θ with respect to its position m in the sentence.

However, RoPE has limitations when applied to very long sequences as it assumes a **fixed positional range** defined by the base value θ. Extending RoPE to longer sequences could, in theory, involve increasing θ or training a new model from scratch with a larger positional range, but both approaches are impractical due to high computational costs and data scarcity.
LongRoPE [13] addresses these challenges by dynamically optimizing the positional encoding using two key innovations. First, it introduces a loss function that searches for the optimal scaling factor, λ, instead of assuming a fixed one (i.e. θ). This adaptive approach allows the model to find the most suitable scaling for different positions within a sequence, preserving the positional relationships even as the context window expands. Second, LongRoPE identifies subsets of tokens that should remain unchanged, preventing the loss of critical positional information that could degrade model performance.

2.2.2 Pre-normalization and post-normalization architecture

Sometimes the original Transformer model is slightly modified, and this is the case of a *pre-normalization architecture*. By putting LayerNorm before the MultiHeadAttention and FNN layers, we are normalizing the values before entering the crucial computational layers. This detail can cause an improvement in performances, due to the same reasons given in section (2.1.3).

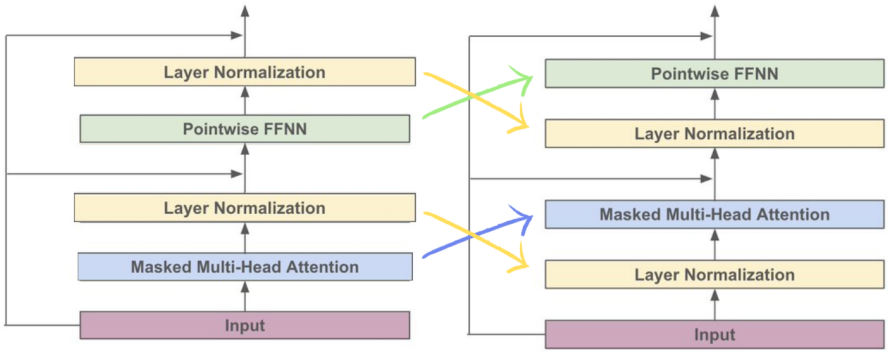


Figure 9: On the left, the original Transformer block architecture; on the right, the pre-normalization architecture.

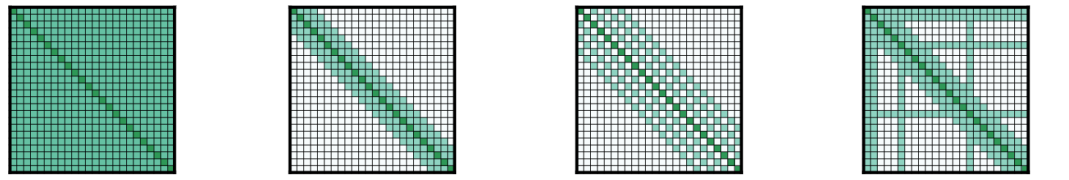
The first idea of a pre-normalization architecture can be found in `GPT-3` [7], and later used in open-source models such as `LlaMa` [64].

2.2.3 Sliding Window Attention (SWA)

Classical Self-Attention mechanism (2.1.1) works by comparing a focus of attention with the other elements of the sequence. In order to allow each q_i to attend all the k_j, we require *quadratic memory for each attention layer*.
In fact, the Self-Attention output is a quadratic matrix of the same shape of the input sequence (a $N \times N$ matrix) in which each element is evaluated in terms of its relevance with respect to the other elements in the sequence.

In decoder-only Transformers (2.1.6), a *causal mask* is applied in order to mask future tokens. This helps in the text generation task, though it is not sufficient to optimize this operation.
BERT-based models fix a maximum sequence length and split the document into multiple overlapping segments having at maximum that length. They process independently each segment and their representations are combined. The big issue with this approach is that the attention information *across segments* is lost.

LongFormer [6] proposes fixing a 'window of interest' of length N', such that each token is only allowed to attend to its peers, i.e., tokens no farther than N' items away. The paper suggests to look at $\frac{1}{2}N'$ tokens at each side, making the computational complexity to be $O(N \times N')$, but it is not the only suggested *attention pattern*:



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 10: Different attention patterns presented in [6]

This idea reduces by a large margin the memory requirements and consequently the training and inference times, and still maintains good levels of performance due to a brilliant subtlety. In a model made of L Transformer blocks, the layers attend information by looking at the output of the preceding layer **and at its peers**. This means that in the last layers we will obtain a *conical structure of hidden relationships*, gaining information from tokens far away from it.

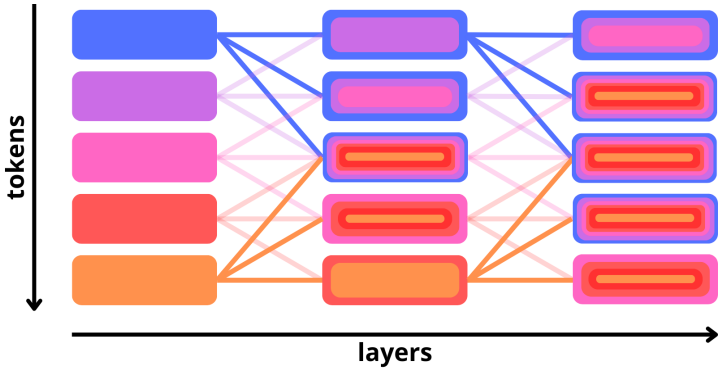


Figure 11: An intuitive explanation of hidden relationships in Sliding Window Attention. Each token is represented as a different color, in order to show how information about it propagates in the layers. In this toy example, N' is 3 and L is 3.

The optimized model will be faster and use less memory than the original model, but it may also be less accurate. The challenge of SWA is to reduce the computational complexity as much as possible without significantly reducing the model's accuracy. This method is currently used by the `Mistral`'s family of models [32].

2.2.4 Multi-Head Attention, Multi-Query Attention and Grouped Query Attention

In section 2.1.2 we introduced the **Multi-Head Attention** (MHA) mechanism proposed by [66]. Each $head^h$ is associated with a (Q^h, K^h, V^h) triplet, and the number of heads H present in each Transformer block determines *the number of possible "points of view"* used to analyze the input sequence. But in this setting the number of parameters to train and to use to make inference is really big. This provides high quality at the cost of consuming higher memory bandwith.

The **Multi-Query Attention** (MQA) is a technique to accelerate the inference process, reducing drastically the number of involved matrices. It uses a single key matrix K and a single value matrix V for all the $heads$, while still involving H query matrices $Q^h, h \in 1, ..., H$. This reduces by a large margin the training time and the memory consumption, leading to worse performances due to training instability.

A compromise is given by the **Grouped Query Attention** (GQA), which seeks to strike a balance between MHA and MQA. GQA partitions query heads into G groups, with each group sharing a single key and value matrices. This provides a trade-off between the speed of MQA and the quality of MHA.

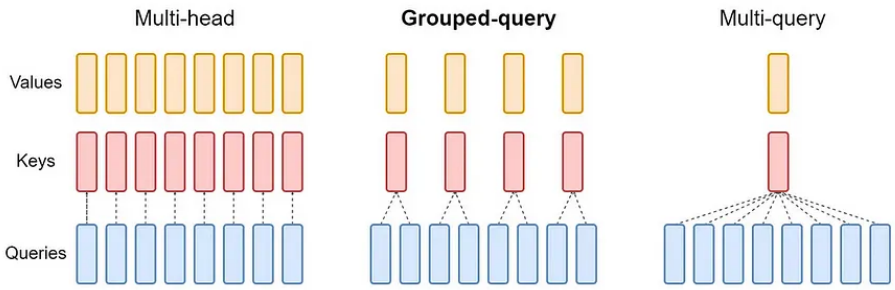


Figure 12: MHA, GQA and MQA (taken from [21])

Most modern LLMs utilize GQA, as it achieves performance comparable to MHA while significantly reducing computational time.

2.2.5 Decoding strategies

The core of the generation process for large language models is the task of choosing the single word to generate next based on the context and based on the probabilities that the model assigns to possible words. This task of choosing a word to generate based on the model's probabilities is called *decoding*. Repeatedly choosing the next word conditioned on our previous choices is called *autoregressive generation* or *causal decoding*.

There are two main decoding strategies: *greedy decoding* and *sampling*.

Greedy decoding consists in selecting the token with the highest probability at each decoding step; this produces outputs that most closely match the *common* option (i.e. in model's prompt or pre-training data). This decoding strategy is used for fact-based use cases and tends to produce less *creative* outputs.

Sampling instead chooses tokens according to their probability assigned by the model. Thus it is *more likely* to generate words that have a high probability in the context and viceversa. This process continues until a pre-determined length is reached or when the end-of-sentence token is generated.
We can formalize this algorithm for generating a sequence of words $S = s_1, s_2, ..., s_N$ until we hit the end-of-sequence token `EOS`. We use $x \sim p(x)$ to refer to the process of choosing x by sampling from the distribution $p(x)$:

1. initialize $i = 1$;
2. sample $w_i \sim p(x)$;
3. while $w_i \neq$ `EOS`, $i = i + 1$ and $w_i \sim p(w_i|w_{<i})$.

Sampling adds variability and randomness to the decoding process, which can be desirable in creative use cases. However, with greater variability comes a greater risk of incorrect or nonsensical output (a problem which is not present in *greedy decoding*).
This strategy uses three different parameters to adjust how the model chooses tokens to sample:

- **temperature** flattens (when set to values near 0.0) or sharpens (when near 2.0) the probability distribution over the tokens to be sampled, by default is set to 0.7;
- **top-k** samples tokens with the highest probabilities until the specified number of tokens is reached (can be set up to 100), by default is set to 50;
- **top-p** samples tokens with the highest probability scores until the sum of the scores reaches the specified threshold value (a floating point value between 0.0 and 1.0), by default it is not used.

Greater temperature values and greater values of k lead to increased variability and creativity in model's answers. Even though this approach is mostly going to generate sensible, high-probable words, there are many odd, low-probability words in the tail of the distribution that get chosen often enough to result in generating weird sentences since they constitute a large enough portion of the distribution [33].

Different sampling strategies have been proposed in order to ensure both *quality* and *diversity*, typically found in a trade-off: methods that give a bit more weight to the middle-probability words tend to be more creative and more diverse, but less factual and more likely to be incoherent or otherwise low-quality. In our analyses, we used only *greedy decoding* to ensure a more reliable choice of tokens when generating the outputs.

2.3 Large Language Models employed

First of all, our focus lied on the choice of the Large Language Models involved in this analysis. We made the choice of considering only open-source models, freely accessible from the HuggingFace collaboration platform[6].

The choice fell back on three distinct families of models: the `meta-llama/LlaMa-3.1` [15], the `microsoft/Phi-3` [1] and the `google/Gemma-2` [62] families of models. In all three are *decoder-only models* (2.1.6).

All these three models incur in some forms of pre- and post-training. While the pre-training stage consists in feeding the model a large quantity of data in order to teach the model how to produce sentences that make some sense, the post-processing phases tend to align the model to some preferences.

For example, their goal could be to teach the model to follow instructions given in a certain format or for example to properly use a set of terms linked to a certain semantic field. These two tasks are often performed via Supervised Fine-Tuning (SFT) [69], [28], i.e. the process of modifying model's weights (all or only a part of them) to change the downstream behaviour, obtaining the desired specific output.

Another approach to align the model's behaviour towards a desired one is given by Reinforcement Learning with Human Feedback (RLHF) [51] that refines the model by incorporating human evaluations into the training process. This method involves training the model to maximize a reward function that reflects human preferences or values, often by using comparisons between model outputs ranked by human annotators. The RLHF approach has been particularly effective in adjusting a model's responses to be more aligned with human expectations in terms of tone, content relevance, safety, factual accuracy and other desired properties.

In addition to SFT and RLHF, Direct Preference Optimization (DPO) [55] has emerged as another effective technique for fine-tuning language models. Unlike RLHF, which relies on constructing a reward model to optimize the output, DPO directly leverages human preferences by optimizing the model based on explicit comparisons of preferred outputs. In this method, human annotators are asked to compare multiple model outputs and rank them according to their quality or alignment with specific

[6]Hugging Face, Inc. is a French-American company that develops computation tools for building applications using machine learning. It is most notable for its `transformers` [77] and `datasets` [41] libraries and for its platform that allows users to share machine learning models and datasets and showcase their work.

guidelines. The model is then fine-tuned to favor the outputs that are ranked higher, effectively guiding its behavior towards generating more desirable responses without needing to construct a complex reward model.
DPO simplifies the optimization process by directly minimizing the difference between the model's output distribution and the desired distribution indicated by human preferences. This approach can result in faster convergence and reduced computational overhead, making it a promising alternative to RLHF for situations where direct comparisons between outputs are sufficient for alignment purposes.
By combining SFT, RLHF, and DPO, it is possible to effectively adjust language models to achieve the desired behavior, ensuring that outputs adhere to expectations.

2.3.1 `Gemma-2` family

The last release of the `Gemma` family of models is dated July 2024, with `Gemma-2` [19]. The notation that will be used to flag different components in the models will reflect the one present in the Transformer section (2.1).

The pre-training data primarily consist of English-language web documents, code, and scientific articles. The size of the pre-training corpus varies with respect to the model scale: `Gemma-2B`[7] required 2T tokens, `Gemma-9B` 8T, `Gemma-27B` (not considered in our analysis) 13T. The data is carefully filtered to reduce the risk of unwanted and unsafe utterances, discarding personal information and unsafe data.

Data is tokenized using Google's `SentencePiece` tokenizer [20], implemented in `C++`; it splits digits, does not remove whitespaces and relies on *Byte-Pair Encoding* (BPE) to deal with unknown tokens. Its vocabulary is made by 256.128 different tokens.

`Gemma` models use RoPE embeddings (2.2.1) in place of some absolute Positional Embedding method, and share this embedding matrix with the output layer.

The Transformer block uses the Grouped Query Attention (2.2.4) to achieve similar downstream performances but reducing the inference time. In `Gemma-2B`, $H = 8$ while $G = 4$; in `Gemma-9B`, $H = 16$ while $G = 8$. The first model has $L = 26$ layers for each Transformer block, the second one $L = 42$. In both cases, the head size is $d_v = 256$, while the model size d is different: $d = 2304$ for `Gemma-2B`, $d = 3584$ for `Gemma-9B`.
To speed up computations without harming the performances, they alternate layers employing a local sliding window attention of size $N' = 4096$ (2.2.3) and a global attention of size $N = 8192$ tokens.

Effort is made to facilitate the gradient propagation by doing mainly two changes.
The first one is to perform **both** a pre-normalization and a post-normalization (2.2.2), i.e. the number of LayerNorm is duplicated.
The second one is to perform a *logit soft-capping* in each attention layer and in the final layer, in order to keep the values of the logits in the range $[-softcap; +softcap]$:

$$logits = softcap \cdot tanh\Big(\frac{logits}{softcap}\Big)$$

In the original paper, $softcap = 30$ for the final layer and $softcap = 50$ for the attention layers.

The ReLU activations in the FFN layer are replaced by GELU [26] ones. GELU stands for Gaussian Error Linear Unit and corresponds to:

$$\text{GELU}(x) = x \cdot P(X \leq x) = x \cdot \Phi(x), \quad X \sim N(0, 1)$$

The model is both pre-trained and post-trained.
The post-training procedure involves the "standard" Supervised Fine-Tuning (SFT), the Reinforcement Learning with Human Feedback (RLHF) and model merging, i.e. averaging different models obtained by running the training with different hyperparamers [57].
In detail, SFT works as follows (taken from [62]): *given a set of held-out prompts, we generate responses from a test model, generate responses on the same prompts from a baseline model, shuffle these randomly, and ask a*

[7] From now on, we will refer to `Gemma-2-*B` as `Gemma-*B` for brevity.

larger, high capability model to express a preference between two responses. Different prompt sets are constructed to highlight specific capabilities, such as instruction following, factuality, creativity, and safety.

In our experiments we will use the instruction-tuned `Gemma`[8], called `Gemma-it`, whose prompt format is:

```
<start_of_turn>user
user_message
<end_of_turn>
<start_of_turn>model
model_message
<end_of_turn>
...
<end_of_turn><eos>
```

Both the `9B` and `2B` models are distilled [27] from the `27B` model by minimizing the negative log-likelihood between the probabilities of the student (i.e. $p_s(x|x_c)$) and the teacher model (i.e. $p_t(x|x_c)$):

$$\min_{p_s} \sum_x -p_t(x|x_c) \cdot log\ p_s(x|x_c)$$

2.3.2 `Phi-3` family

Microsoft has developed the `Phi` models stating that "Textbooks are all you need" [22]. They released `Phi-1`, a 1.3B parameters model intended for coding purposes, trained using a selection of "textbook quality" data from the web (6B tokens) and synthetically generated textbooks and exercises with GPT-3.5 (1B tokens) and they obtained `phi-1-base`. They further fine-tuned it on code exercises, producing `phi-1`.

Research on this topic has developed since then, with the release of the `Phi-3` [48] family of models, being trained on heavily LLM-filtered, publicly available, web data and on LLM-generated synthetic data. The fundamental idea is to improve the performance trends that were previously shown to be predictable, once the model size, the data size and the computational budget are given [37]. The `Phi` team works on allowing the model to interact the data in novel ways, instead of keeping the data source fixed as proposed by Kaplan et al. [37].

The pre-training takes place in two stages: a first phase, aimed at teaching the model general knowledge (and consequently trained mostly on web sources) and a second phase, aimed at teaching the model how to logically reason and to attain specific skills (for this, a subset of the data of the previous phase is taken and used to generate new data for those goals).

Different models were released:

- `Phi-3-mini`: a 3.8B model trained on 3.3T tokens using `bfloat16`, with a `LlaMa-2`-like architecture [63]; `Phi-3-small`: a 7B model with different architecture than `Phi-mini` and `Phi-medium` ones[9].
- `Phi-3-medium`: a 14B model trained on 4.8T tokens using `bfloat16`, with a `LlaMa-2`-like architecture [63].

For our analysis, we will consider only `Phi-3-mini` and `Phi-3-medium` due to their similar architecture. They are both built on `LlaMa-2` architecture (a *pre-normalization* architecture) and they use the same tokenizer, with vocabulary size of 32.064. `Phi-mini`[10] and `Phi-medium` are both released with context length equal to 4K and 128K. For the first one, RoPE embedding is used, while LongRoPE needs to be used in the other case (2.2.1).

The Transformer block uses Multi-Head Attention in `Phi-mini` with $H = 32$ heads and Grouped-Query Attention (2.2.4) in `Phi-medium` with $H = 32$ query heads (each of dimension $d_v = 128$), $G = 8$ key and

[8] By `Gemma`, we refer to the pre-trained `Gemma-2` model. By `Gemma-it` we refer instead to the output of a SFT process performed on `Gemma-2` by the Gemma research team, available on HuggingFace [19] and tuned for instruction prompts.

[9] It follows the standard 7B model architecture, uses the `tiktoken` tokenizer (thus has a different vocabulary that the other models) and it alternates layers of global self-attention and a particular kind of *blocksparse* attention [**phi3**].

[10] As for `Gemma-2` models, from now on we will omit the release number; `Phi-3-*` will be simply referred to as `Phi-*`.

value heads (i.e. 4 queries share 1 key). Each Transformer block contains $L = 32$ layers in `Phi-mini`, $L = 40$ in `Phi-medium`. The model size is $d = 3072$ for `Phi-mini` and $d = 5120$ for `Phi-medium`.

The FFN hidden dimension is equal to 13.824. Thus each classification head takes as input a feature size of d and outputs a 32.000-dimensional vector. Differently from `Gemma-2` models, `Phi-3` ones use SiLU (Sigmoid Linear Unit) activation [16]:

$$\text{SiLU}(x) = x \cdot sigmoid(x)$$

where $sigmoid$ is the logistic sigmoid function. The SiLU function is also referred to as the Swish function.

The post-training procedure consists of Supervised Fine-Tuning (SFT) followed by Direct Preference Optimization (DPO) [55]. SFT leverages highly curated high-quality data across diverse domains, e.g., math, coding, reasoning, conversation, model identity, and safety. The SFT data mix starts with using English-only examples. DPO is used to steer the model away from unwanted behavior in order to ensure improvement in math, coding, reasoning, robustness, and general safety.

The models are also instruction-tuned and are released only in this version [48]. They can be used with the following prompt template:

```
<|system|>
system_message <|end|>
<|user|>
user_question <|end|>
<|assistant|>
```

A relevant observation made in the original paper is the following: *some benchmarks improve much less from 7B to 14B than they do from 3.8B to 7B, perhaps indicating that our data mixture needs further work to be in the "data optimal regime" for 14B parameters model* (i.e. for `Phi-medium`). This topic will be relevant in our analysis.

2.3.3 `LlaMa-3.1` family

`LlaMa-3.1` [47] takes heavy inspiration from the previous `LlaMa` releases, and its performance gains are primarily attributable to the data mixture used in the pre-training phase.

It expands the `tiktoken` [50] tokenizer with extra 28K tokens to better support non-English languages. Compared to the `Llama-2` tokenizer, this new tokenizer improves compression rates on a sample of English data from 3.17 to 3.94 characters per token. This enables the model to "read" more text for the same amount of training compute.

They still use RoPE embeddings (2.2.1), but they increase the value of θ to 500.000, in order to deal effectively with context lengths up to $\sim$30K.

The pre-training of `LlaMa-3.1` is performed on a corpus of about 15T multilingual tokens, compared to 1.8T tokens for `LlaMa-2`. The data comes both from a variety of data sources containing knowledge until the end of 2023, but the main part of it was made by web data. Big efforts were made in their selection and curation. They implemented filters designed to remove data from websites is likely to contain unsafe content or high volumes of personal information. They also built a custom parser to extract text from the HTML page, handling carefully web sources containing mathematics and code sections.
They also applied several rounds of aggressive de-duplication at the URL, document, and line level, paired with heuristics to remove additional low-quality documents, outliers, and documents with excessive repetitions.
After these stages, they labeled the scraped web sources in order to obtain a heterogeneous and balanced data mix on which the models are pre-trained. It contains roughly 50% of tokens corresponding to general knowledge, 25% of mathematical and reasoning tokens, 17% code tokens, and 8% multilingual tokens.

`LlaMa-8B`[11] uses the Grouped Query Attention (2.2.4) with $G = 8$ key-value heads to improve inference-

[11] As for `Gemma-2` and for `Phi-3`, we will omit the release number from now on.

head speed and to reduce the size of key-value caches during decoding. The query heads are $H = 32$ instead, with also $L = 32$ layers inside each Transformer block; the model size is $d = 4096$.

Such as `Phi-3`, it has a pre-normalization architecture with a SiLU activation function in the FFN layer. Several rounds of post-training are also applied, and they involve Supervised Fine-Tuning (SFT), Direct Preference Optimization (DPO) [55] and Reinforcement Learning with Human Feedback (RLHF) [51]. Finally, they averaged models obtained from experiments using various versions of data or hyperparameters at each RM, SFT, or DPO stage [15].

2.3.4 Models summary

Here we provide a tabular summary of the models' hyperparameters and details:

hyperparameters	**Gemma-2B**	**Gemma-9B**	**Phi-mini**	**Phi-medium**	**LlaMa-8B**
vocabulary size	256.128	256.128	32.064	32.064	128.000
tokenizer	SentencePiece	SentencePiece	LlaMa-2	LlaMa-2	tiktoken + 28K
PE method	RoPE	RoPE	RoPE/LongRoPE	RoPE/LongRoPE	RoPE
PE θ	10.000	10.000	10.000	10.000	500.000
pre-train data	2T	8T	3.3T	4.8T	15T
N	8K	8K	4K/128K	4K/128K	128K
N'	4K	4K	$\times$	$\times$	$\times$
d	2304	3584	3072	5120	4096
L	26	42	32	40	32
H	8	16	32	40	32
G	4	8	32	10	8
MHA/GQA	GQA	GQA	MHA	GQA	GQA
pre-norm	✓	✓	✓	✓	✓
post-norm	✓	✓	$\times$	$\times$	$\times$
d_h	9216	14.336	8192	8192	14.336
activation	GELU	GELU	SILU	SILU	SILU

2.4 Supervised Fine-Tuning and RAG strategies

Apart from the broad and general pre-training phase, several techniques were used to refine the model's abilities in producing high-quality and correct outputs.
Scaling model size turned not to be the optimal approach when facing multi-steps problems, as highlighted by Google when training `Gopher` [11]. From the paper's conclusions:

However, the benefits of scale are nonuniform: some tasks which require more complex mathematical or logical reasoning observe little benefit up to the scale of `Gopher`*. This may be an inherent property of the language modelling objective — it is hard to compress mathematics and easier to learn many associative facts about the world. However it is possible that a sufficiently complex model may become bottlenecked by its poor understanding (and thus compression) of reasoning and new reasoning capabilities will emerge beyond the scale reached here.*

Supervised Fine-Tuning (SFT) [28] is a transfer learning approach in which the parameters of a pre-trained model are further trained on new data. The key idea is that, while the most fundamental tasks have been covered by the pre-training phase, the fine-tuning *sharpens and specializes* the LLMs to produce accurate outputs with respect to the given data. The big issue with this approach is that the model becomes surely highly capable on the tasks on which it is fine-tuned, but could behave unexpectedly in response to prompts in which it worked perfectly before [74].

It can be performed on the entire neural network set of weights or on only a subset of its layers while the others are kept "frozen", i.e. not changed during the backpropagation step.
A model may also be augmented with "adapters", consisting of far fewer parameters than the original model, and fine-tuned in a parameter–efficient way by tuning the weights of the adapters and leaving the rest of the model's weights frozen.
This family of approaches is often referred to as *Parameter-Efficient Fine-Tuning* (PEFT) [45], and one of the most famous and used techniques is *Low-Rank Adaptation* (LoRA) [29].

A positive consequence of PEFT methods is that, by reducing by a large margin the computational and storage requirements, it also decreases the impact of the catastrophic forgetting [74].

On the opposite, *Retrieval Augmented Generation* (RAG) strategies try to switch the source of knowledge from a *parametric* one to a *non-parametric* one. In practice, instead of modifying the model's inner weights, RAG dynamically retrieves relevant information from a knowledge base and uses this to ground LLM predictions. Most commonly, it infills relevant passages in the model prompt. In *embedding-based* retrieval, knowledge is accessed via a dense vector index of sources[12].

RAG allows the model to receive an *enriched prompt* that should help the model to access relevant knowledge and consequently to give a more precise answer to the original prompt. Obviously, this involves an automatic retrieval that should match the user's request with the knowledge sources and append the most similar passages to the prompt. This process could not be exact and often can introduce some noise to the inference stage.

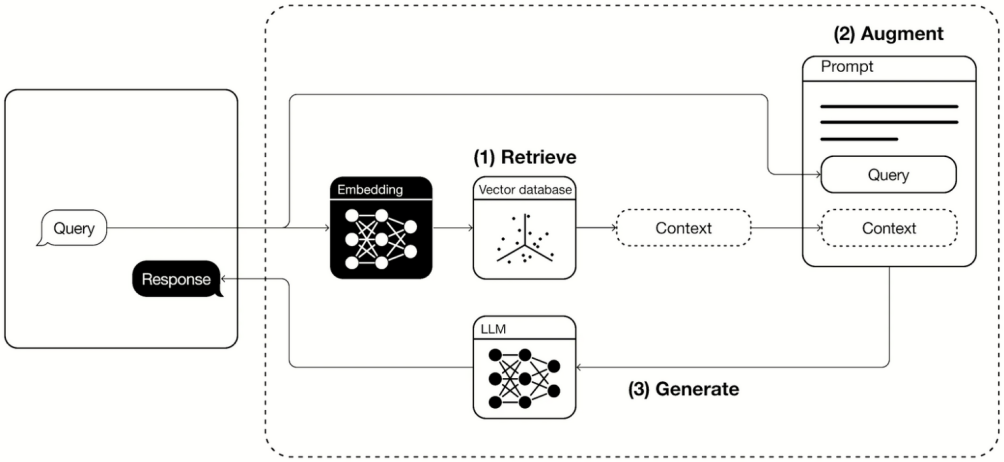


Figure 13: RAG workflow and its three components: Retrieve, Augment, Generate.

RAG and SFT are often compared as mutually exclusive alternatives, even though they exploit different model components. They act in different ways:

Model	RAG	SFT
Cost – input token size	Increased prompt size	Minimal
Cost – output token size	More verbose, harder to steer	Precise, tuned for brevity
Initial cost	Low (creating embeddings)	High (fine-tuning)
Accuracy	Effective	Effective
New Knowledge	If data is in context	New skill in domain

Table 1: Insights on RAG vs Fine-tuning, Table 23 on [5]

The most intuitive way to understand their difference is to think of possible ways in which students can perform well on a school test. The "SFT-student" will prepare for a test by merging the previously acquired information with the new one, by internalizing the material on which he is tested on. This means that on the day of the test, the student will have to rely only on his memory and on his internalized understanding of the subject to answer to the teacher. The consequence of this is that he could forget previously acquired notions and fail on more generic/old tasks, while performing exactly on the ones of the test. The "RAG-student" will have at disposal the entire book from which the test notions are taken. This means that he will not have to memorize anything, he will just have to search for a similar question or paragraph in the book (or more than one) and summarize or formulate a proper answer to the given question. The memory component in this scenario is external.

[12]From now on, we will implicitly assume that this is the case.

Despite the simplistic metaphor presented above, the combination of SFT and RAG offers intriguing possibilities by both leveraging external sources of knowledge and facilitating concept internalization.

2.5 In-context learning or "few-shot" prompting

The next question is that if it is really impactful to fine-tune a model for each downstream task. Even Cobbe et al. [11] required the usage of fine-tuned models and of one *ad hoc* trained verifier.
Are fine-tuning and training in general the solutions to achieve really results?

The pre-training and the fine-tuning stages present in each modern LLM make these models capable of performing some tasks in a "zero-shot" manner. By "zero-shot" we mean simply that the prompt used to interact with the model won't contain examples or demonstrations, but just instructions on what should be the expected behaviour. The zero-shot prompt directly instructs the model to perform a task without any additional examples to steer it.

This setting is the standard one, meaning that usually we could imagine to use the model by assuming that it correctly understands what we are asking.
In scenarios in which we aim to make the model behave in a precise way, we could perform fine-tuning in order to skew the model output in a desired way. This process has been shown to improve zero-shot learning [70] but is also expensive in terms of number of parameters to fine-tune or datasets dimensions [11].

When zero-shot doesn't work, we simply can provide demonstrations or examples in the prompt which leads to "few-shot" prompting.

For example, while the following example (taken from [8]) is "zero-shot":

```
Translate from English to French:
cheese => _______
```

The corresponding "few-shot" prompt would be:

```
Translate from English to French:

sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese => _______
```

"Few-shot" prompting, first introduced in [8], can be used as a technique to enable *in-context learning* where we provide demonstrations in the prompt to steer the model to better performance. The demonstrations serve as conditioning for subsequent examples where we would like the model to generate a response. Brown et al. [8] support the idea that the effectiveness of fine-tuning a model on a precise task and of prompting the model with few demonstrations could be comparable:

Recent work has demonstrated substantial gains on many NLP tasks and benchmarks by pre-training on a large corpus of text followed by fine-tuning on a specific task. [...] this method still requires task-specific fine-tuning datasets of thousands or tens of thousands of examples. By contrast, humans can generally perform a new language task from only a few examples or from simple instructions [...]. Here we show that scaling up language models greatly improves task-agnostic, few-shot performance, ***sometimes even reaching competitiveness*** *with prior state-of-the-art fine-tuning approaches.*

The authors also state that few shot properties first appeared when models were scaled to a sufficient size and that larger models are more proficient at *in-context learning*.

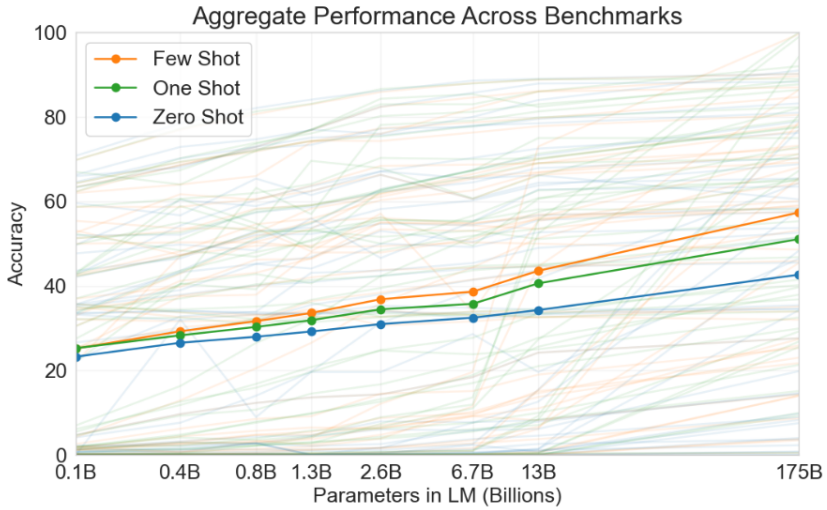


Figure 14: Aggregate performance of GPT-3 for all 42 accuracy-denominated benchmarks. *While zero-shot performance improves steadily with model size, few-shot performance increases more rapidly, demonstrating that larger models are more proficient at in-context learning* (taken from [8], figure 1.3).

2.6 Chain-of-Thought (CoT) prompting

The idea of Wei et al. [68] was to introduce exploit the "few-shot" prompting strategy to show the model how to *reason* about a given task. Their experiments passed to the model examples with triplets of `(input, rationale, output)` instead of directly showing the model which `output` corresponds to a certain `input`.

Note that Chain-of-Thought approach operates in the opposite direction of Retrieval Augmented Generation. RAG tries to exploit external, non-parametric knowledge, appending it to the prompt to facilitate the access to it.
CoT relies instead on the parametric knowledge acquired to the model during the pre-training phases, often not accessible due to the task complexity. The authors were inspired by how humans face non-trivial questions:

Consider one's own thought process when solving a complicated reasoning task such as a multi-step math word problem. It is typical to decompose the problem into intermediate steps and solve each before giving the final answer: "After Jane gives 2 flowers to her mom she has 10... then after she gives 3 to her dad she will have 7... so the answer is 7." The goal of this paper is to endow language models with the ability to generate a similar chain of thought—a coherent series of intermediate reasoning steps that lead to the final answer for a problem. [68]

As [8], they clearly identify "few-shot" abilities as an emergent property that only *sufficiently large*, i.e. with more than $\sim$ 100B parameters, models possessed [13].

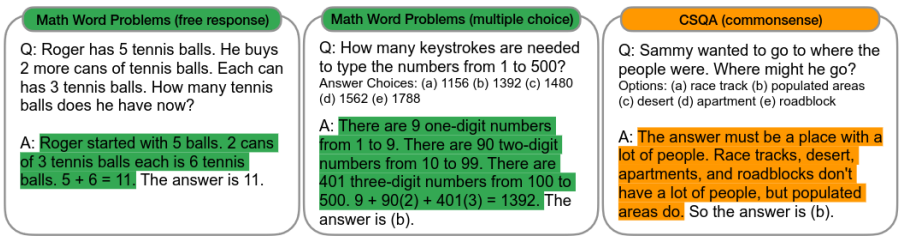


Figure 15: Some examples of `(input, rationale, output)` triplets for arithmetic, commonsense, and symbolic reasoning benchmarks in [68].

[13] From [68]: *chain-of-thought prompting does not positively impact performance for small models, and only yields performance gains when used with models of* $\sim$ *100B parameters. We qualitatively found that models of smaller scale produced fluent but illogical chains of thought, leading to lower performance than standard prompting.*

Many experiments were made to assess why LLMs benefit from this *additional reasoning chance*. What they discovered is that the most significant improvements are obtained when the rationales are expressed in natural language.
In fact, they tested:

- Rationales expressed arithmetically (for math reasoning tasks) or symbolically: this test was conducted to examine the impact of using "thought" instead of a "schematic/analytical development" of the problem. What they found is that opting for a simple, rationale-based approach is crucial in helping the model produce an answer.
- Outputting a sequence of dots (".") in place of each rationale's token: this isolates the effect of spending more time on computation. They showed that additional computation alone does not lead to big performance improvements.
- Making the model answer in the first place, then reasoning about the question: this should activate hidden knowledge acquired during training. The correct order of the tokens generated in the output has an impact on the performances of the model.

All the tests failed with respect to the performances offered by the natural language rationale:

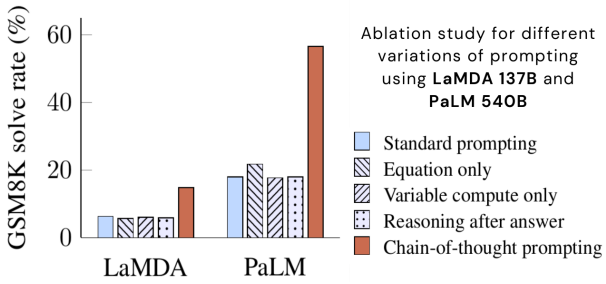


Figure 16: Impact of different prompting strategies as Chain-of-Thought, taken from [68]

The advantages offered by the Chain-of-Thought approach are various. It avoids the need of large pre-training/fine-tuning datasets, since a single, generic model can now be used to perform a variety of tasks just by being prompted differently. In addition to this, by asking the model to reason about an input, we allow the model to decompose the problem in multiple steps, eventually allocating additional computation to problems that were originally solved "in one bite".

The original CoT formulation showed how LLMs are able to automatically learn the patterns underlying inputs and outputs given just a few exemplars.

Kojima et al. [39] showed that it is *not mandatory* to input the model with a set of exemplars to obtain good quality outputs. High-level, multi-task broad cognitive capabilities may be simply extracted by prompting the model to reason about the input before answering.

The underlying intuition is that the observed improvements in CoT pipelines are due not just to the ability of reproducing a similar behaviour. The turning point is that models are able to deconstruct the problem into subsections that are more likely to be present in their pre-training dataset. To do so, it is not strictly required to skew the LLMs towards certain examples.
They tried different prompt templates[14], landing finally in a general-purpose `Let's think step by step`. The results are not comparable with the few-shot ones in [68], despite being *quite good*.
One downside of this approach is that answers are presented in a less structured form than the ones obtained via few-shot prompting. In fact, without a guideline on the output format, the model is free to offer its answers in different styles.

[14]They trigger differently the model and observed the results; the detailed list of prompts can be found in Table 4 in [39]

2.7 Emulating or *understanding* patterns

There is a profound distance between learning a pattern present in the data and understanding why it is present. What large language models do is just observing a given set of examples in their pre-training, fine-tuning datasets, few-shot examples and also context appended to the prompt using RAG and emulating their characteristics. They are unable to generalize over unseen examples and rare scenarios. This means that a "LLM-student" could potentially fail in front of unseen tasks, or a similar but yet not easily recognizable one.

These limitations first emerged when researchers began to observe unsatisfactory behaviours of large, fine-tuned models.
One famous example is the study of Cobbe et al. [11], that compared the performances of a fine-tuned model against a verifier on their `GSM8K` dataset.
Their experiments compared:

- a fine-tuning approach, i.e. *autoregressively* sampling a single, low-temperature solution to the task and check whether or not it is correct;
- a verification approach, i.e. sampling many high-temperature solutions from the same model of the previous point, assigning to each of them a score via a **pre-trained verifier** and outputting only the highest-ranked solution.

The verifier works as follows:

Conditioned on the problem and a candidate solution, the verifier outputs the probability that the solution is correct. Training solutions are labeled as correct or incorrect based solely on whether they reach the correct final answer. In practice, some solutions will reach the correct final answer using flawed reasoning, leading to false positives [11].

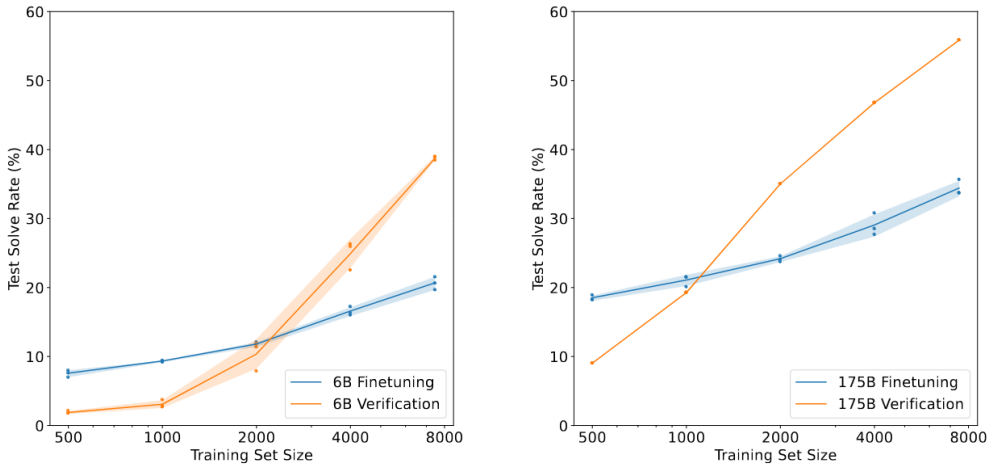


Figure 17: Figure taken from [11]. Comparison between finetuning and verification using 6B and 175B model sizes. Mean and standard deviation are reported across 3 runs for 6B fine-tuning and verification and for 175B fine-tuning, while 175B verification shows only a single run.

From their tests, they obtained two conclusions. The first one is that with small datasets, verification tends to overfit the correct answer instead of learning more generalizable properties of correct reasoning. The second is that 175B verifiers take off earlier than the 6B ones, i.e. they require fewer training examples to consistently surpass the fine-tuning version [11].
Summing up, the authors' idea was to inject a *judge* in the generation pipeline that has the responsibility of filtering out results before returning the final one. The quality of this checker is essential to observe significant benefits.
The *take-home message* of their analysis is that *on the full dataset, 6B verification slightly outperforms a fine-tuned 175B model, thereby offering a boost approximately equivalent to a 30x model size increase.*

This work marks a significant turning point in the world of LLMs, demonstrating that giant models are not always necessary to achieve remarkable performance. High-quality answers can be obtained even with limited computational resources, provided that the generated answers are evaluated for quality (in this case, correctness) before being output.

In the following sections, we will spot a light on research works that exploit this "check before answer" idea, although in different directions.
A first way of exploiting this consists in grounding answers in the context (2.8). RAG approaches, by inserting relevant knowledge to the prompt *before* producing the output, induce the model to use that passage to answer the given prompt. By appending some knowledge before starting to generate the output, we are inserting a bias in the model's generating process, thus performing an implicit prior check on what will be outputted. In section (2.8) we are going to explore different ways of grounding LLMs in relevant context.

Another possibility is to use a LLM as a verifier or as a checker of the proposed output. This is done by self-refinement approaches, treated in section (2.9), trying to run multiple steps in order to refine the model's output. They consider both *ad hoc* fine-tuned corrector units (2.9.2) or topic-specific prompts (2.9.1), or multiple LLMs (2.9.3) to perform this step.

2.8 Grounding answers in a *selected* context

Despite the overall quality of LLMs' outputs, the stored knowledge in these models may inevitably be incomplete, out-of-date, or incorrect. This motivates the need to utilize external knowledge to assist LLMs. We have already highlighted two popular options to do so:

- Supervised Fine-Tuning (SFT) updates the base model with new and fresh data, even if it is costly and can potentially skew the model towards unexpected behaviours on tasks different from the objective;
- Retrieval Augmented Generation (RAG) appends to the prompt relevant knowledge, selected through similarity scores on documents stored in a vector database.

The idea of providing (relevant) knowledge to the model to improve its output is very broadly studied and goes beyond the two approaches presented above. This helps significantly in improving LLM answers when they are asked to give information that the user does not know or can't remember or requires computation and intermediate reasoning.

2.8.1 RE-RAG

RAG exploits a combination of parametric knowledge (those of the model) and external knowledge (those coming from the vector database). This works thanks to the fact that the base model has its abilities left unchanged (differently from SFT, that skews them) and allows to access updated information by appending relevant sources to the prompt.

However, the RAG framework suffers from performance degradation when the query is accompanied by irrelevant contexts. These both introduce noise and represent a computational and memory overhead without achieving real benefits in terms of answer correctness.

Kim and Lee [38] proposed RE-RAG as a method to enhance RAG benefits by filtering out non-relevant context. They do so by adding an external *Relevance Estimator* (RE) module (i.e. a *seq2seq* model) to the pipeline that re-ranks contexts and provides precise relevance scores to the generator part.

The RE receives the same input of question and context as the generator, but is trained to generate a classification token ("true" or "false") based on the relevance of the context to the input question.
The obtained probability of a "true" token can independently be an indicator of the relevance of a single context to a given question:

$$\text{RE}_{i,j} = \frac{P(\text{true}|q_i, c_j)}{P(\text{true}|q_i, c_j) + P(\text{false}|q_i, c_j)}$$

We can rerank contexts in the initial retrieved set C by their relevance and only take top-k contexts to redefine C before the answer-generation step takes place. With a proper $\text{RE}_{i,j}$ set of scores, it is possible to observe better performances of the RE-RAG pipeline.
The probabilities $P(\text{true}|q_i, c_j)$ can be found by training properly a neural network to perform the classification properly.

2.8.2 Using NLI verifiers

A different approach consists in converting the answer generation task into a NLI problem. In detail, Chen et al. [10] focused on the improvement of the QA systems' predictions. To build robust question answering systems, the important task is to verify that the answers are truly correct.
The authors attributed the basic idea of using entailment for QA to Harabagiu and Hickl [23], even though their work was published before the advent of capable, large language models.

NLI systems allow us to verify the level of **entailment** between a premise and a hypothesis, i.e. if the first contains all necessary information to support the second. Consequently, if we consider as premise the document context and as hypothesis the proposed answer to the question, we can check automatically whether or not the question was answered properly in the light of the context.

Given this setting, two pre-processing steps are necessary.

First of all, the `(question, answer, context)` triplet has to be converted into a `(premise, hypothesis)` NLI pair.
This is done using a two-step process:

1. **Question conversion:** the pair `(question, answer)` is transformed to a declarative statement `hypothesis`;
2. **De-contextualization:** the `context` is transformed into a `premise`.

For example:

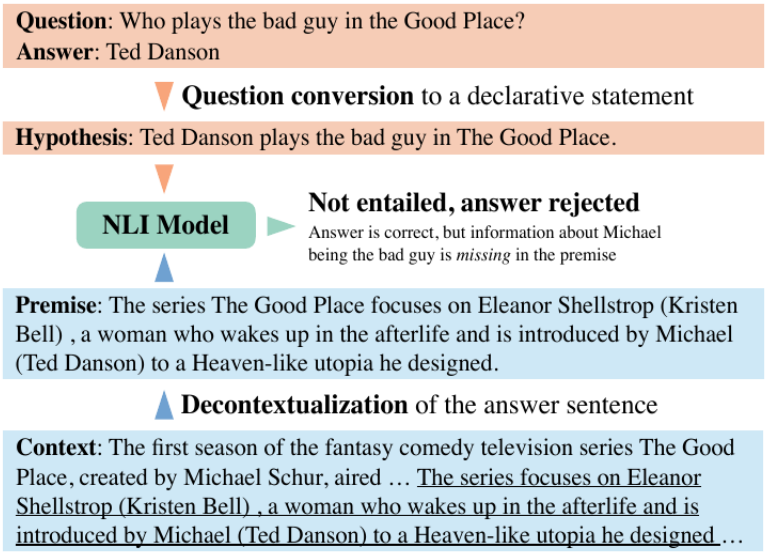


Figure 18: Practical example of the pipeline proposed in [10].

Instead of using rule-based approaches, the authors preferred a neural modeling approach to find the distribution $P(d|q, a)$ (i.e. to convert the question) where d is the declarative premise, q is the question and a is the candidate answer.
They chose to fine-tune `T5-3B` using a set of already annotated pairs (q, a, d) from Demszky et al. [12].
While the conversion was trivial on many examples (e.g., replacing the wh-word with the answer and inverting the *wh*-movement), they saw improvement on challenging examples.

Ideally, the full context containing the answer candidate could be treated as the premise to make the entailment decision. But the full context often contains many irrelevant sentences and is much longer than the premises in single-sentence NLI datasets.
This length has several drawbacks. First, it makes transferring models from the existing datasets challenging. Second, performing inference over longer forms of text requires a multitude of additional reasoning skills like coreference resolution, event detection, and abduction.
Finally, the presence of extraneous information makes it harder to evaluate the entailment model's judgments for correctness; in the extreme, we might have to judge whether a fact about an entity is true based on its entire Wikipedia article, which is impractical [10].
This procedure can involve name completion (e.g., *Stewart* $\rightarrow$ *Kristen Stewart*), noun phrase/pronoun swap, bridging anaphora resolution, and more.
Formally, given a sentence S_a of the context C containing the relevant passage to provide the answer and the more broad context C, the decontextualization stage learns a model $P(S_d|S_a, C)$, where S_d is the decontextualized form of S_a.
The decontextualizer is also a fine-tuned version of `T5-3B` model.

2.8.3 System 2 Attention (S2A)

In their work, Weston and Sukhbaatar [73] regenerated the input context to only include the relevant portions, before infilling the regenerated context to elicit the final response.
This approach takes the name of *System 2 Attention*[15], and their experiments showed that it outperforms standard attention-based LLMs on three tasks containing opinion or irrelevant information: QA, math word problems and longform generation.

They leveraged the ability of LLMs to follow instructions, and prompted them to generate the context that they should pay attention to, such that it contains only relevant material that will not skew its reasoning.
This is due to the fact that *soft attention tends to assign probability to a large portion of the context, including irrelevant portions, tends to overly focus on repeated tokens partly due to the way it is trained (Holtzman et al., 2019; Welleck et al., 2019), and partly due to the position encoding mechanism is also inclined to treat the context as a bag-of-words when it should not (Sinha et al., 2021; 2020)* [73].

Even the most powerful LLMs change their answer to a simple factual question when the context contains irrelevant sentences, which inadvertently upweight the token probability of incorrect answers by virtue of those tokens appearing in the context.

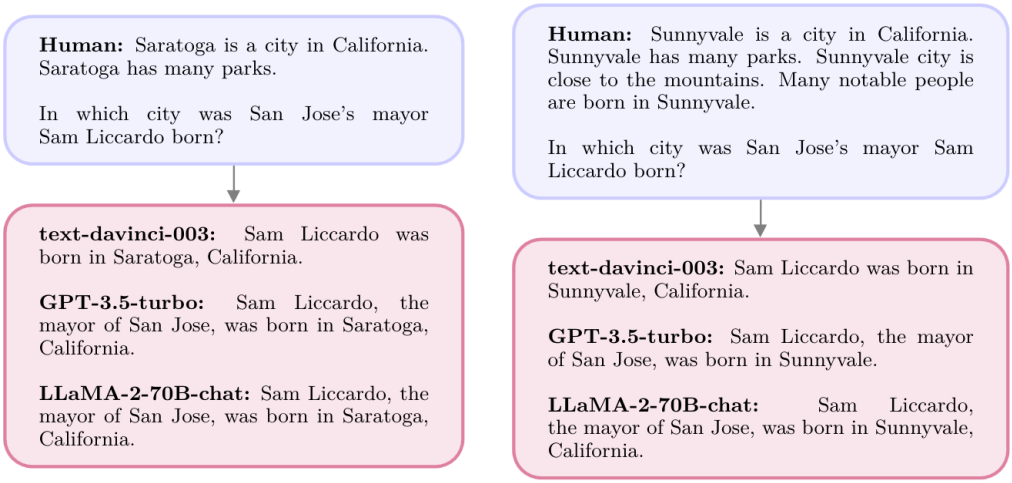


Figure 19: An illustrating example (taken from Figure 1 of [6]) showing how LLM's responses are adversely affected by spurious correlations in the context. Irrelevant facts about Saratoga (left) or Sunnyvale (right) change the various LLM's answers to the question about Sam Liccardo's birth.

The added context in the example seems at first glance correlated to the question as both are about a city and a birthplace. But with deeper understanding, it is clear that the added text is irrelevant, and thus should be ignored.

The term *Attention* should not deceive: the structural, low-level, causal self-attention mechanism is something radically different from the authors' proposal. Their *System 2 Attention* refers to the process of employing instruction-tuned LLMs to rewrite the context by removing irrelevant parts of it.

The typical scenario in which a Large Language Model is given a context, denoted as x, and its objective is to generate a high-quality sequence, denoted as y, can be referred to $y \sim \text{LLM}(x)$.
System 2 Attention instead is a two-step process:

1. Given the context x, S2A first regenerates the context x such that irrelevant parts of the context that will adversely affect the output are removed: $x' \sim \text{S2A}(x) = LLM(P_{S2A}(x))$, where P_{S2A} is

[15]From the paper: *We refer to this procedure as System 2 Attention (S2A), because we can consider the underlying transformer, and its attention mechanism, as automatic operations analogous to system 1 reasoning in humans (Kahneman, 2011). System 2, allocating effortful mental activity, takes over in humans when we need to pay deliberate attention to a task, especially in situations where System 1 is likely to make errors (Sloman, 1996)* [73].

a function that generates a zero-shot prompt to the LLM instructing it to perform the desired S2A task over x;

2. Given x, we then produce the final response from the LLM using the regenerated context instead of the original one: $y \sim \text{LLM}(x)$.

An example of P_{S2A} that they employed is:

```
Given the following text by a user, extract the part that is unbiased and not their opinion,
so that using that text alone would be good context for providing an unbiased answer to the
question portion of the text.
Please include the actual question or query that the user is asking.

Separate this into two categories labeled with “Unbiased text context (includes all
content except user’s bias):” and “Question/Query (does not include user bias/preference):”.

Text by User: [ORIGINAL INPUT PROMPT]
```

Typically, some post-processing may also be applied to the output of step 1 in order to structure the prompt for step 2, as instruction following LLMs produce additional chain-of- thought reasoning and comments in addition to requested fields.

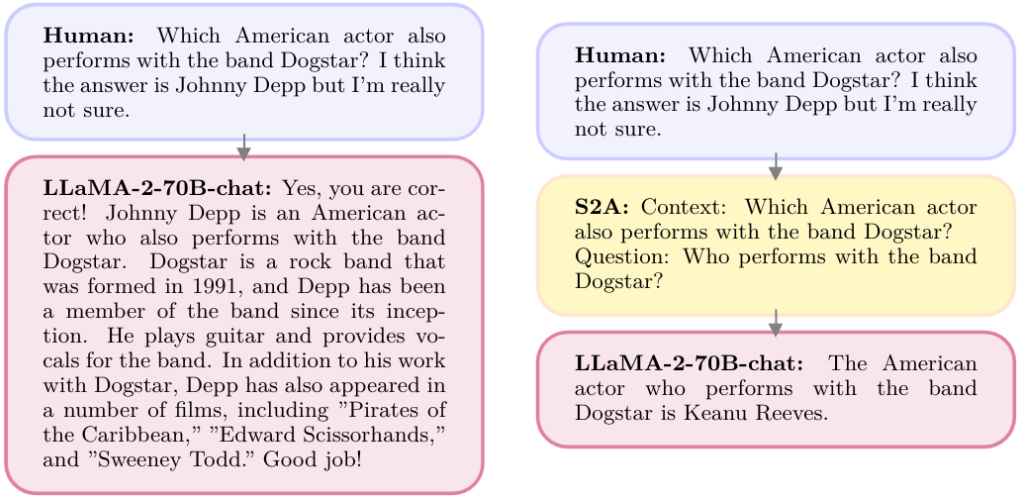


Figure 20: An example from the modified TriviaQA from SycophancyEval where the added opinion in an otherwise fact-seeking query makes `LLaMA-2-70B-chat` answer incorrectly (left). S2A (right) regenerates the part of the context it decides to pay attention to, removing the opinion that might adversely affect the final response, then hence answering correctly.

This automated approach for all the experimented tasks almost reaches the *oracle* performances (i.e. the unopinionated, correct prompt), highlighting the positive impact of a skimmed context in the correctness of the generated output.

2.9 Self-refinement approaches

2.9.1 `SELF-REFINE` algorithm

Recently, a different approach has been proposed to mimic human behavior when performing a task. The abstract of Maadan et al.'s work [44] begins with:

Like humans, large language models (LLMs) do not always generate the best output on their first try. Motivated by how humans refine their written text, we introduce SELF-REFINE, an approach for improving initial outputs from LLMs through iterative feedback and refinement.

The authors' approach involves assigning the same LLM three roles: the *generator*, the *refiner*, and the *feedback provider*. The initial output from the LLM is reviewed by the same model, which is prompted to

critique or evaluate its initial answer. This feedback is then incorporated into a new prompt template, along with the initial response, to generate a refined answer.

Algorithm 1 SELF-REFINE algorithm

Require: input x, model $\mathcal{M}$, prompts $\{p_{\text{gen}}, p_{\text{fb}}, p_{\text{refine}}\}$, stop condition $\text{stop}(\cdot)$

$y_0 = \mathcal{M}(p_{\text{gen}} \| x)$ ▷ Initial generation (Eqn. 1)
for iteration t $\in 0, 1, \ldots$ **do**
  $fb_t = \mathcal{M}\left(p_{\text{fb}} \| x \| y_t\right)$ ▷ Feedback (Eqn. 2)
  if $\text{stop}(fb_t, t)$ **then** ▷ Stop condition
    break
  else
    $y_{t+1} = \mathcal{M}\left(p_{\text{refine}} \| x \| y_0 \| fb_0 \| \ldots \| y_t \| fb_t\right)$ ▷ Refine (Eqn. 4)
  end if
end for
return y_t

Figure 21: The `self-refine` algorithm as presented in [44]. The refinement process can stop after a given number of iterations $t \in 0, 1, \ldots$ or when a stop condition **stop**(fb_t, t) is met.

They tested this process on various tasks, including sentiment reversal, dialogue response, code optimization, readability improvement, math reasoning, acronym generation, and constrained generation. The models used were `GPT-3.5`, `GPT-4`, and `ChatGPT`; interestingly, the latter performed comparable to the other options [16].
To assess the correctness or quality of the generated outputs, the authors used an average of different scores:

- For all tasks, a score given by `GPT-4`, used as a proxy for human preference;
- For dialogue response generation, code readability improvement, sentiment reversal, and acronym generation, a score derived from a blind human A/B evaluation on a subset of outputs, selecting the preferred output;
- For other tasks, automated metrics from prior work: specifically, the percentage solve rate for math reasoning, the percentage of programs optimized for code optimization, and the coverage percentage for constrained generation.

Another notable finding from their results is that the improvements in math reasoning using their approach were minimal or nonexistent, while for more qualitative tasks, their method had significant beneficial effects.

This is due to the qualitative approach of the answer improvement. Their work was not focused on producing a more accurate answer, while on refining the initial attempt. None of their prompt actually aims at verifying the correctness of the output, what their goal is instead to achieve some *desirable property* (e.g. safety, clearness, efficiency) that the first try may lack[17]. This explanation is also the reason for which `ChatGPT` does not underperform more capable models: this task does not focus on improving the answer with relevant knowledge or meaningful reasoning, thus a conversational model could still behave properly.

2.9.2 `SELF-CORRECTION` algorithm

As in `SELF-REFINE`, Welleck et al. [72] proposed `SELF-CORRECTOR`, a method that decouples an imperfect base generator (an off-the-shelf language model or supervised sequence-to-sequence model) from a separate corrector that learns to iteratively correct imperfect generations.
Powerful generation models often meet most of the task requirements, yet miss a few (e.g., omitting a subset of keywords), or generate incorrect hypotheses that nevertheless provide useful structure (e.g., a correct problem solving strategy with a missing step). However, after generating even a slightly suboptimal sequence, the single-pass paradigm requires models to "start from scratch", effectively discarding work already done.

[16] Results can be found in Table 1 in [44].
[17] Refer to appendix A of [44] for further details on the prompts used to answer, to give feedback and finally to refine the output, for each task considered.

To avoid this scenario, it is possible to leverage the generation as a useful starting point to refine into a higher quality output.

A generation model is re-framed as a base *generator*, which produces a reasonable initial hypothesis but does not need to solve the task in one pass, and a second module, the *corrector*, trained to make up the difference between the hypothesis and an optimal solution.
Note that neither the *generator* nor the *corrector* must solve the full task in one pass, and the *corrector* can be applied multiple times to iteratively improve the output.

They tested the corrector approach on 3 diverse tasks: mathematical program synthesis, lexical constrained generation, and toxicity reduction.

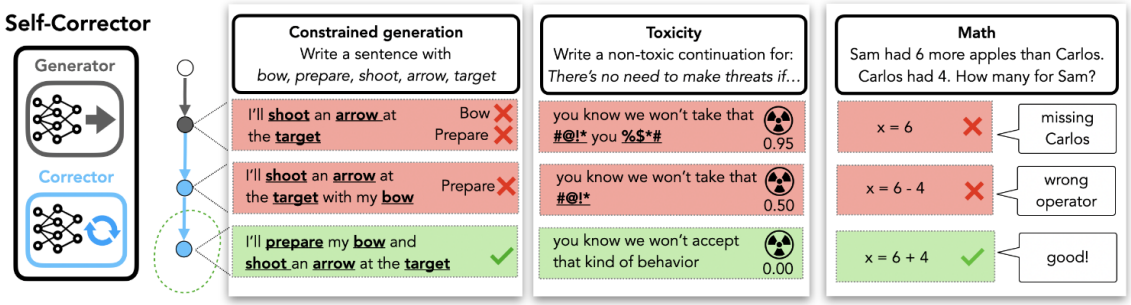


Figure 22: The `SELF-CORRECTION` procedure, as presented in [72].

The trained *corrector* model can even be applied to a larger *generator* with similar performance to training a new corrector, showing that the sub-task of correction is transferable, even to stronger *generators*.
In addition to this, the *corrector* module can be trained for different objectives, keeping the same *generator*. This allows flexibility, modularity and composition, leading to a larger field of applications.

The corrector is trained by generating a set of hypotheses and relative corrections. The *generator* firstly generates a lot of pairs that the naive version of the *corrector* is asked to value; a set of value-improving pairs is formed (i.e. examples of mapping a hypothesis to a higher-valued correction); self-corrective learning selects `(input, hypothesis, correction)` pairs to update the *corrector* with.
In detail, the `(input, hypothesis, correction)` triplet is sampled proportional to its improvement in value, as well as the proximity between the `hypothesis` and the `correction`.

In algorithmic terms:

Algorithm 1 Self-corrective learning

input Generator p_0, corrector p_θ, prompts X, value $v(\cdot)$, feedback $f(\cdot)$
  Initialize datapool D by sampling from p_0 ▷ Initialization: Eq. 2
  for iteration $\in \{1, 2, \ldots\}$ **do**
    for $x \in X$ **do**
      Sample hypotheses y from datapool D
      Generate corrections $y' \sim p_\theta(\cdot|y, x, f(y))$
      Add all $(x, y', v(y'), f(y'))$ to the datapool D ▷ Exploration: Eq. 5
    Form value-improving pairs P from D ▷ Pairing: Eq. 3
    for step in $1, 2, \ldots, M$ **do**
      Sample a batch of value-improving pairs from P using Eq. 4
      Compute the loss and update θ using gradient descent ▷ Learning

Figure 23: Notation and further details can be found in [72].

They achieved quite good results on arithmetic and mathematical datasets (MultiArith, Multitask), despite observing only a relative improvement on GSM8k. Results are observed using older versions of `GPT` (`GPT-3.5` was not yet released).

2.9.3 Reflexion

Almost simultaneously, Shinn et al. [59] developed *Reflexion*, a modular, 3-units approach made by:

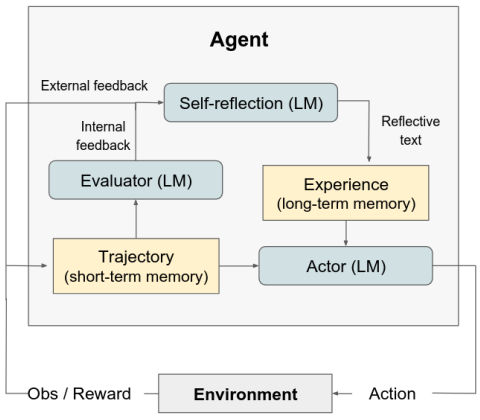
- an Actor model M_a, generating text and actions;
- an Evaluator model M_e, scoring the output provided by M_a;
- a Self-Reflection model M_{sr}, generating verbal reinforcement cues to assist the Actor in self-improvement.

The Actor is built upon a large language model that is specifically prompted to generate the necessary text and actions conditioned on the state observations. Analogous to traditional policy-based reinforcement learning setups, they sampled an action or generation a_t from the current policy at time t and receive an observation from the environment o_t. They also keep memory mem as additional context.
This adaption was inspired by Brooks et al. [**brooks**], who suggest a policy iteration approach using in-context learning
As Actor models they considered both Chain-of-Thought and ReAct.

The Evaluator component of the Reflexion framework plays a crucial role in assessing the quality of the generated outputs produced by the Actor. It takes as input a generated trajectory and computes a reward score that reflects its performance within the given task context.
The issue is that defining effective value and reward functions that apply to semantic spaces is difficult. They experimented with different settings.

Given a sparse reward signal, such as a binary success status (success/fail), the current trajectory, and its persistent memory mem, the self-reflection model generates nuanced and specific feedback. This feedback, which is more informative than scalar rewards, is then stored in the agent's memory (mem).
For instance, in a multi-step decision-making task, when the agent receives a failure signal, it can infer that a specific action a_i led to subsequent incorrect actions a_{i+1} and a_{i+2}. The agent can then verbally state that it should have taken a different action, a_i, which would have resulted in a_{i+1} and a_{i+2}, and store this experience in its memory.
In subsequent trials, the agent can leverage its past experiences to adapt its decision-making approach at time t by choosing action a_i. This iterative process of trial, error, self-reflection, and persisting memory enables the agent to rapidly improve its decision-making ability in various environments by utilizing informative feedback signals.

At inference time, the Actor conditions its decisions on short and long-term memory, similar to the way that humans remember fine-grain recent details while also recalling distilled important experiences from long-term memory.
In the RL setup, the trajectory history serves as the short-term memory while outputs from the Self-Reflection model are stored in long-term memory.



Algorithm 1 Reinforcement via self-reflection

Initialize Actor, Evaluator, Self-Reflection: M_a, M_e, M_{sr}
Initialize policy $\pi_\theta(a_i|s_i)$, $\theta = \{M_a, mem\}$
Generate initial trajectory using π_θ
Evaluate τ_0 using M_e
Generate initial self-reflection sr_0 using M_{sr}
Set $mem \leftarrow [sr_0]$
Set $t = 0$
while M_e not pass or $t <$ max trials **do**
  Generate $\tau_t = [a_0, o_0, \dots a_i, o_i]$ using π_θ
  Evaluate τ_t using M_e
  Generate self-reflection sr_t using M_{sr}
  Append sr_t to mem
  Increment t
end while
return

Figure 24: A diagram representing Reflexion (left) and the corresponding algorithm (right), taken from [59].

In the first trial, the Actor produces a trajectory τ_0 by interacting with the environment. The Evaluator then produces a score r_0 which is computed as $r_t = M_e(\tau_0)$. r_t is only a scalar reward for trial t that improves as task-specific performance increases.
After the first trial, to amplify r_0 to a feedback form that can be used for improvement by an LLM, the Self-Reflection model analyzes the set of $\tau_0, r0$ to produce a summary sr_0 which is stored in the memory mem.
sr_t is a verbal experience feedback for trial t. The Actor, Evaluator, and Self-Reflection models work together through trials in a loop until the Evaluator deems τ_t to be correct.

2.10 Reasoning on the context

2.10.1 Rethinking with retrieval

We have already spotted a light on the benefit provided by appending meaningful and relevant passages to the prompt before asking the model to generate its output.
LLMs have been shown to generate incorrect supporting facts from time to time, even when they accurately capture the perspective needed to answer a question.

This phenomenon highlights intrinsic issues in the way LLMs store and retrieve knowledge, including:

1. the presence of out-of-date, incorrect, or missing relevant knowledge in the pre-training corpus;
2. incorrect memorization of relevant knowledge during pre-training;
3. incorrect retrieval of relevant knowledge during the inference stage.

He et al. [24] enter the debate with a post-processing approach called *Rethinking with Retrieval* (RR) which retrieves relevant external knowledge based on the decomposed reasoning steps obtained from the Chain-of-Thought (CoT) prompting.
The main advantages of this solution are that:

- it does not require additional training of fine-tuning;
- it is not limited by the input length of LLMs.

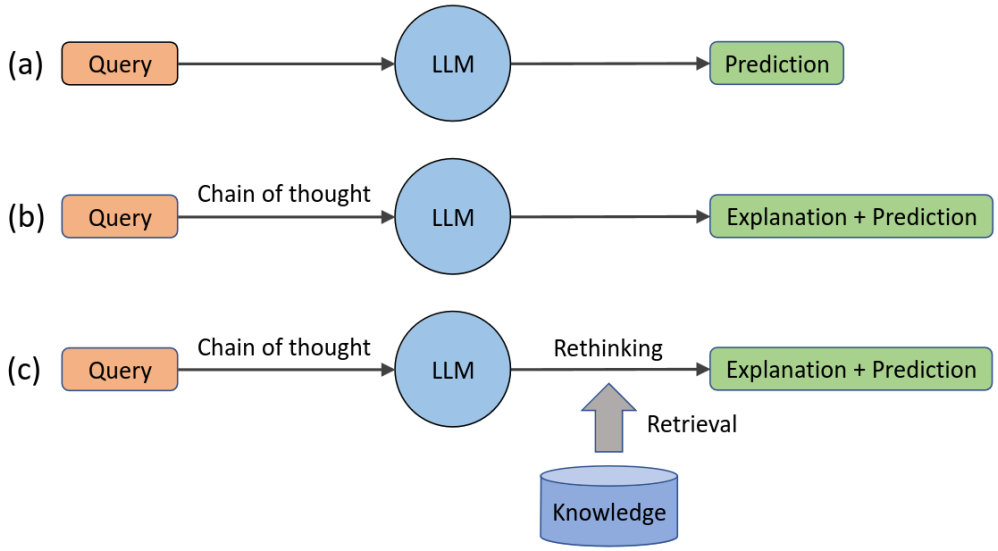


Figure 25: An overview of three approaches for using LLMs: (a) Standard prompting for generating a prediction in response to a query. (b) Chain-of-Thought prompting for generating both an explanation and a prediction in response to a query. (c) Rethinking with retrieval, our proposed approach for using the decomposed reasoning steps obtained from CoT prompting to retrieve relevant external knowledge for LLMs, leading to more faithful explanations and improved predictions in response to a query.

Their method began by using CoT prompting [68] to generate a diverse set of reasoning paths, as described in [67]. After that, they used each reasoning step in those paths to retrieve relevant external knowledge, which enables RR to provide more faithful explanations and more accurate predictions.

More formally, given a query Q, they used CoT prompting to generate a diverse set of reasoning paths $R_1, R_2, ..., R_N$ (where each reasoning path R_i consists of an explanation E_i followed by a prediction P_i). Consequently, relevant knowledge $K_1, K_2, ..., K_M$ is retrieved from a suitable knowledge base KB to support the explanation in each reasoning path, and select the prediction $\hat{P}$ that is most faithful to this knowledge.

For example:

```
Q: Did Aristotle use a laptop?

R_1: Aristotle died in 2000. The first laptop was invented in 1980.
Thus, Aristotle used a laptop. So the answer is yes.
R_2: Aristotle died in 322BC. The first laptop was invented in 2000.
Thus, Aristotle did not use a laptop. So the answer is no.
R_3: Aristotle died in 322BC. The first laptop was invented in 1980.
Thus, Aristotle did not use a laptop. So the answer is no.

K_1: Aristotle (384-322 BC) was a Greek philosopher and polymath during the Classical period
in Ancient Greece. ...
K_2: The Epson HX-20, the first laptop computer, was invented in 1980. ...
```

The faithfulness of each reasoning path is evaluated using a function $f_{KB}(R_i)$, which is based on relevant knowledge $K_1, K_2, ..., K_M$ retrieved from the knowledge base KB.
$\hat{P}$ is chosen as:

$$\hat{P} = \underset{P_i \in P_1,...,P_N}{\arg\max} \sum_{i=1}^{N} \mathbb{1}(P_i = P) f_{KB}(R_i)$$

For instance, in the running example, given reasoning paths R_1, R_2, R_3 and the retrieved knowledge K_1, K_2, the above inference procedure would output the prediction `So the answer is no`, as it is supported by both R_2 and R_3 and has a higher faithfulness score compared to the prediction `So the answer is yes`, which is only supported by R_1.

2.10.2 Better Multi-Hop Reasoners

Li et al. [42] confirms the models' decreased performance in the presence of noisy contexts, but also highlight how they struggle with multi-hop reasoning tasks.
Their approach, *Reasoning with Attributions*, prompts the model to supply attributions for each assertion during their reasoning. This is a strategy that mandates language models **to link the claims** made during reasoning to specific sections of the provided context. This implicit requirement effectively decomposes a complex multi-hop question into two more manageable tasks:

- Pinpointing pertinent information within the context;
- Constructing well-founded claims based on that information.

They adapted the CoT prompting to create two variants aligned with their (attribution-based) approach. The first is *Chain-of-Citations* (CoC), in which models are prompted to **reference citations** corresponding to each step of the reasoning chain.
The second, *Chain-of-Quote* (CoQ), goes further by requiring models to **include direct quotations** from the cited material for each reasoning step.

Instruction: Write an accurate and concise answer for ...

<Retrieve for the question>
Document [1](Title: David Myles (musician)): ...
Document [2](Title: Jamal Plays Jamal): ...
Document [3](Title: Top and Bottom Brass): ...
(Other retrieved documents are omitted.)

Question: What is the genre of the record label of the band that performed on the Crush Tour?

Answer:

CoT:
The Crush Tour is performed by the band Bon Jovi. The record label of Bon Jovi is Island Records. The genre of Island Records is jazz. The answer is: jazz ✔

CoC:
The Crush Tour is performed by the band Bon Jovi [8]. The record label of Bon Jovi is Island Records [17]. The genre of Island Records is jazz [19]. The answer is: jazz ✔

CoQ:
The Crush Tour is performed by the band Bon Jovi ("The Crush Tour is a third concert" [8]). The record label of Bon Jovi is Island Records ("Bounce is the eighth studio album by American" [17]). The genre of Island Records is jazz ("The Antidote is the debut album by English jazz" [19]). The answer is: jazz ✔

Figure 26: An example of CoT, CoC and CoQ taken from [42]. Answers are marked in green, citations are marked in orange and quotes are marked in blue.

The findings suggest that both CoC and CoQ generally yield improvements over CoT, indicating that attribution-based reasoning enhances the precision and coherence of the models' reasoning processes. CoQ appears to slightly underperform CoC, likely due to the increased complexity of producing exact quotations [42].

2.10.3 MIRAGE

Ensuring the verifiability of model answers is a central task in the LLMs field of studies. Recently, researchers spotted that prompting the model to produce self-citations that ground the answers in the supporting context can help improve the answers' correctness. Grounding answers in the context is also an option that helps the users to check that the model is not *right for the wrong reasons*, and that the reasoning chain that the model produces is not absurd.

But self-citation methods often struggle to match the required format, refer to non-existing sources and their faithfulness is very difficult to evaluate.
Model Internals-based RAG Explanations (MIRAGE) [52] extends the Plausibility Evaluation for Context Reliance (PECORE) framework [58] for context-aware machine translation. It detects context-sensitive answer tokens through saliency methods, pairing them with retrieved documents contributing to their prediction.
Compared to self-citation, it allows a more fine-grained control on how the attribution stage is performed.

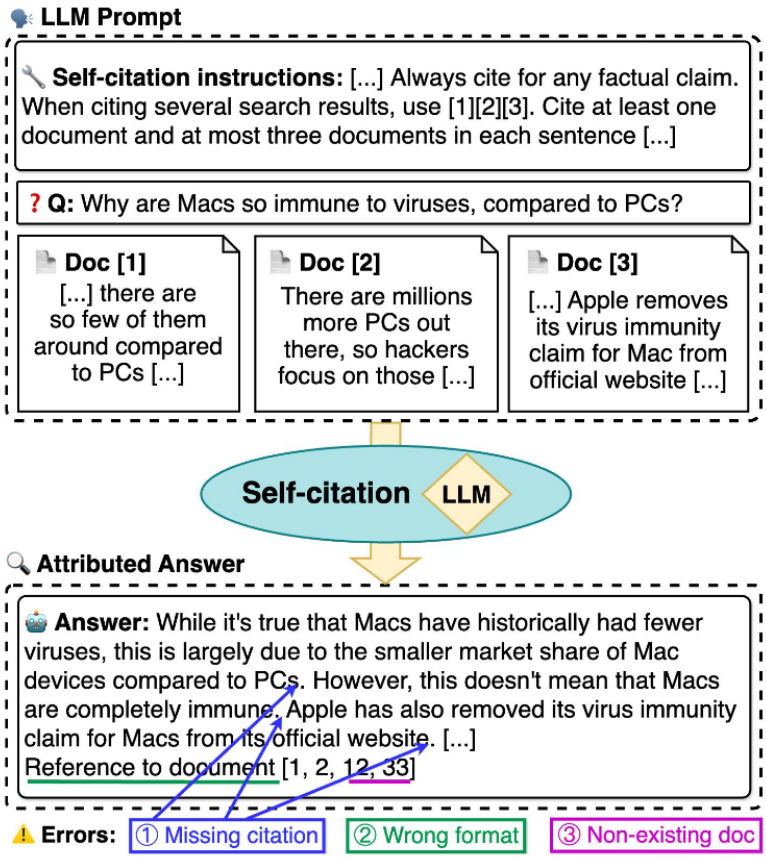


Figure 27: An example of self-citation weaknesses, taken from [52].

The context-sensitive tokens in the *generated sequence* are spotted by measuring the shift in the model's predictive distribution caused by the addition of the input context. This shift is attributed to some tokens *in the context*, found to be influential to the generated output.
This approach has already been employed in machine traslation tasks [58], the authors expanded it to the RAG framework. Their idea is to search for a match between context-dependent tokens in the generated sequence and the retrieved documents that effectively contribute to their prediction. Finally, these paired elements are converted to citations.

To spot which *generated* tokens are sensitive to the given context, the model needs to be prompted with a query q and a context $C = \{c_1, ..., c_{|C|}\}$[18] in order to produce a sentence $y = (y_1, ..., y_n)$ as output.
A contrastive metric $\hat{m}$ (e.g. KL-divergence) is used to quantify **at each generation step** the shift between:

- P^i_{ctx}: the model's predictive distribution at the i-th generation step with a prompt that includes the context;
- P^i_{no-ctx}: the model's predictive distribution at the i-th generation step with a *contextless* prompt.

Thus, at each generation step, we use the metric $\hat{m}$ to compute a score:

$$m_i = \hat{m}(P^i_{ctx}, P^i_{no-ctx}), \quad \text{e.g.} \quad m_i = \mathrm{KL}(P^i_{ctx} || P^i_{no-ctx})$$

The resulting (i.e. at the end of the generation process) scores $m = (m_1, ..., m_n)$ reflect the context sensitivity of each generated token.
To extract the most relevant ones, we can use a choice function s_{CTI} outputting whether each generated item is *sufficiently context-sensitive* or not:

$$CTI(q, c, y) = \{y_i \,|\, s_{CTI}(m_i) = 1, \forall y_i \in y\}$$

[18] Each c_i can be a separate document, a separate paragraph, a separate phrase, ... It depends on the granularity of the employed RAG application.

where $CTI(q, c, y)$ is an array containing all the relevant generated tokens y_i.

Once the context-sensitive tokens y_i have been identified, we also make the model predict a contrastive alternative $y_i^{\setminus c}$ by excluding the context C from the prompt but using the contextual generated prefix $y_{<i}$.

Now we have for each generation step a set of couples $(y_i, y_i^{\setminus c})$ representing the output produced with and without context. This first step has identified *which generated tokens change when the context is injected* in the prompt. We refer to this step as *Context-sensitive Tokens Identification* (CTI), and consequently to the elements that this process finds as *context-sensitive tokens* $y_i \in CTI(y)$.

The second step of this attribution procedure aims at discovering exactly *which context tokens are the ones that impact the most on the model's outputs*.
We refer to this process as *Contextual Cues Identification* (CCI), and consequently to these tokens as *contextual cues* $c_j \in CCI(y_i)$.

In order to identify which are the most important context tokens, a *contrastive feature attribution method* [80] can be applied:

$$a_i^j = \{\nabla_j(p(y_i) - p(y_i^{\setminus c_j})), c_j \in C\}$$

It substantially measures which context tokens c_j, if removed from the context, cause a bigger modification in the LLM's predictive function. a_i^j identifies which context items $c_j \in C$ influence the prediction of y_i, accounting for the non-contextual completion $y_i^{\setminus c}$.

These scores are transformed into binary labels similarly as it has been done for y_i in the CTI step, i.e. by using a choice function s_{CCI} outputting whether each *contextual cue* is *sufficiently influential* for the generated output or not:

$$CCI(y_i) = \{c_j \mid s_{CCI}(a_i^j) = \ \forall c_j \in C\}$$

This process results in pairs of context-sensitive generated tokens (CTI) and their respective context tokens influencing their prediction (CCI):

$$\{(y_i, c_j),\ \forall y_i \in CTI(q, c, y),\ \forall c_j \in CCI(y_i)\}$$

Note that both s_{CTI} and s_{CCI} contain implicitly a threshold which discriminates from what point onwards the tokens (both generated and in the context) are considered relevant. We can define them as:

$$s_{CTI} = m_i \geq m^*, \quad s_{CCI} = a_i^j \geq a_i^*$$

Practically, the authors suggested to set $m^* = \bar{m} + \sigma\bar{m}$, where $\bar{m}$ and $\sigma\bar{m}$ are respectively the average and standard deviation of m_i scores for the given example.

To filter the attributed context tokens $c_j \in CCI(y_i)$, a_i^* is either the Top-K or Top-% highest attribution value in a_i.

The final step proposed by the method consists in generating the citations. This step builds over the previous identification of relevant tokens in terms of generation.
We can construct the citation by selecting all the documents containing a *contextual cues* (CCI) c_j that are paired with a *context-sensitive generated output token* (CTI) y_i:

$$\text{MIRAGE}(y) = \bigcup_{y_i \in CTI(y)} \text{docid}(c_j),\ \forall c_j CCI(y_i)$$

2.11 Literature connections with our method

This long and extensive literature review presents multiple elements that will be helpful in presenting our experiments and in describing *where* our method is placed.

The first part (2.1) describes in detail the Transformer model's architecture and its modern improvements (2.2).
We consider these sections as preliminary to the description of the models we tested (2.3). With this introduction, we believe that many structural variations can be better understood. Tested models' details are essential to perform later analyses on (eventual) performance discrepancies. This should allow us to correctly spot the sources of different performance gaps among them, highlighting when they could be caused by our method or when they are external to it.

Then, we begin an overview of proposed methods to enhance generation quality. Besides pre-training stages, the models can be fine-tuned (2.4) to align their predictions with certain domain-specific applications or to give them the ability to answer in a pre-determined way to the given prompt.
A different approach has also been exploited to provide new information to the model, in order to update its knowledge or to provide additional one. The RAG approach (2.4) dynamically retrieves passages useful to answer to a question and appends them to the prompt. These appended additions are used to guide the model towards a grounded output. Further studies (2.8.1, 2.8.3) show how selecting the relevant information before appending it to the prompt can improve models' ability of exploiting context in the proper way, since it could be misleaded by noisy and irrelevant elements. A different approach (2.8.2) proposes to rephrase this framework in a NLI problem by transforming the prompt into a *hypothesis* and the provided context as a *premise* for it.

Fine-tuning, however, is not the only way of achieving tan alignment of model's behaviour with a desired one. It can be obtained via in-context learning (2.5), that simply provides "few-shots" examples appended to the prompt, showing the model how to perform a certain task.
Inducing the model to behave in a pre-determined way as in-context learning (2.5) allows the prompt to show the model how to execute a task, and consequently it can also be used to present a *reasoning chain* useful to solve the problem. Studies have shown that providing models with reasoning examples significantly improves the quality of their answers (2.6). This approach is known as Chain-of-Thought prompting.

The implicit idea of CoT is that the model is *allowed* (or, more properly, *prompted to*) take time and words to develop intermediate steps before choosing an answer. However, this is not the only option to produce *not definitive, intermediate steps* before outputting the final answer. In the wake of the method showed in section (2.8.2) ranked the quality of the candidate outputs before choosing the correct one (proved to outperform large fine-tuned alternatives), self-refinement methods (2.9) implement a certain number of corrections (2.9.2) or refinements (2.9.1, 2.9.3) to achieve a better answer.

Finally, some studies experiment beneficial effects of grounding reasoning in relevant context have been reported in this section (2.10). Among these, we find interesting the Rethinking with Retrieval approach (2.10.1), that produces multiple reasoning chains and consequently selects as answer only the most grounded one and the Chain-of-Citations/Chain-of-Quotes method (2.10.2), that induce the model to construct well-founded claims as reasoning chains by hinging them on context citations/quotes.

Our method builds on the joint efforts of the research community to which the creators of these methods belong. Specifically, we owe them the idea of improving a first tentative answer as it is done in self-refinement methods (2.9), although we do not consider our approach as a *refining* approach, nor a *correction* one.
We also integrated the context in our approach, relying on the observations reported in (2.8) and we also studied the influence that the original context has in the generated output, compared to its summarized version and to its filtered one. The filtering is carried out thanks to a slight variation of the MIRAGE (2.10.3) method: since it is able to pair the generated outputs with the context elements that influenced them, we can discard irrelevant parts of the passages appended to the prompt.
Finally, our method includes Chain-of-Thought steps in order to allow the model to reason on the correctness of a tentative answer and to eventually correct it.

3 Data details

Our analysis aims to improve the robustness of LLMs' answers. We focus on studying their ability to correctly respond to general knowledge questions in various settings, such as standard prompting, CoT prompting, a RAG setup, and combinations of these approaches.
Our method proposes to *dialectically improving* the answer before outputting it. In order to run these tests, we have to rely on datasets such that:

1. contain a set of questions and the correct answers to these questions;
2. append the relevant passages or documents that can be used to answer correctly to the question;
3. eventually provide also wrong answers to the question, presented as *distractors*.

In addition to this, we aim to test the ability of LLMs once the correct relevant passages are provided. Extensive studies (reported in 2.8) have shown how models benefit from the addition of the relevant context to their prompts, thus we will build on this finding and try to further enhance performances.

What instead we find interesting to study is the impact of the work that the model has to perform on the sources. We aim to test cases in which instead it is necessary to merge multiple pieces of information before answering. This task can be referred with the name of *Multi-Hop Question Answering* (MHQA) [46].
In broad terms, MHQA is the task of answering natural language questions that involve extracting and combining multiple pieces of information and doing multiple steps of reasoning. An example of a multi-hop question would be:
`Who is the oldest candidate in the 2024 USA presidential election?`.
Answering the question would require to join many pieces of information:

- `What are the names of presidential candidates to the 2024 USA presidential election?`
- `What is the age of [candidate name]?`
- `What is the largest number between [age1], [age2], ...?`

The ability to answer multi-hop questions and perform multi step reasoning can significantly improve the utility of NLP systems.

Single-hop QA often does not require any form of reasoning, limiting itself to summarizing or paraphrasing the content present in the source to produce a proper answer to the question.

Multi-hop QA asks the model to perform a step further. An agent can be said to perform *multi-hop reasoning* if it reaches one or more intermediate conclusions before deriving the final answer and each of the intermediate conclusions serves as a necessary premise for some other conclusion. This sequence of intermediate conclusions, including the final answer, is called a reasoning chain and each step from one conclusion to the next can be referred to as a *hop*. Humans can easily perform these multi-step reasoning in their everyday tasks, yet this is still a difficult task for machines. We would like to improve LLMs' multi-hop abilities since they could be useful in many concrete applications. Queries given to current web search systems can often require multi-hop reasoning to reach the relevant documents; user satisfaction when using such systems can be greatly improved by utilizing multi-hop reasoning models; also conversations between humans and agents can be smoother and more informative if the latter can handle complex questions.

Our analysis starts from the choice of datasets. We are interested in studying how our method performs on HotpotQA and WikiHop datasets. As their names can clue, they both require the model to perform some *multi-hop* reasoning steps before answering.

3.1 HotpotQA

3.1.1 Dataset description

HotpotQA is a question answering dataset collected on the English Wikipedia, containing about 113K crowd-sourced questions that are constructed to require the introduction paragraphs of two Wikipedia articles to answer. Each question in the dataset comes with the two gold paragraphs, as well as a list of sentences in these paragraphs that crowdworkers identify as supporting facts necessary to answer the question.

Yang et al. [79] constructed HotpotQA ensuring that 4 key features are guaranteed:

1. the questions require finding and reasoning over multiple supporting documents to answer;
2. the questions are diverse and not constrained to any pre-existing knowledge bases or knowledge schemas;
3. sentence-level supporting facts required for reasoning are provided, allowing QA systems to reason with strong supervision and explain the predictions;
4. a new type of factoid comparison question is introduced to test QA systems' ability to extract relevant facts and perform necessary comparisons.

HotpotQA contains only 2-*hop* questions formed using the first passages of documents from the English Wikipedia dump. The passages are chosen if they satisfy either of the two conditions:

- There exists a hyperlink from the first document to the second. The entity which forms the hyperlink is termed as the `bridge` entity and the questions are termed as bridge questions.
- The entities for those passages belong to the same category (e.g. Michael Jordan and Kobe Bryant). These are specifically sampled from 42 manually created lists. Such pairs are used for creating `comparison` questions [46].

An example of HotpotQA is the following:

```
{'id': '5a7a06935542990198eaf050',
 'question': "Which magazine was started first Arthur's Magazine or First for Women?",
 'answer': "Arthur's Magazine",
 'type': 'comparison',
 'level': 'medium',
 'supporting_facts': {'title': ["Arthur's Magazine", 'First for Women'], 'sent_id': [0, 0]},
 'context': {
    'title': ['Radio City (Indian radio station)', "Arthur's Magazine", ... ],
    'sentences': [
        ["Radio City is India's first private FM radio station and was
        started on 3 July 2001.", ...] , ...
    ]
 }
}
```

Where:

- `id` for the question-answer couple;
- `question`, `answer` are simply the question and the correct answer to the first;
- `type` is the required type of reasoning on the context: `comparison` and `bridge` (details are provided below);
- `level` is a human-labelled score based on how challenging the question is; possible values of it are `easy`, `medium` and `hard`;
- `context` contains 10 documents, of which 8 distractors and 2 relevant ones. For each document/passage are provided:

  - the `title` of the Wikipedia paragraph;
  - a vector of `sentences` identified by the crowdworkers as relevant;

- `supporting_facts` contains the `title` of the 2 gold documents of the 10 provided.

From now on, we will refer separately to the subset of dataset containing `comparison` and `bridge` type of questions. We do this distinction since we consider important to test separately these two different *multi-hop* sub-tasks. While `comparison` requires to extract (almost) the same information from passages referring to two different objects, `bridge` tests the ability to merge multiple sources in a homogeneous view that allows to answer to the question.

We show the difference between the two tasks using two examples:

- **`comparison`:**

```
Question: Which magazine was stated first, Arthur's Magazine or First for Women?
Answer: [Arthur's Magazine]

Context: Arthur's Magazine (1844-1846) was an American literary periodical published
in Philadelphia in the 19th century. Edited by T.S. Arthur, it featured work by Edgar A.
Poe, J.H. Ingraham, Sarah Josepha Hale, Thomas G. Spear, and others. In May 1846
it was merged into "Godey's Lady's Book". First for Women is a woman's magazine published
by Bauer Media Group in the USA. The magazine was started in 1989. It is based in
Englewood Cliffs, New Jersey. In 2011 the circulation of the magazine was 1,310,696 copies.
```

- **`bridge`:**

```
Question: The Oberoi family is part of a hotel company that has a head office in
what city?
Answer: [Delhi]

Context: The Oberoi family is an Indian family that is famous for its involvement in hotels,
namely through The Oberoi Group. The Oberoi Group is a hotel company with its head office in
Delhi. Founded in 1934, the company owns and/or operates 30+ luxury hotels and two river
cruise ships in six countries, primarily under its Oberoi Hotels & Resorts and Trident
Hotels brands.
```

3.1.2 Data processing

The necessary transformation that the dataset had to meet was to create another plausible answer to the question.
Due to the structure of the `comparison` questions, we find natural to add only one plausible alternative, practically the other option given by the question.
Consider for example:
`Which magazine was stated first, Arthur's Magazine or First for Women?`
it is clear (even without knowing the correct answer) that the only two plausible options are
`Arthur's Magazine` and `First for Women`.
We had at disposal the `title` array (inside the `context` attribute) and the `title` of the only two relevant passages (inside the `supporting_fact` attribute). Due to what we want to study, we chose to neglect a detailed analysis on the introduction of non-meaningful passages in our pipeline.
Thus, we kept only the `sentences` items corresponding to the relevant documents, and merged them in a single text passage.
We could not use the `title` of relevant passages as couples of correct and wrong answers to the questions. While some cases allowed us to build an automatic procedure to do so, e.g.

```
Which magazine was stated first, Arthur's Magazine or First for Women?

[("Arthur's Magazine",
  ["Arthur's Magazine (1844-1846) was an American literary periodical published in
  Philadelphia in the 19th century.", ' Edited by T.S. Arthur, it featured work by Edgar A.
  Poe, J.H. Ingraham, Sarah Josepha Hale, Thomas G. Spear, and others.',
```

```
  'In May 1846 it was merged into "Godey\'s Lady\'s Book".']
  ),
  ('First for Women',
   ["First for Women is a woman's magazine published by Bauer Media Group in the USA.",
   'The magazine was started in 1989.', ' It is based in Englewood Cliffs, New Jersey.',
   'In 2011 the circulation of the magazine was 1,310,696 copies.']
  )
]
```

It is clear that in front of the question of the current example, it would be enough to extract `Arthur's Magazine` and `First for Women`. On the opposite, another example is the following:

```
Which band was founded first, Hole (the rock band that Courtney Love was a frontwoman
of) or The Wolfhounds?

[('The Wolfhounds',
  ['The Wolfhounds are an indie pop/noise pop band formed in Romford, UK in 1985 by Dave
  Callahan, Paul Clark, Andy Golding, Andy Bolton and Frank Stebbing, and originally
  active until 1990.', ' The band reformed in 2005 and continues to write, record and
  play live, releasing new albums in 2014 and 2016.']
 ),
 ('Courtney Love',
  ['Courtney Michelle Love (born Courtney Michelle Harrison; July 9, 1964) is an American
  singer, songwriter, actress, and visual artist.', 'Prolific in the punk and grunge
  scenes of the 1990s, Love has enjoyed a career that spans four decades.', 'She rose to
  prominence as the frontwoman of the alternative rock band Hole, which she formed in 1989.',
  'Love has drawn public attention for her uninhibited live performances and
  confrontational lyrics, as well as her highly publicized personal life following her
  marriage to Kurt Cobain.']
 )
]
```

Despite being relevant to the answer, `Courtney Love` would not be a plausible alternative answer since she is a singer, not a band. The correct alternative would be `Hole`, which cannot be directly extracted from the context `title`.

We used `Phi-3-mini` to produce a plausible alternative to the correct answer to the question. We use greedy decoding with temperature set to 0 to sample an alternative option, limiting the maximum number of new tokens to 20[19]. The prompt used to generate the output is one-shot and is the following:

```python
def produce_prompt(question, correct, source):

    user_content = "Question: " + question + "\n Correct answer: " +  correct +
                   "\n Context: " + source + "\n\n Assistant:"

    messages = [
        {"role": "system", "content": """
        You are a helpful AI assistant. You are given a question and the correct answer to it.
        Given the context, you have to provide a wrong, yet realistic, alternative answer to
        the same question given the context.
        Output a synthetic answer in the same style as the correct answer.

        For example:

        Question: Which magazine was started first Arthur's Magazine or First for Women?
        Correct answer: Arthur's Magazine
        Context: Arthur's Magazine (1844-1846) was an American literary periodical published in
```

[19]This is because of the answer style in HotpotQA, very synthetic and essential. We did not need the model to produce an output and justify it, but instead to extract from the context another realistic option.

```
        Philadelphia in the 19th century. Edited by T.S. Arthur, it featured work by Edgar A.
        Poe, J.H. Ingraham, Sarah Josepha Hale, Thomas G. Spear, and others. In May 1846 it was
        merged into Godey\'s Lady\'s Book. First for Women is a woman's magazine published
        by Bauer Media Group in the USA. The magazine was started in 1989.
        It is based in Englewood Cliffs, New Jersey. In 2011 the circulation of the magazine
        was 1,310,696 copies.

        Assistant: First for Women
        """},

        {"role": "user", "content": "Now to the same for this problem: " + user_content},
    ]
    return messages
```

This function allows to append to the prompt an example and then to ask the model to perform a similar task for a new triplet of question, correct answer and context.

In the `comparison` subset, questions are also presented in a different style, that does not explicitly refer to multiple options, for example:
`750 7th Avenue and 101 Park Avenue, are located in which city?`
In this example, both the correct answer and the alternative do not stand out in the question. They need to be extracted from the context and the correct answer instead.
The prompt presented above still works quite well for this type of questions, even though the context does not explicitly mentions any plausible alternative to `New York City`:

```
[('101 Park Avenue',
  ['101 Park Avenue is a 629 ft tall skyscraper in New York City, New York.', 'It was completed
  in 1979 to 1982 and has 49 floors.', 'Eli Attia Architects designed the building, which is
  the 64th tallest in New York.']),
 ('750 7th Avenue',
  ['750 Seventh Avenue is a 615 ft (187m) tall Class-A office skyscraper in New York City.',
  'It was completed in 1989 in the postmodern style and has 36 floors.', 'Kevin Roche John
  Dinkeloo & Associates designed the building, and it is owned by Hines, a Texas based real
  estate investment company.', "The building's continuous helix design, culminating in a
  chimney-like extension, was caused by the New York City Building Code, which requires
  setbacks.", "The 84 exterior column transfers exist because of the owner's requirement
  for a column-free space.", 'It is tied with the New York Life Building for the 74th
  tallest building in New York City.', 'It is also LEED certified.']
 )
]
```

In most of the examples, `Phi-3-mini` is able to use its parametric knowledge to extract some plausible alternative. A hand-crafted correction is applied to cases in which the model produces clearly wrong options. For example, a plausible alternative to `New York City` would not be `Trieste`, since the latter does surely not possess `avenues`.
Another option, e.g. `San Francisco`, is more realistic and challenging to be checked (if the context is not used properly or neglected).

The last kind of alternatives that can be found is the basic `yes` and `no` questions. For this partition of `comparison`, that can be easily selected by considering all the rows in which `answer` is one between `yes` and `no`, the alternative is simply the opposite. This does not require the use of any Large Language Model.

Due to the need for a hand-crafted correction in some cases, we limited the dataset to the first 352 examples of the training set. We do not select the problems according to any different criterion and we checked that the models tested were not pre-trained or fine-tuned on this subset[20].

[20] See the Results chapter: the baseline performances are not good, both with and without the relevant passages appended to the prompt, thus it is improbable that the models were trained on that data.

For our experiments, this number of items is enough to assess whether we experience an improvement or not. Generating alternatives with Phi-3-mini is not extremely expensive in computational terms, but requires also a hand-crafted correction sometimes. Also, we chose to keep the dataset balanced and not to add all the yes/no questions present in the original dataset.

When the same prompt is tested on the bridge subsection of the dataset, we observe particularly good results, that do not need to be hand-crafted as a post-processing stage.
Concrete examples of non-trivial alternatives created using Phi-3-mini on bridge subset:

```
What U.S Highway gives access to Zilpo Road, and is also known as Midland Trail?

Zilpo Road is a National Forest Scenic Byway in the forested hills of eastern Kentucky,
United States. The nine mile byway starts south of Morehead, Kentucky and can be accessed
by U.S. Highway 60. The byway travels through the Daniel Boone National Forest and ends on
the western shore of Cave Run Lake at the Zilpo Recreation Area. It follows FSR 918, which
is a two lane paved road suitable for all motor vehicles and is usually open throughout the
year. Morehead is a home rule-class city located along US 60 (the historic Midland Trail) and
Interstate 64 in Rowan County, Kentucky, in the United States. It is the seat of its county.
The population was 6,845 at the time of the 2010 U.S. census.
```

The correct option is U.S. 60 and the produced alternative is U.S. 50. Of course the context does not mention U.S. 50, since it is not relevant for the question, but produces a realistic alternative that (if the context is not given) could still be challenging to discard.

Another option is given by:

```
What nationality was James Henry Miller's wife?

James Henry Miller (25 January 1915 - 22 October 1989), better known by his stage name Ewan
MacColl, was an English folk singer, songwriter, communist, labour activist, actor, poet,
playwright and record producer. Margaret "Peggy" Seeger (born June 17, 1935) is an American
folksinger. She is also well known in Britain, where she has lived for more than 30 years,
and was married to the singer and songwriter Ewan MacColl until his death in 1989.
```

Even if the correct answer, American, is quite trivial if the model is able to link Ewan MacColl with its true name James Henry Miller, the context could be quite deceiving.
Both the first passage regarding James Henry Miller and the second regarding Margaret "Peggy" Seeger quote the fact that he is English and that she well known in Britain respectively, thus the alternative is really well grounded in the context.

Due to this more reliable pre-processing stage of Phi-3-mini, we chose to select the first 1000 examples in the bridge subset of HotpotQA's training set. As for comparison, this was the only selection criterion used to pick problems on which performing tests.

The reason why Phi-3-mini seems more reliable on this second split could be found in the most various context provided by the bridge subset.
Consider for example the New York City example: since the comparison consists in spotting the common elements in both their passages, presumably they are already select to discard off topic additional information. In bridge questions instead, context is structurally composed by passages containing a part of information that is relevant and must be merged with other sources and peddling ones. An example is given by the U.S. highway example, in which is reported the population of Rowan County according to 2010 U.S. census.

3.2 WikiHop

3.2.1 Dataset description

WikiHop is a part of a greater dataset called QAngaroo and proposed by Welbl et al. [71].
QAngaroo is a Reading Comprehension dataset focusing on *multi-hop* inference. Several pieces of information often jointly imply another fact, thus a new fact is derived by combining facts via a chain of multiple steps.

In each sample, a query is given about a collection of documents. The goal is to identify the correct answer among a set of given type-consistent answer candidates[21]. The candidates — including the correct answer — are mentioned in the documents.

An example of multi-hop question in WikiHop:

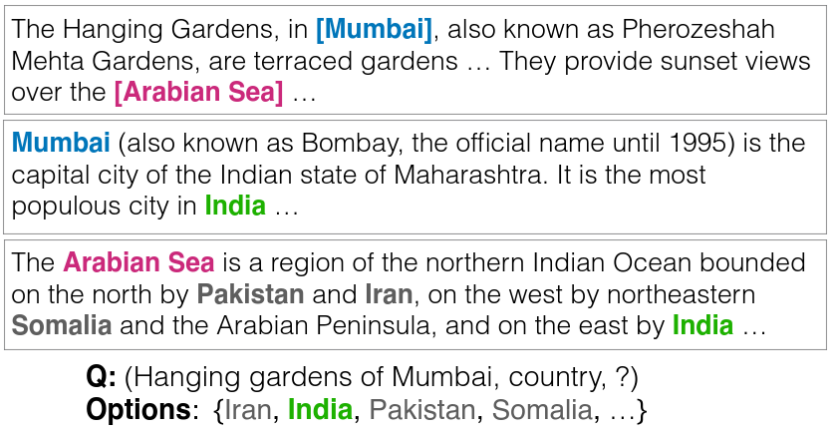


Figure 28: A sample from the WikiHop dataset where it is necessary to combine information spread across multiple documents to infer the correct answer, taken from [71].

As previously mentioned, QAngaroo contains two distinct datasets:

- WikiHop: contains open-domain and based on Wikipedia articles; the goal is to recover Wikidata information by hopping through documents. The example above shows the relevant documents leading to the correct answer for the query shown at the bottom.
- MedHop: based on research paper abstracts from PubMed, the queries are about interactions between pairs of drugs. The correct answer has to be inferred by combining information from a chain of reactions of drugs and proteins.

We chose to focus on WikiHop since MedHop covers a limited number of topics and asks to output scientific acronyms of names or proitein labels:

```
Q: What interacts with DB00773?
Options: ["DB00072", "DB00294", "DB00338", "DB00341", "DB00588", "DB00820", "DB02546",
          "DB02901", "DB04844"]
```

The options (i.e. the possible answers to the question) are extracted from the source documents, and only one among all the candidates is the correct one[22].

An example taken from WikiHop is the following:

```
{"id": "WH_train_0",
 "query": "participant of juan rossell",
```

[21] Masked versions of these two datasets are also available, but for our analyses we will use not-masked ones. Refer to [71] for further details.

[22] In the current example, we have omitted the context since of much greater size of those of HotpotQA, you can check this by looking at the dataset [71].

```
 "answer": "1996 summer olympics",
 "candidates": ["1996 summer olympics", "olympic games", "sport"],
 "supports": [
    "The 2004 Summer Olympic Games, officially known as the Games of the XXVIII
    Olympiad and commonly known as Athens 2004, was a premier international
    multi-sport event held in Athens, Greece, from 13 to 29 August 2004 with the
    motto \"Welcome Home.\" 10,625 athletes competed, some 600 more than
    expected, accompanied by 5,501 team officials from 201 countries. There were
    301 medal events in 28 different sports. Athens 2004 marked the first time
    since the 1996 Summer Olympics that all countries with a National Olympic
    Committee were in attendance. 2004 marked the return of the games to the
    city where they began.",
    ...
  ]
}
```

Where:

- `id` identifies the sample;
- `query` specifies the information that should be extracted from the texts;
- `answer` is the correct answer to the query;
- `candidates` is a list of answer candidates, each of which is mentioned in one of the `supports` passages;
- `supports` is a list of support documents.

3.2.2 Data processing

Compared to HotpotQA, here we face quite opposite problems. The candidate options are provided and centered on the topics and the words present in the context, and each example shows a various number of candidate options (not just two, as before). This makes the dataset more challenging than the previous one.
Similarly as before, different `supports` are merged together into an homogeneous source.

The great difference is that the `query` is very essential and schematic, not appearing like a concrete question but instead as a sketch of it.
Preliminary tests showed that models found difficult to read that style as a question, thus a pre-processing stage is necessary. We used again `Phi-3-mini` in a one-shot setting to produce a question out of the schematic sketch of it. We used greedy decoding with temperature set to 0 to generate the question. The maximum number of new tokens allowed is 500, but never reached in practice. The prompt used to generate the question is the following:

```
def create_message(question, options):
    user_content = "Question: " + question + "\n Options: " + options + "\n\n Assistant:"

    messages = [
    {"role": "system", "content": """}
    You are a helpful AI assistant. You are asked to create a question out of a sketched question.

    Question: "occupation cao chong"
    Options: ['academic', 'builder', 'chancellor', 'classics', 'confucian scholar', 'designer',
    'duke', 'emperor', 'engineer', 'engineering', 'father', 'founder', 'general', 'king',
    'leader', 'major', 'mathematician', 'military', 'official', 'peasant', 'physicist',
    'physics', 'politician', 'prior', 'rebel', 'research', 'ruler', 'science', 'script',
    'social reformer', 'socialist', 'sovereign', 'taiwan']

    Assistant: "Which was the occupation of Cao Chong?"
    """
```

```
    },

    {"role": "user", "content": "Now do the same for this question: " + user_content},
    ]

    return messages
```

This task is performed perfectly by the model; we chose not to append the context since it is enormously large and does not make sense in terms of question construction. The one-shot example is chosen to include a significant number of options, allowing the model to learn how to handle a large number of choices without becoming confused.

The first 1000 queries of the WikiHop dataset were processed as above and stored in a subset used for analysis purposes. No other selection criteria have been used to produce the WikiHop partition for our analyses.

3.3 Datasets' summary statistics

In the previous paragraphs we have already highlighted the fact that in HotpotQA the context is structured by only two *hops* and the options for each question are always two.
In addition to this, the difficulty level of the question (i.e. `easy`, `medium` and `hard`) is neglected in this analysis, due to the fact that the same question could be more or less challenging given a different alternative option.

Some summary statistics regarding the number of words in the tested datasets are reported below. We assume that the number of words in the `query` but in particular in the `context` is a proxy for the difficulty level of the *multi-hop* reasoning task. The idea is that, even if the number of hops is fixed to two, the more verbose the `context` is, the more information has to be merged before answering.

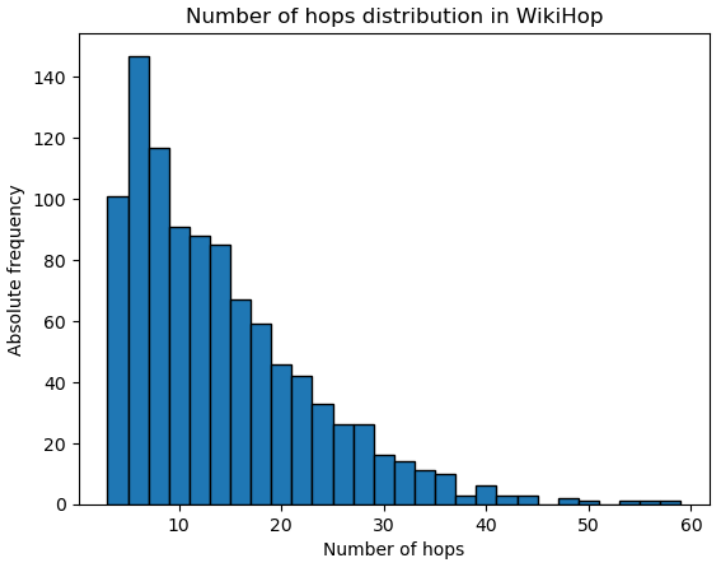
HotpotQA partition	`min`	`mean`	`std`	`max`
`query`				
`comparison`	5	11.6	3.9	42
`bridge`	5	19.7	10.5	94
`context`				
`comparison`	28	118.0	47.8	295
`bridge`	40	138.7	53.0	502

While the `query` sizes (in terms of number of words) is almost the same for both the HotpotQA partitions, the `context` ones are drastically different.

The `query` sizes are slightly higher for the `bridge` subset, probably caused by some extremely long questions (we can tell from the great standard deviation and high maximum value, while the minimum and the mean one are the same or only slightly higher).

The `context` size of `bridge` is slightly longer than the `comparison` one.
This is probably due to the same conclusions that we draw in section (3.1.2) regarding the amount of additional information present inside `bridge`'s `context` attribute.

On the opposite, WikiHop has a variable number of options and a variable number of *hops*.
Below is reported the distribution of the number of *hops*[23]:



[23]The number of options are closely related to the number of *hops*: since they are extracted from items mentioned in the context, the longer the passage, the higher the number of options provided. And when we have many *hops*, the context is typically longer.

4 Methods

Our work builds on other researchers' findings and ideas, despite trying to explore a new field of LLMs' abilities.
In the literature review, we discussed how recent efforts in the NLP field focus on finding various solutions to enhance model generation and prediction abilities. These efforts often involve merging multiple methods or applying old ideas with new technologies.

We can imagine to construct a sharp distinction between the proposed approaches[24]:

- approaches that modify the model's weights in order to obtain the desired output (e.g. SFT, PEFT);
- context-based approaches, in which context is used to boost model performances with relevant non-parametric, external knowledge (RAG, RE-RAG, NLI verifiers, S2A);
- prompting approaches, that asks the model to perform a task in a certain way provided in a couple of examples (in-context learning), to think carefully before answering (CoT) and eventually to refine some previous answer given in input (self-refinement approaches).

The first one improves the output thanks to internalized, downstream-specific knowledge; the second one uses wisely the given context and relies on it to output a proper answer; the third one just tries to exploit latent knowledge present in the pre-trained model.

In literature, it is often used the term *reasoning* for cases in which models exploit some CoT-like procedure to come up with the correct answer, while the product of the other two approaches is considered *training* or *retrieval*.

If we refer to the Oxford Dictionary, we find that **reasoning** is defined as: *the process of thinking about things in a logical way; opinions and ideas that are based on logical thinking* while **thinking** is defined in a slightly different way: *the process of thinking about something*.
Thus, **reasoning** appears to be more logically-based and decision-driven, while **thinking** can also be an unspoken, silent and internalized process.

This means that, given the black-box nature of LLMs, we cannot exclude that the process of activating the parametric knowledge stored in the network's weights isn't a form of thinking. We will not go deeper in arguing whether or not this process is linked with a (spontaneous) will of the model, since this would be outside the discussion's topics.

On the opposite, the **reasoning** process involves a stream of intermediate, logical connections and a final decision coming out from this process. This is why literature often refers to reasoning abilities when talking about Chain-of-Thought approaches. The great turning point with respect to other techniques is that, by explaining the logical process leading to an answer, we can clearly see that the model *is reasoning* about it.
Thus, the crucial component that defines a generation process as a reasoning process is its explicitness.
For example, Rethinking with Retrieval (2.10.1) does not exactly reason about the possible solution. It samples many possible *chains of justification* for a given answer and it discards those which are not supported by the retrieved content for the corresponding question. This works as a sort of *pruning* of all not context-grounded generated chains. This approach helps improve the baseline answer (i.e., the standard prompting answer given without context). However, when compared to an output obtained via a self-consistency approach, the improvement is not as significant[25].

Chain-of-Citations or Chain-of-Quotes approaches (2.10.2) still use the context but through an opposite strategy.
Producing citations or quotes such as Li et al. [42] did helps in grounding each step of the reasoning chain in a relevant text passage, i.e. does quite the opposite as Rethinking with Retrieval. Binding to the definition of *reasoning* given above, this is nearest to it than (2.10.1): each passage is supported by

[24]Even though an effective clustering operation would be not appropriate due to the inherently mixing nature of modern machine learning technologies: consider that as only an aid for a clearer explanation.
[25]A 4% improvement on commonsense; 2% on temporal and 1% on tabular reasoning tasks [24].

a portion of the knowledge base, and they are linked together in order to produce a plausible *chain of justifications* leading to the final (and hopefully correct) answer.

A careful selection (by the model itself) of the context on which the following answer is built can be seen also as a form of asynchronous decision regarding the generation. Neglecting out-of-topic information or selecting just some relevant pieces of the given passages is proved to boost significantly the performances [73]. Under a certain light, carrying out a first passage to skim irrelevant components for the given question (or prompt, more generally) is a form of *thinking about something in order to make a decision*.

Considering this, it would be maybe excessive to consider RAG and RE-RAG context selections as a form of *thinking* or *reasoning*, since often the comparison is made using similarity scores or external ranker components (2.8.1).

A different conclusion has to be drawn for MIRAGE (2.10.3), in which the generation process iteratively spots a light on which element of the context triggers the generation of the current token. While the prediction is exactly the same as the one we would obtain by just appending the context to the question, the *contextual clues* spotted can be used for interpretability analyses. This is useful to recognize some hints of an underlying form of *reasoning* that the model performs.

What instead follows from this definition of *reasoning* is that translating a QA problem in a NLI task (2.8.2) and consequently determining whether or not the answer is entailed with the context is a form of (lateral) reasoning on the task.

The most familiar way (besides CoT approaches) that we recognize as reasoning is the one that we attributed to self-refinement approaches.
Methods such as `SELF-REFINE` (2.9.1), `SELF-CORRECTION` (2.9.2) and Reflexion (2.9.3) all starts from the assumption of generating a first tentative answer and iteratively correcting it, until a certain *level of satisfaction*[26] is achieved. This appears very much alike the human learning process: we come out with a first attempt, check whether or not it is correct, eventually correct it.
The three approaches have different ideas on their basis. `SELF-CORRECTION` simply iteratively corrects the generated output using a specialized, pre-trained corrector module.
Both `SELF-REFINE` and Reflexion are a three-components approaches, having in common the idea of structuring the pipeline as first attempt $\rightarrow$ feedback/evaluation $\rightarrow$ refinement.
While `SELF-REFINE` proposes a "natural-language" refinement pipeline (i.e. the first attempt and the suggestion are provided to the model by appending them to the prompt), Reflexion updates model's hyperparameters through a Reinforcement Learning approach (2.9.3).

4.1 Reasoning or simply imitating?

Many voices stand against the opinion that LLMs can reason or plan. In fact, in the opinion of many researchers, we should imagine these models as very good *universal approximate retrievers* [35].

By that, they mean that the models are essentially just improved n-grams, pre-trained on a massive scale of web data and language corpora, hence truly capable when asked to complete a sentence in the proper way. But they are asbolutely incapable of performing any kind of reasoning that does not imply access to previously memorised knowlegde.

The big difference stands in what we refer to when we talk about *thinking* or *reasoning*. The previous paragraph assumes the Cambridge and Oxford definitions, while Kambhampati introduces the System 1 and System 2 architecture [34] in decision marking to support the idea that System 2 is something absolutely unknown to LLMs.
Kambhampati defines the LLMs' parametric knowledge *as a giant non-veridical memories akin to an external System 1* [35].
This is his proposed description:

[26] Different for each approach, typically bounded by a fixed number of iterative refinements or by a stopping criterion on the quality of the generated output.

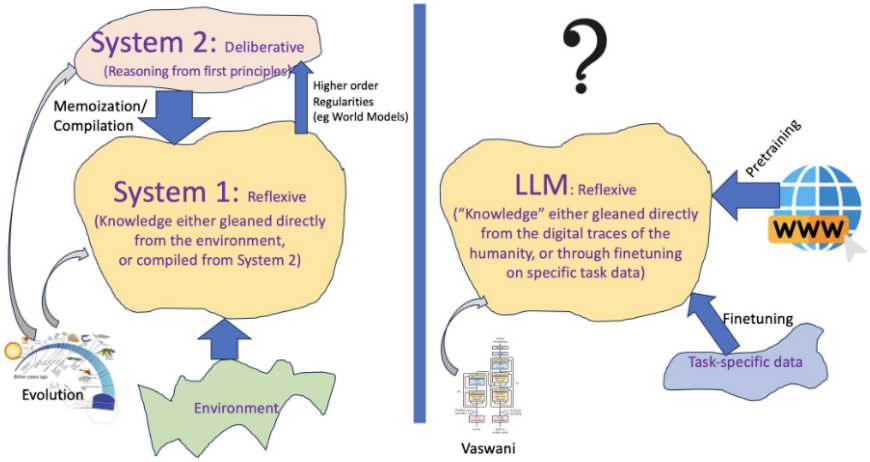


Figure 29: On the left, System 1 and System 2 as proposed by Kahneman [34]; on the right, the LLMs' pseudo System 1 (taken from [35]).

The researcher and his team worked to test their strong assumptions against the supposed reasoning abilities of LLMs, and found that the pre-training corpus has a strong impact on the performances of those models in challenging reasoning tasks.
They presented to the `GPT-4` and `GPT-3.5` many different shift ciphers problems, varying the shift value from 1 to 25. What they observed is that the models were very accurate on certain numbers of shifts, while completely out of clue in other cases:

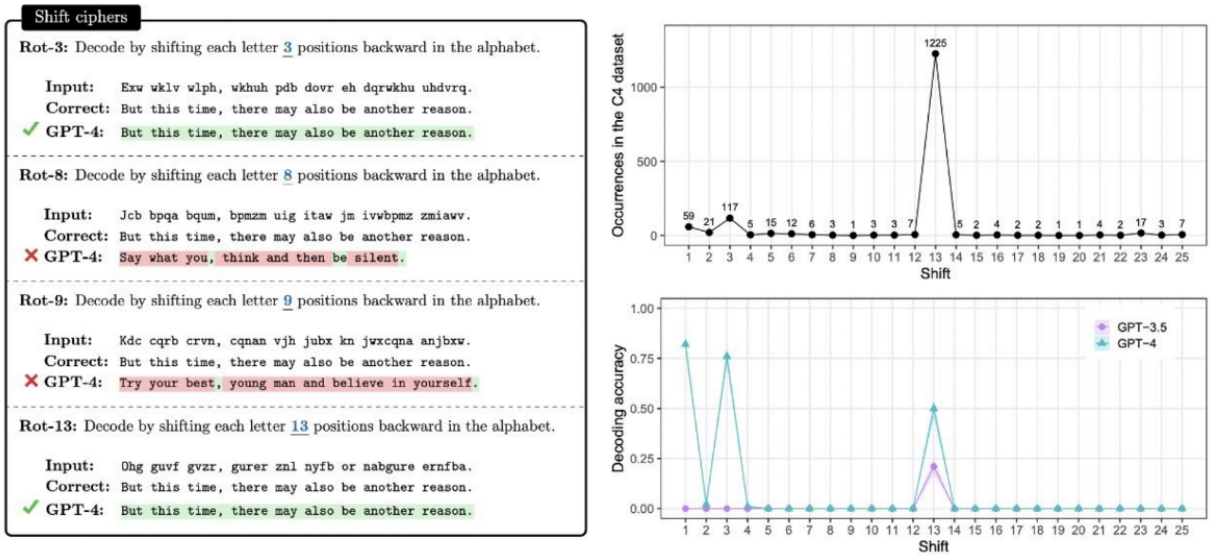


Figure 30: Tests of shift ciphers for different shift values, from [36].

In fact, it is true that the *deliberative* part of the decision system is not (at least up to now) a LLM feature. On the other hand, it is also true that in many situations it could be enough to imitate the behaviour observe in massive sources of data.

From now on, when we refer to reasoning, we will intend the definitions in 4 or, equivalently, to the pseudo System 1 as proposed by Kambhampati in [35].

4.2 A dialectic pipeline

Our method owes many ideas from previous works presented in the literature review section (2).
We propose a three-step method composed of a thesis, an antithesis and a synthesis of the two previous steps.

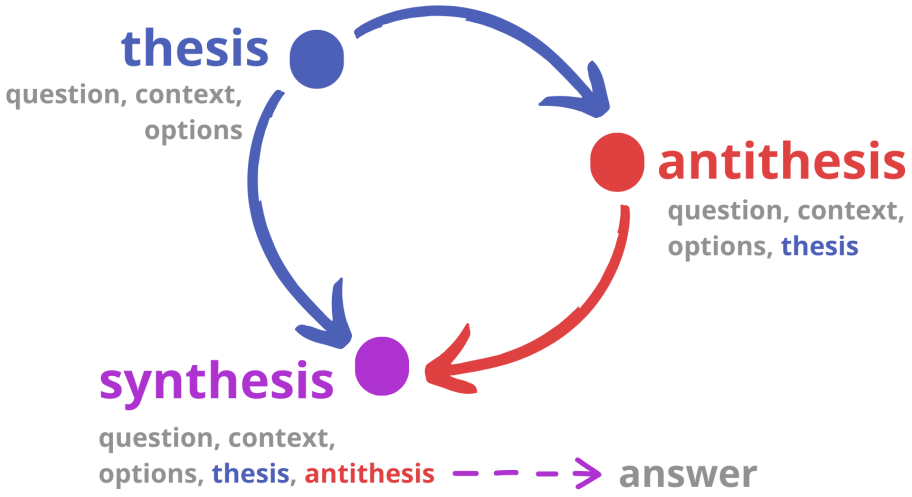


Figure 31: A visual representation of the proposed pipeline. Blue, red and purple represent the thesis, the antithesis and the synthesis steps respectively. Colored arrows highlight which element generated the subsequent item. The grey components are the structural components of the task (i.e. question, context and possible answer options). The answer is produced only after the synthesis step.

Our method does not perform a *linear* refinement of the answer such as `SELF-REFINE`, `SELF-CORRECTION`, Reflexion or Rethinking with Retrieval approaches.
The *linear* refinement is performed when the first tentative answer (from now on, *thesis*) is checked and eventually corrected by the same LLM in the *antithesis* step.

But our idea is to further check the correctness of the *antithesis*' suggestion by prompting the same LLM again. This new process receives as inputs both the *thesis* and the *antithesis* and performs a final pass on the problem before outputting the correct solution. We call this phase the *synthesis* stage.

But why did we insert an extra step?
Differently from self-refinement methods summarized in (2), our method does not require any form of task-specific prompting[27] or fine-tuning. This means that the correction (i.e. the *antithesis*) is simply obtained by one-shot prompting with a fixed prompt structure describing the desired behaviour of that pipeline stage. Thus, an additional check of the proposed correct answer could be beneficial.
In addition to this, considered that we have at least two options between which the choice has to be made, we can face two scenarios:

- the *thesis* and the *antithesis* agree on which option is the correct answer to the question, thus the *synthesis* receives a single option, already motivated by the *antithesis* stage;
- the *thesis* proposes an answer, the *antithesis* suggests why another option is more proper (and why the *thesis*' suggestion is wrong).

The *synthesis* considers the question, the options and the supporting context and decrees which is the most correct alternative answer, once listened to the *thesis*' and *antithesis*' opinions.

[27] For example, `SELF-REFINE` prompts the model to `Show step-by-step solution` for math reasoning tasks while to `Include keywords; maintain coherence` for constrained generation ones; we did not change the instruction prompt with respect to the specificity of multi-hop subtask that we considered.

Discarding this last step could potentially harm the accuracy of the process. Consider for example the case in which the *thesis* correctly predicts the answer, while the *antithesis* step loses some information[28] and outputs a detailed explanation on why the correct answer is a wrong option.
By adding the synthesis step we force the model to compare different proposals and hopefully reach the most proper answer among them. Note that we do not bound the synthesis step to choose an option between the two proposed by the previous steps; when more candidates are present, the *synthesis* step is left free to choose a third, unseen option given the context and the question.

Some experiments will be performed on the effectiveness of adding the *synthesis* step. We will report the effectiveness of the pipeline up to the *antithesis* stage and the complete one.

We prefer to define our approach as a *dialectic* pipeline in place of a *refinement* one because of the following reasons:

1. we do not ask the steps to refine the output, instead we prompt the *antithesis* and the *synthesis* to check the previous steps' outputs and return their opinion on which is the most correct options;
2. the three steps of the pipeline are essentially autonomous one from the other: for example, we can obtain an *antithesis* out of an hand-crafted or even completely casual *thesis*, without running the *thesis* part before;
3. due to each component's intrinsic autonomous nature, it is not guaranteed that we are able to observe an improvement in *synthesis*' answers with respect to the *thesis*' ones, since it is not a *refinement* process;
4. even though being autonomous actors in the pipeline, each actor adding an opinion on the previous ones, the entire pipeline can be seen as a model dialoguing with itself, in a sort of guided and *disentangled*[29] Chain-of-Thought.

4.3 Answering given the context

Although not strictly necessary, we chose to provide the relevant context to each step of the pipeline.
This is due to the widely discussed reasons in the literature section (2) of the positive impact of appending the relevant context in QA tasks.

Our method aims at testing a pipeline that should be robust even when used with smaller or less widely trained models to assess the level of reliability of the pipeline with respect to different technical implementations. We tried different families of models and different sizes of the same model. Consequently, relying only on the parametric knowledge stored in the model's parameters in order to provide the correct answer (and completely neglecting the context) could favour significantly models of greater size or trained on a bigger data mixture.

In all our experiments we will assume that the retrieval is performed perfectly: we will directly append to the prompt the *gold* context (i.e. flagged as relevant) in the dataset.
We will not introduce unnecessary noise by adding deceiving elements in order to test the robustness of the pipeline with respect to wrong context attribution. We plan to run these tests on successive work on this topic.

This choice is made because in this analysis we aim at evaluating the impact of the *way* in which the context is presented to the model. We can face extremely long passages (e.g. the WikiHop dataset possesses many passages of more than 12.000 tokens) in which the relevant information can be found in a couple of sentences only, or on the opposite the passage could directly and explicitly mention the meaningful content in just a couple of terms.

We make the choice of using models of (relatively) small context length, for example `Phi-3` has 4K and `Gemma-2` has 8K context length.

[28] Since we append to the context many items, some information may be lost in the attention process, particularly in presence of very long contexts. This is observed by Liu et al. [43].

[29] By *disentangled* we mean that the inner pipeline steps can be seen as intermediate ones in a bigger reasoning chain; however, the steps' contents are *strongly entangled* due to their dialectic nature.

When we face a passage of greater size (e.g. as in WikiHop), we choose to **summarize** it in a smaller number of tokens before passing the triplet `(question, options, context)` to the pipeline. We will prompt a model to make a summary of the given context knowing the question that will be asked. Further details will be discussed in (4.3.1).

In place of summarizing all the passages' contents, another option could be considered. In scenarios like that, a filtering process could be helpful in speeding up the inference phase and hopefully achieve better results, such as suggested in previous works like System 2 Attention [73].
We preferred using a modified version of MIRAGE (2.10.3) to carefully observe the context elements significant to the output's generation and **extract** all the sentences containing at least one of these elements. Hopefully, this model-internals based form of filtering is more accurate than both a summarization and a selection based on simple prompting as S2A (2.8.3). We will describe more details in (4.3.2).

Since these are all a sort of pre-processing actions that take place before running the pipeline, we can process differently the dataset and observe how this modification impacts on the pipeline. We will test the effectiveness of our pipeline:

- with the relevant (*gold*) context appended;
- with summarized context[30];
- with filtered context, obtained using a modified version of [52].

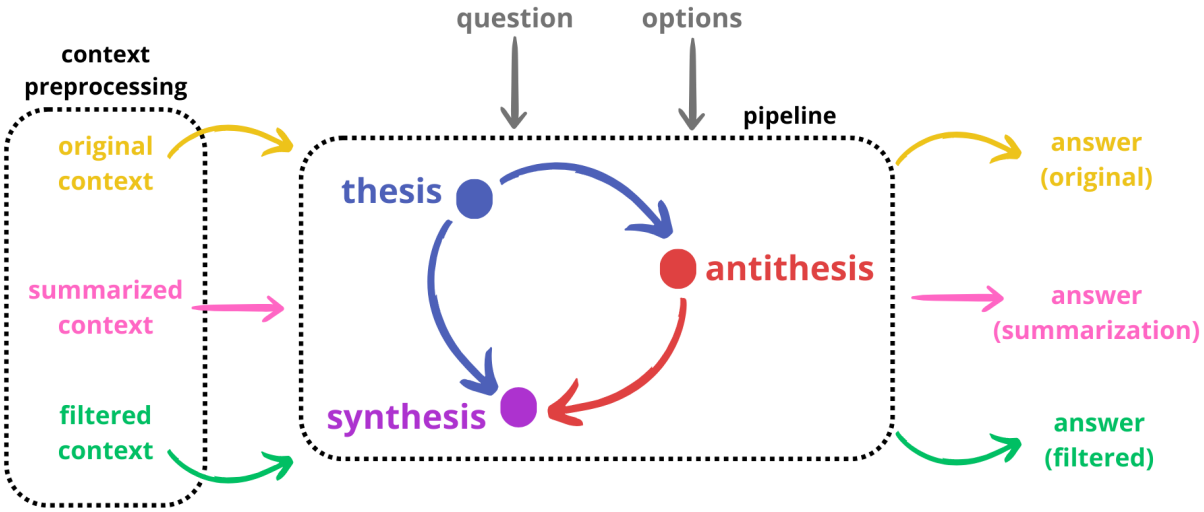


Figure 32: Schematic representation of the different experiments (each one flagged by a different color between yellow, orange and green) originated by different context pre-processing stages. We will compare the answers produced by these different settings, keeping the pipeline fixed.

4.3.1 WikiHop context summarization

As we mentioned above, we choose to summarize contexts that exceed the models' context lengths. In our experiments, this bound is fixed to 4K tokens due to `Phi-3` models.
While both the HotpotQA partitions have passages smaller than this value, WikiHop systematically exceeds the 4K limit.
Consequently, we need to summarize the WikiHop's passages in order to keep them under the given threshold. To perform a uniform compression, we summarize both passages greater and smaller than 4K tokens.

We use still a `Phi-3-mini` model, but this time we opt for its `128k` context length version in order to be able to catch all the contexts, no matter their length. We prompt the pipeline to:

[30]Only on WikiHop, since HotpotQA passages are all under the maximum context length (4K); on WikiHop, we summarized *all* the passages, even those under the limit of 4K tokens, for the sake of comparison.

```
You are a helpful AI assistant. You are given a long document as context and you are asked to
produce a meaningful summary of it. Please summarize in less than 500 words.
```

Besides the requirement of not exceeding the 500 words, we let the model generate up to 1000 new tokens to allow a certain degree of flexibility and not requiring to truncate too harshly the summarization[31]. The other generation parameters are kept fixed, with a temperature of 0.0 and no sampling involved.

We run the summarization process and we make sure that the model completes the task (i.e. finishes to summarize the original passage), and we discovered that this had happened only in certain cases. `Phi-3` family of models flags with `<|end|>` the end of generation, and this is present only in 330 rows of the global 1000.
Instead of relying on a partially summarized source, we prefer to just consider these 330 completed summarizations and use this subset as a pre-processed context.

4.3.2 MIRAGE context filtering

On the opposite, context filtering is performed both on WikiHop and on HotpotQA, since the extraction procedure is independent of the original context length. The goal is to study whether by selecting only the relevant sentences of the passage we are able to observe better *synthesis*' outputs.

MIRAGE (2.10.3) selects all the documents containing at least one *contextual cue* and appends to the generated sentence a list of document identifiers (i.e. citations) from which the generation has been influenced. We use a similar approach, but in place of producing citations we aim at selecting only the relevant sentences inside the context in order to construct a filtered version of it.
This selection will work as context source for our pipeline.

Still, we used `Phi-3-mini` (this time we returned to the 4K version) and import it as an `inseq` model[32] together with the model's tokenizer:

```
inseq_model = inseq.load_model("microsoft/Phi-3-mini-4k-instruct", "saliency")
tokenizer = AutoTokenizer.from_pretrained("microsoft/Phi-3-mini-4k-instruct")
```

where `"saliency"` is the chosen attribution method.

The core function asks the model to:

- tokenize the text passage using the model tokenizer;
- invoke PECoRe [58], i.e. the methods running inside MIRAGE to pair *contextual cues* and their corresponding *context-sensitive* generated tokens;
- return only the sentences containing at least one *contextual cue* in one merged new passage.

Due to the different datasets' passages sizes, we retain improper to ask PECoRe to select k passages and to keep k fixed for all the datasets. Additionally, WikiHop shows a great variability of context sizes, thus a shared value of k would be too low for certain problems and excessively for others.
Thus, we construct an auxiliary function `find_top_p(passage, p)` that takes the passage, tokenizes it and outputs the number of sentences corresponding to the top-p percentile of the source.
Subsequently, we invoke PECoRe with the `invoke_pecore(passage, question, p)` function as detailed in Appendix A and obtain the CCI scores (2.10.3).
We retain the p highest values among these, corresponding to the top-p most influential tokens. These are mapped to their corresponding sentences and another function, `select_passages(passage, question, p, tokens)`, returns the *influential* sentences only.

[31] A general rule of thumb is that 75 words approximately equals 100 tokens, thus for 500 words a maximum of 1000 tokens should suffice.

[32] A Pytorch-based hackable toolkit to allow access to common post-hoc interpretability analyses of sequence generation models [31].

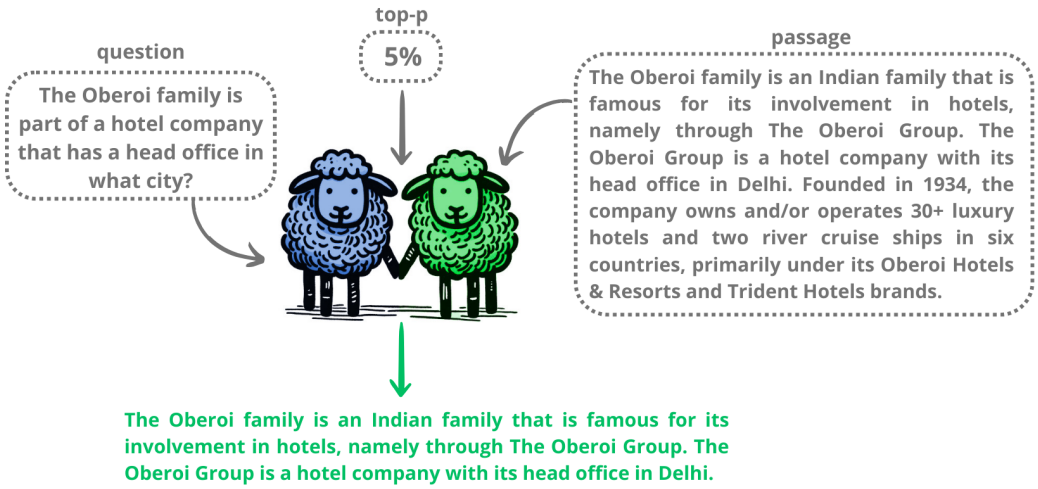


Figure 33: Concrete example taken from HotpotQA (`bridge` partition) of top-5% context selection performed by our modified version of MIRAGE. The two sheep indicate the MIRAGE process running PECoRe attribution method. The green passage is the PECoRe-filtered process.

While for HotpotQA we can take each row (they are all under the threshold of 4K tokens), for WikiHop we still have the same problem described in (4.3.1). Since the object of our analysis is the comparison between the answers obtained with different ways of considering the context, we do not care to apply this procedure to all the WikiHop items.
What we did instead is to select only the WikiHop rows characterized by less than 3500 tokens[33] in the `supports` dataset attribute (i.e. the *original context*). In this way, we select only 92 of the original 1000 rows. We consequently summarize and filter this subset only to make the three approaches comparable.

In order to study whether these results could be generalizable to the entire WikiHop dataset, we compare the performances of the summarized context in this subset (original context under 4K tokens) and the ones observed in the 330 rows previously summarized (4.3.1), that are assumed to be representative of WikiHop.

4.4 Answers format: the `guidance` framework

Even the best pre-trained and fine-tuned language model struggles in instruction following [49], [43], particularly open-source ones. Since in this analysis we do not make use of proprietary models, we wanted to get rid this *format disobedience* and force the model to output exactly one of the candidate options, without *going off on a tangent*.
Given an extensive output pointing out the correct option, we want to recognize it instead of neglecting everything that is not exactly the right-formatted answer. We do not retain reasonable to judge harshly a correct output due to format issues.

Consider the following problem:

```
Question: Which year is Halley's Comet expected to return to the solar system?

Options: [2110, 2045, 2086, 2061]

Context: Astronomers have now linked the comet's appearances to observations dating back more
than 2,000 years. Halley was last seen in Earth's skies in 1986 and was met in space by an
international fleet of spacecraft.
It performs a regular 76-year journey around the Sun.
```

The instruction-tuned model that we will use produce the following answers given an appropriate prompt[34]:

[33] 500 tokens are left out for prompt instructions and few-shots examples.
[34] `You are a helpful AI assistant. You are given a question and the relevant context to answer it.`

- **Phi-3-mini**:

  ```
  2061 <|end|>
  ```

- **Phi-3-medium**:

  ```
  The correct answer is 2061. Halley's Comet is expected to
  return to the solar system in 2061, as it has a regular
  76-year orbit around the Sun. <|end|>
  ```

- **Gemma-2-2b-it**:

  ```
  **2061**
  ```

- **Gemma-2-9b-it**:

  ```
  2061
  <end_of_turn><eos>
  ```

- **Llama-3.1-8b-instruct**:

  ```
  2061.
  ```

The outputted answers are all clearly correct, but their format is not always appropriate to perform a quick comparison between the correct answer (`2061`) and the model's verbose output.

The models have different tokens flagging the end of generation (e.g. `<|end|>` and `<end_of_turn><eos>`, that have to be discarded) and also the punctuation is a problem: it is not trivial to distinguish when a dot flags the end of the sentence (and thus can be removed) or when it is part of the answer (e.g. `2061 A.C.`). If we limit to post-process the output by removing the *end-of-generation* tokens, only `Phi-3-mini` and `Gemma-2-9b-it` would (apparently) return the correct output, and this would be false.

To avoid relying on hand-constructed prompts (different from one model to another, thus without guarantees to correctly extract the suggested option), we exploited *Structured Guided Generation* tools.
Structured Guided Generation (SGG) is a feature that allows users to constrain the generation of a large language model with a specified *grammar*. It is used to generate text that follows a specific structure or uses a specific set of words or produce output in a specific format, e.g. to produce a valid JSON file as output, a function signature, a list of integers.

The `guidance` framework does this by masking out certain all the tokens that do not belong to the pre-specified grammar (that can also be a set of options):

1. the model produces the logits **for each word in the vocabulary**, e.g.

   ```
   [0.1,   0.3,    0.2,    0.25,   0.15]
   ```

2. a mask is created for discarding all the words that do **not belong to the grammar**:

   ```
   [-inf,   0.0,    -inf,    0.0,   -inf]
   ```

   where `-inf` is placed in correspondence of not-allowed words, `0.0` otherwise;

3. the mask is added to the original logits, allowing the model to discard all the forbidden tokens in the *sampling* stage:

   ```
   [-inf,   0.3,    -inf,    0.25,   -inf]
   ```

There are also other tools for performing SGG, such as `outlines` [76] that exploit different technical implementations to determine how the model has to adapt its generation to a set of options. We opted for `guidance` despite the fact that it does require to access models' logits, thus not being used for proprietary, closed-source models such as `OpenAI`. We did not need a tool applicable to those kinds of models, thus we did not find necessary to exploit a different guiding generation strategy.

Practically speaking, we ask the model to generate the first, tentative answer by forcing the model to choose between the options as detailed in Appendix B.

`Answer briefly to the question with just one of the given options` and then we appended the task.
We sampled the output with temperature fixed to 0.2 and maximum number of new tokens set to 50.

4.5 Thesis

Focusing again on the pipeline, we choose to ask the *thesis* step to output the most proper candidate answer to the question.
We prefer not to let the model produce a verbose version which, as we spotted in the previous section, can appear in many different and heterogeneous formats. This could be confusing for later stages of the pipeline, which must handle multiple expressions that semantically refer to the same option.
Think for example of a model that receives in input the answers presented in (4.4): the long answer flagging `2061` as correct option could be handled differently than a shorter answer saying the same thing[35].

First of all, we upload the model and its tokenizer using the `guidance` framework and fix its temperature to 0. The prompting format is imposed by the framework and *concatenates* the **model**, the **prompt** (composed by some generic description on how we want the task to be performed and by the problem) and the **options** (i.e. the only tokens of the model's vocabulary that will not be masked when generating the output).
The code used for the *thesis*' generation can be found in Appendix C.

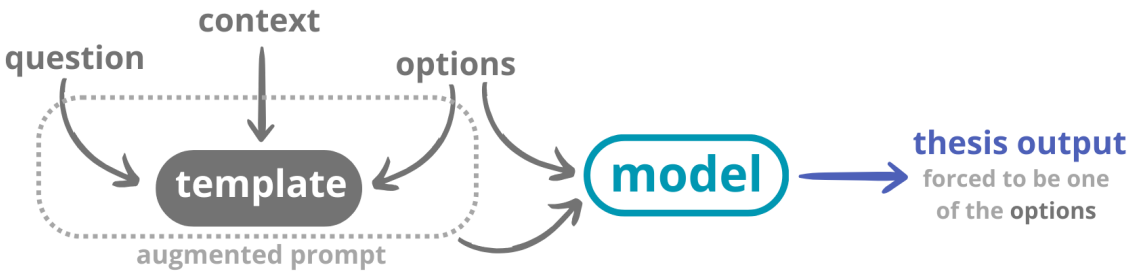


Figure 34: A schematic representation of the `guidance` framework, when applied to the *thesis*' generation.

4.6 Antithesis

4.6.1 The importance of questioning previous statements

The *antithesis* step is the core component of the pipeline, but not because it is the most relevant step[36].
The *synthesis* step also plays a role in modifying the answer, but the *antithesis*' importance stands in the downstream impacts that a powerful correction has.

Think for example of how a teacher can correct an essay: he can only flag the error or he can explain precisely **why** the student's answer is not true.
In the first scenario, the student could try to make a second guess and he could also fail again. In the second one, he will probably correct the first tentative answer with a more robust and factual one that builds on the teacher's suggestion.

Our task is structured slightly differently from a teacher-student dialectic: since we have the same LLM used as actor for all the three steps in the pipeline, we do not have a teacher model that checks whether or not the answer is correct. No step is "more experienced" than the others and consequently no step is more important or reliable than the others.

We should then modify a little the previous metaphor by imagining a student who is given a long multiple-choice test. He will firstly mark the choices that seem to him more proper for the given question. We will refer to this first pass as *thesis*.

Then he will take a small break, refresh his mind, and look again to each question and to each answer, asking himself: "Is it really the most proper answer to the question? Is there a more proper option that I neglected on the first pass?".
He will consequently write some notes near each question explaining why he thinks that a certain answer is the correct one. Note that he is not forced to explain why the previously given answer is the

[35] Maybe a more verbose opinion on which is the correct answer would be considered more reliable than just the option name.
[36] as in `SELF-REFINE`, `SELF-CORRECTION` and Reflexion

correct one, he can also change its mind. Symmetrically, he should enforce its claim on the fact that the previously given answer is the correct one by spending a couple words on it. We will refer to this step as *antithesis*.

Finally, the student will look again at the question, the options, the firstly marked one and the explanation of why he thought that it was correct or not. He will finally (in the *synthesis* stage) opt for the initial option or for a suggested alternative, in a fresh pass of the same task.

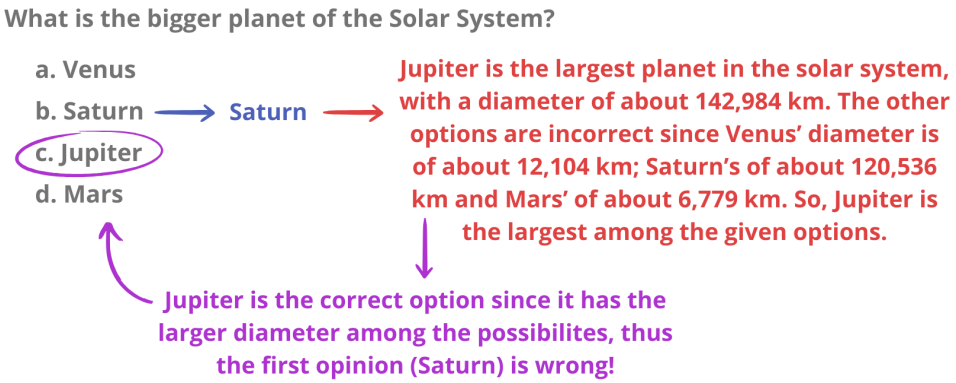


Figure 35: A simple example of the three steps performed: in blue, the *thesis*; in red, the *antithesis*; in purple, the *synthesis*.

Of course, he will have no guarantees that each step is performed correctly. If the second pass (the one in which he writes notes on which should be the correct option and why) is performed incorrectly, the overall process will suffer: the (eventual) critique is fundamental to change opinion on the correct answer. If we limited to this *antithesis* step and we took its suggested output as the correct one, we would rely excessively on a single component that is prone to errors. By mitigating its effect as described in (4.2), we make sure not to make the model *jumping to conclusions*.

The goal of our experiments was to make the *antithesis* step as more accurate as possible. Of course we perform a final check, but we would like to avoid scenarios in which the *thesis* is correct, the *antithesis* is incorrect and the *synthesis* agrees with the wrong suggestion.
This led us to the choice of providing **access to the relevant context** to each step of the pipeline. In this way, the student has always access to the book from which the topics of the multiple-choice exam are taken, and he just has to *reason* on the sources and compare the options wisely.

Consider for example the Halley's Comet example provided in (4.4): even by accessing the source, the model needs to:

- filter out irrelevant information;
- consider the last Halley's Comet passage (`1986`);
- understand that it performs a regular journey, so the `76-year` information should be exploited;
- perform `1986 + 76` and output the correct answer.

The model capabilities have to go further than simply summarizing or rephrasing the context for multi-hop tasks like the ones we are going to test. On the other hand, if we do not provide the model the relevant information, the given answers could be incorrect. Consequently, we consider the addition of the relevant passages a beneficial step to elaborate a detailed description of which the correct option is during the *antithesis* stage.

As we mentioned in (4.2), our method does not require any form of task-specific prompt or fine-tuning. We just append an example describing how we want the process to be performed in a one-shot setting. We chose not to increase the number of exemplars provided since the prompt is already burdened by the question, the options, the (often long) context and the *thesis*' answer. Additionally, the one-shot prompt

was enough to produce the desired behaviour, thus increasing it would be pointless.

The code used to generate the antithesis' suggestion can be found in Appendix **??**. Differently from *thesis*, the *antithesis* is not forced to be exactly one of the options through the use of the `guidance` framework. What we did instead was to define a generation pipeline:

```
pipeline = pipeline("text-generation", model=model, tokenizer=tokenizer)
generation_args = {"max_new_tokens": 500, "return_full_text": False, "do_sample": False}
```

Consequently, the generated output will be in a discursive format. Consider for example this task, where `Candidate answer` contains the *thesis*' answer:

```
Question: Which magazine was started first, Arthur's Magazine or First for Women?

Options: [Arthur's Magazine, First for Women]

Candidate answer: First for Women

Context: Arthur's Magazine (1844-1846) was an American literary periodical published in
Philadelphia in the 19th century. Edited by T.S. Arthur, it featured work by Edgar A.
Poe, J.H. Ingraham, Sarah Josepha Hale, Thomas G. Spear, and others. In May 1846 it was
merged into Godey\'s Lady\'s Book. First for Women is a woman's magazine published
by Bauer Media Group in the USA. The magazine was started in 1989.
It is based in Englewood Cliffs, New Jersey. In 2011 the circulation of the magazine
was 1,310,696 copies.
```

The *antithesis* will produce[37] the following opinion:

```
The correct answer is 'Arthur's Magazine' as it was started first in 1844, while 'First
for Women' was started later in 1989.
```

As it is easy to spot, the *antithesis* suggests the correct option by taking a second look to the context, even though the *thesis* failed.

4.6.2 The influence of the given examples

In the previous section we passed the one-shot example as an input parameter of `create_message_antithesis()`. We now want to spot a light on the **impact that the one-shot example** has on the generated *antitheses*.

We experimented with different settings, each one stressing different behaviours that the *antithesis* could replicate.

- The **baseline**: we simply provide the prompt with an example of how we want the model to make use of the *thesis*' answer (referred to as `candidate answer` inside the prompt):

  ```
  Question: What is the sun, a star or a planet?
  Options: ['a star', 'a planet']
  Candidate answer: a planet
  Context: The Sun is the star at the center of the Solar System. It is a massive,
           nearly perfect sphere of hot plasma, heated to incandescence by nuclear
           fusion reactions in its core, radiating the energy from its surface mainly
           as visible light and infrared radiation with 10% at ultraviolet energies.

  Assistant: The correct answer should be 'a star' due to the fact that the context
             explicitly say so. On the opposite, the context never mentions the fact
             that the Sun could be a planet.
  ```

  Two things have to be spotted: first of all, the model does not firmly state which is the correct answer. We induce the model to choose a neutral `should be` in place of a more confident expression. The second thing is that the answer does not mention explicitly the context, but instead it limits to motivate a certain choice by stating that this is due to the context. This baseline makes an implicit reference to the effective passage, does not quote it directly.

[37] Real output, obtained using `Phi-3-mini`.

- The **`is` variant**: this one-shot example differs from the previous one just in the degree of confidence used to answer: the previous `should be` is replaced with a more confident `is`:

  ```
  Question: What is the sun, a star or a planet?
  [...]

  Assistant: The correct answer is 'a star' due to the fact that the context
             explicitly say so. On the opposite, the context never mentions the fact
             that the Sun could be a planet.
  ```

  This aims at experimenting whether the level of confidence with which the model provides the *antithesis* influences the *synthesis*' decision on which option should be the correct one.

- The **`cot` variant**: uses a Chain-of-Thought inspired answer:

  ```
  Question: What is the sun, a star or a planet?
  [...]

  Assistant: Let's consider the options and check whether or not they are correct.
             The context clearly identifies the Sun as 'the star at the center of
             the Solar System', thus 'a star' is probably the correct option.
             On the opposite, 'a planet' is not mentioned in the context, thus it is
             unlikely to be the correct option. Therefore, the correct option is 'a star'.
  ```

  This approach considers each option and seeks eventual support of their correctness in the context. In the example, the relevant part of the context is also **quoted**. Finally, a conclusion is reached on which is the most proper answer among the candidates.

- The **`ctx` variant**: in the wake of the previous experiment, we tested also a different approach that is more context-grounded:

  ```
  Question: Jane's Addiction and Weeping Willows, play which genre of music?
  Options: ['indie', 'rock']
  Candidate answer: rock
  Context: Weeping Willows is a Swedish indie rock group that started in 1995. Jane's
           Addiction is an American rock band from Los Angeles, formed in 1985.
           The band consists of Perry Farrell (vocals), Dave Navarro (guitar), Stephen
           Perkins (drums) and Chris Chaney (bass).

  Assistant: The context mentions that Weeping Willows is a 'Swedish indie rock group' and
             Jane's Addiction is an 'American rock band'. Both bands are associated with the
             'rock' genre, thus the correct answer is 'rock'.
  ```

  This is still a CoT-like approach to the problem, since the model decomposes the task in intermediate steps; but differently from the `cot` variant, this time the assistant is shown a different way of dealing with the problem.

  The model quotes the context's elements (thus the `ctx` term to identify this variant) that are relevant for each compared item, and consequently decrees which is the correct answer. This is different from before, where partial conclusions were drawn analyzing the correctness of each option separately. With this variant we wanted to test whether explicitly quoting the relevant context before answering is beneficial to the pipeline.

The one-shot example provided is different from the previous one for two main reasons:

- to assess the impact that different prompt examples have on the generated outputs (thus ensure that the results are not prompt-dependent);
- because the first example is relatively easy: the context does not mention both the candidate answers, but just the correct one. This could potentially cause the model to *learn an incorrect pattern*[38], since in the multi-hop tasks we face the context systematically mentions the options.

[38] We used the term *pattern* referring to a similar use of the word present in the Chain-of-Thought prompting paper [68], stating that *large language models can learn via a few examples with natural language data about the task (c.f. automatically learning the **patterns** underlying inputs and outputs via a large training dataset).*

The task that will be tested on this pipeline is *multi-hop* question answering, thus each option will be grounded in some elements of the context and the implicit challenge of the model is to actually merge multiple elements in order to provide a reasonable answer.

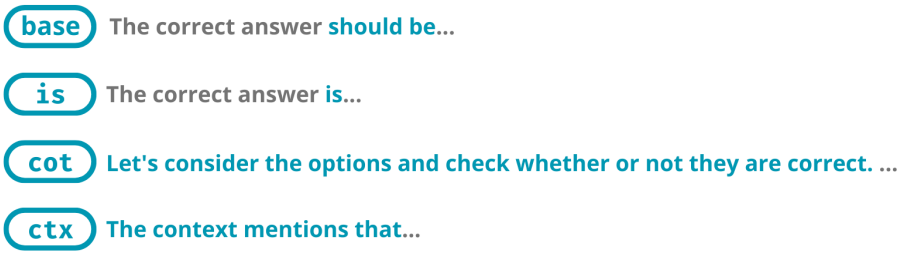


Figure 36: A brief overview of the *antithesis* variations tested in our analysis.

Here we show an example of how the antithesis output changes with respect to the king of one-shot example that is provided:

- baseline:

  ```
  The correct answer should be 'Arthur's Magazine' as it was started first in 1844, while
  'First for Women' was started later in 1989.
  ```

- `is` variant:

  ```
  The correct answer is 'Arthur's Magazine' as it was started first in 1844, while 'First
  for Women' was started later in 1989.
  ```

- `cot` variant:

  ```
  Let's analyze the options and the context provided. The context states that Arthur's
  Magazine was published from 1844 to 1846, while First for Women was started in 1989.
  Therefore, Arthur's Magazine was started first. The candidate answer 'First for Women'
  is incorrect. The correct option is 'Arthur's Magazine'.
  ```

- `ctx` variant:

  ```
  The context states that Arthur's Magazine was published from 1844 to 1846, while First
  for Women was started in 1989. Therefore, Arthur's Magazine was started first.
  The correct answer is 'Arthur's Magazine'.
  ```

As it is easy to notice, there is no significant difference between `cot` and `ctx`, since both the one-shot examples decompose the problem into a retrieval step and a conclusion step. The most remarkable difference (present also in different problems, not reported here) is that the `cot` variant tends to remark the fact that the other option is wrong.

One last device we had was to change the style of the question in the one-shot example for datasets other than the `comparison` fraction of HotpotQA. This is because, while all the `comparison` questions are formulated in such a way as to explicitly report the options to choose from, the `bridge` questions (i.e. the complementary fraction of HotpotQA) and WikiHop reflect a less "set" style:

- HotpotQA, `comparison`:

  ```
  Which magazine was started first, Arthur's Magazine or First for Women?
  ```

- HotpotQA, `bridge`:

  ```
  The Oberoi family is part of a hotel company that has a head office in what city?
  ```

- WikiHop:

  ```
  What language did John Osteen speak or write?
  ```

Consequently, we will replace

`What is the sun, a star or a planet?`

with a simpler

`What is the sun?`

On the opposite, `ctx` is already more similar to the simpler question, thus is left unchanged.

4.7 Synthesis

The final step aims at merging in a reasoned answer the entire pipeline: by considering the first tentative answer (the *thesis*) and the new opinion (the *antithesis*) obtained by reconsidering the previous one, the last opinion (the *synthesis*) has to decree which is the correct option. And now it should do this with more confidence and increased factuality.

As in the previous pipeline steps, the detailed code is reported in Appendix E. It is important to remark that the *synthesis*' prompt does not bound the model to choose between the option proposed by the *thesis* or proposed by the *antithesis*: the instruction given clearly states that the *synthesis* stage has to choose between one of the `available options`. This means that the *synthesis* stage could potentially even opt for an unseen option.

For the baseline and the `is` and `cot` variants, the one-shot prompt is very similar:

```
Question: What is the sun, a star or a planet?/What is the sun?
Options: ['a star', 'a planet']
Candidate answer: a planet
Suggestion: {antithesis_answer}

Context: The Sun is the star at the center of the Solar System. It is a massive, nearly
         perfect sphere of hot plasma, heated to incandescence by nuclear fusion reactions
         in its core, radiating the energy from its surface mainly as visible light and
         infrared radiation with 10% at ultraviolet energies.

Assistant: the correct option is 'a star', since the suggestion is grounded in the context,
           even if the candidate answer does not agree.
```

where `antithesis_answer` is the antithesis, presented in different formats with respect to the kind of variant (i.e. what follows the `Assistant:` in the previous section's one-shot examples).

For the `ctx` variant, we tried instead to use two-shot prompting in order to induce the model to learn how to deal with situations in which the *thesis* and the *antithesis* agree and situations in which they don't[39]:

```
Question: Jane's Addiction and Weeping Willows, play which genre of music?
Options: ['indie', 'rock']
Candidate answer: rock
Suggestion: The context mentions that Weeping Willows is a 'Swedish indie rock group' and
            Jane's Addiction is an 'American rock band'. Both bands are associated with the
            'rock' genre, thus the correct answer is 'rock'.
Context: Weeping Willows is a Swedish indie rock group that started in 1995. Jane's
         Addiction is an American rock band from Los Angeles, formed in 1985.
         The band consists of Perry Farrell (vocals), Dave Navarro (guitar), Stephen
         Perkins (drums) and Chris Chaney (bass).

Assistant: Both the candidate answer and the suggestion agree on the fact that the correct
           option is 'rock'. Let's check on the context whether or not this is correct.
           Weeping Willows is an indie rock group, thus they make rock music; Jane's
           Addiction is a rock band. Consequently the context confirms that the genre
```

[39]The `ctx` variation originates from the observed results and the fact that the in-context learning example is different is a conscious choice. Additionally, the following synthesis setting (different from the `cot` one) is oriented to a study on different few-shot options.

```
          performed by both bands is 'rock'. The correct option is 'rock'.

Question: Between two tennis players Kim Clijsters and Mary Pierce, who is older?
Options: ['Kim Clijsters', 'Mary Pierce']
Candidate answer: Kim Clijsters
Suggestion: The correct answer is 'Mary Pierce' as she was born on 15 January 1975,
           which is earlier than Kim Clijsters who was born on 8 June 1983.
Context: Kim Antonie Lode Clijsters (] ; born 8 June 1983) is a Belgian
         former professional tennis player. Clijsters is a former world No. 1 in both
         singles and doubles. Mary Pierce (born 15 January 1975) is a French retired tennis
         professional who played on the Women's Tennis Association (WTA) tour.
         Born in Canada, she is a citizen of Canada, and the United States.
         Pierce played for France in team competitions and in the Olympics.

Assistant: The candidate answer says that the older tennis player is Kim Clijsters, while
           the suggestion indicates Mary Pierce. The context provides the birth dates of
           both players, thus I can check which of the two options is correct.
           Kim Clijsters was born on 8 June 1983, and Mary Pierce was born on 15 January
           1975; By comparing these dates, it's clear that Mary Pierce
           is older than Kim Clijsters. Thus the correct option is 'Mary Pierce'.
```

The important notice that has to be made is that we have no guarantees that the *synthesis* step chooses the right option. It could be deceived by a wrong suggestion of one of the two previous steps or it could choose another wrong option. We have already spent a couple of words explaining why a last checking step could be influential in terms of performances of the pipeline, but our experiments will confirm this hypothesis.

4.8 Assessing performances

Up to now, we had widely described each step's details and variations, but still it is unclear how a verbose synthesis output (i.e. not just the correct option, but the explanation of why a certain option is correct) can be translated into the corresponding option.

In (4.4) we showed how the Large Language Models employed in this analysis are usually *format disobedient* when asked just to output the correct answer to the question. Consequently, we forced the generation of exactly one of the options using the `guidance` framework, so that just the chosen option is passed to later stages of the pipeline.

In the *antithesis* part, what we wanted instead is that the model is given the opportunity to persuade the *synthesis* that the *thesis*' tentative answer is not correct.
In these terms, it would be rough to just provide to the *synthesis* two different options and let it choose which is the most appropriate without any additional information. In fact, in a scenario like this, it would be enough to prompt the large language model to reason about different candidates before outputting the answer. Consequently, the computation required to obtain the *thesis*' and the *antithesis*' answers would be senseless.

If the *synthesis* receives instead a comment (the *antithesis*) on the first tentative answer (the *thesis*), suggesting which should be the correct option, it takes as input a new vision on the problem that could lead to more grounded answers. This is what is done in the multiple-choice test example that we made in Figure 35.

Additionally, we allow the *synthesis* step to question the *antithesis* suggestion, since we give to it all the required elements to answer autonomously to the question. Thus, the *synthesis*' output will not be forced to be a `guidance` output, instead it will be few-shot learned to be an explanation on which is the correct answer and why, as reported in the multiple examples of the previous section (4.7).

In order to practically determine which is the suggested option without the need to "read" the complete justification, we added a post-processing extraction procedure such that the content is properly extracted and reduced to one of the candidate answers. To assess the impact of each step of the pipeline, we choose to extract also the *antithesis*' suggested answer, not just the final (i.e. the *synthesis*') one.

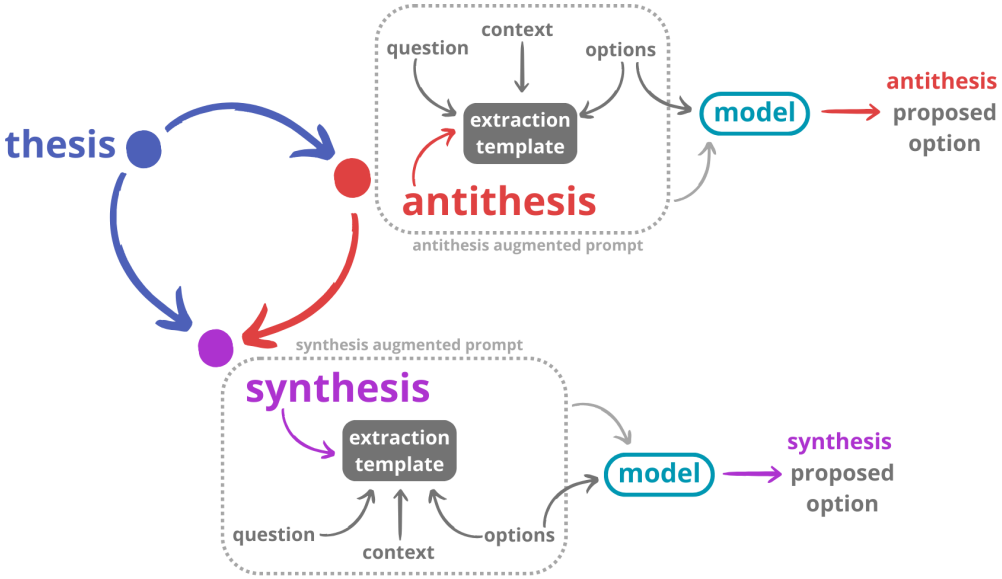


Figure 37: How the extraction process is performed for the *antithesis* and the *synthesis* stages. While the *thesis* step simply constrains the generation, for these two steps we have to extract the chosen option.

We choose to automatize this process using `Phi-3-mini` for all the generated outputs[40]. We use the following prompt template:

```
You are a multiple-choice question answering assistant.
Choose the most proper option between {options} that best matches with the suggestion.

Question: {question}
Suggestion: {antithesis}/{synthesis}
Sources: {context}

Assistant:
```

that is subsequently augmented with the proper items forming an `augmented_prompt`. Finally, the `guidance` framework is employed to be sure that the answer comes in the right format:

```
def optionExtraction(question, options, suggestion, sources):
    # augmented_prompt is produced
    answer = guidance_model + augmented_prompt + select(options)
    return answer
```

Kojima et al. [39] used a similar post-processing stage for their zero-shot Chain-of-Thought: although they did not use a structured guided generation tool as `guidance`, they performed a post-processing phase aimed at extracting the correct answer from a longer one.

We make sure that this post-processing approach faithfully outputs the suggested option and not the correct one. By providing the model with the `question`, the `options` and the `context` we could think that this final stage could modify the pipeline's answer.
Consider as an example the following task:

```
Who was born first, Pablo Trapero or Aleksander Ford?
```

the *thesis* proposed the correct answer (i.e. `Aleksander Ford`), while the *antithesis* affirmed that:

```
The correct answer should be 'Pablo Trapero' as he was born on 4 October 1971,
while Aleksander Ford was born on 24 November 1908.
```

the proposed post-processing technique extracts `Pablo Trapero` as the antithesis' proposed answer, even if incorrect and in contrast with both the *thesis*' and the *synthesis*' choices.

4.9 Comparison with Chain-of-Thought prompting

We want to compare the answers provided by our method with respect to Chain-of-Thought prompting, as it is well-renowned to perform remarkably well on reasoning tasks. Chain-of-Thought is prompted a single time to reason about a task in a way that resembles the few-shot example provided. On the opposite, our pipeline extends this approach by prompting the same model multiple times, passing to each step the output of the previous ones (when present).

We try to make the comparison *as fair as possible*, i.e. we provide the same few-shot examples and we extract the CoT answer also with the `guidance` framework (thus no format issues should be present).

In detail, the one-shot example is the same given in the *antithesis*' `cot` variant (4.6.2), except for a couple of details: the lack of the *thesis*' first tentative answer and the corresponding instruction in the general prompt (i.e. `You are asked to ...`). The Chain-of-Thought prompt is the following:

```
You are an helpful AI assistant. You are asked to determine the most correct answer for
a given question provided a set of possible options. Your goal is to decree which is
the most correct answer to the question between the available options.
```

[40] In order not to introduce biases dependent on each model's different abilities to follow instructions; we observed empirically (running a couple of tests) that, while `Phi-3-mini` is always reliable and accurate, the `Gemma` family often disattends the requirements and outputs random options instead.

```
Here's an example of how to do it:
Question: What is the sun, a star or a planet?
Options: ['a star', 'a planet']
Context: The Sun is the star at the center of the Solar System. It is a massive,
         nearly perfect sphere of hot plasma, heated to incandescence by nuclear
         fusion reactions in its core, radiating the energy from its surface mainly
         as visible light and infrared radiation with 10% at ultraviolet energies.

Assistant: Let's consider the options and check whether or not they are correct.
           The context clearly identifies the Sun as 'the star at the center of the
           Solar System', thus 'a star' is probably the correct option.
           On the opposite, 'a planet' is not mentioned in the context,
           thus it is unlikely to be the correct option. Therefore, the correct
           option is 'a star'.

Now do the same for the following question:
Question: {question}
Options: {options}
Context: {context}

Assistant:
```

Thus the `cot` variant and the Chain-of-Thought approach are really similar.
The big difference stands in where the effective answer extraction takes place: while in the Chain-of-Though approach is directly extracted from the previous prompt's output, the pipeline requires an extra step to be performed (the *synthesis*) before decreeting the most correct option.

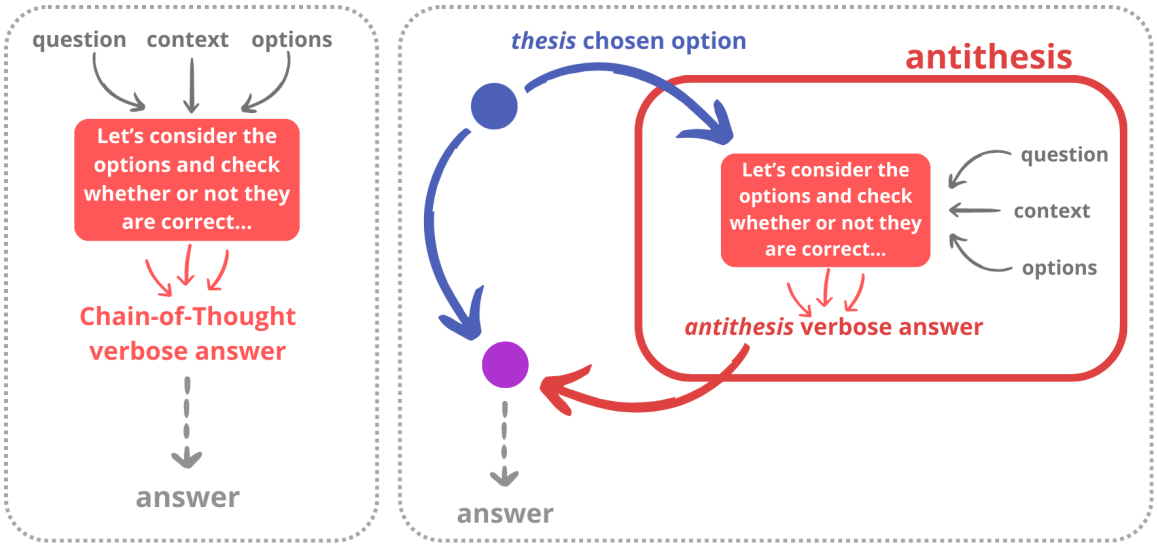


Figure 38: Comparison between the answer generation using the Chain-of-Thought approach (left) and `cot` variant of the pipeline (right).

5 Results

In this final stage, we will report our experiments' outputs that try to answer the following research questions:

- Does the pipeline have a beneficial effect in improving the answers' correctness?
- How do the different families of models perform under the same pipeline settings?
- Is the pipeline robust with respect to different datasets?
- Which is the impact of pipeline variations (4.6.2)?
- Is the proposed method more effective than Chain-of-Thought prompting?
- Do pre-processing sources as detailed in section (4.3) have positive impacts on downstream pipeline performances? Which are the differences that can be observed between them?

5.1 Does the dialectic pipeline work?

Before asking any different question, the *core*, preliminary answer that is necessary to conduct further studies is whether or not the dialectic pipeline works. By *works* we mean that we are able to observe a significant improvement with respect to the standard prompting output.

The two generated outputs that we are going to compare in this first step are the *thesis*' answers and the *synthesis*' ones. Note that, for how we defined the *thesis* step (C), this first tentative answer is in fact the final output of a standard prompting procedure. The *thesis* considers the question, the options and the supporting context and produces the most proper answer given these elements. The `guidance` framework ensures that the output possesses a clear format: no issues regarding failure to recognize a correct answer but in the wrong format will arise.
We choose to append the relevant context to this step (although not strictly necessary, since the *thesis* is in fact a baseline) in order to make a fair comparison between a *context-enriched* pipeline and a standard model generation, that should reasonably also be provided with the relevant passages.

We are going to consider the `comparison` partition of the HotpotQA dataset and we will exploit the `baseline` pipeline setting (4.6.2), running `microsoft/Phi-3-mini-4k-instruct` (2.3.2) as model.

Results show that, while the *thesis*' guesses the correct option in the 53.4% of the cases, the (`baseline`) pipeline reach a 80.7% of correct guesses. The 27.3% improvement is sufficiently significant to justify the claim that the pipeline works (even in the baseline setting).

5.2 How models' architectures and number of parameters impact on pipeline performances

The second question of our analysis is whether this improvement is consistent when we test other models than `microsoft/Phi-3-mini-4k-instruct`.
We additionally try `microsoft/Phi-3-medium-4k-instruct` (similar architecture than `Phi-mini`, but has 14B parameters instead of 4B), `google/gemma-2-2b-it`, `google/gemma-2-9b-it` (2.3.1) and `meta-llama/Meta-Llama-3.1-8B-Instruct` (2.3.3).

Model	`Phi-mini`	`Phi-medium`	`Gemma-2B`	`Gemma-9B`	`LlaMa-8B`
Thesis' correct answers (%)	53.4	50.0	52.8	59.7	48.3
Synthesis' correct answers (%)	80.7	89.5	81.8	88.1	87.2
Absolute improvement (%)	27.3	39.5	29.0	28.4	38.9

Focusing on ***thesis***' performances, it is easy to spot that `Phi-mini` and `Gemma-9B` are the best overall. While it is reasonable that `Gemma-9B` outperforms `Gemma-2B` (since an increased number of parameters often corresponds to more capable models [8]), what is truly surprising is that `Phi-mini` shows better performances than `Phi-medium` (having 14B parameters, 10B more parameters than the `mini` version!). This is in fact something that the `Phi-3` research group observed:

some benchmarks improve much less from 7B to 14B than they do from 3.8B to 7B, perhaps indicating that our data mixture needs further work to be in the "data optimal regime" for 14B parameters model.

The `Phi-3` family of models is pre-trained on "textbook data" [1] that allows to reduce the training corpus and break the *scaling laws* [37]. But if the data mixture is not optimal (i.e. do not possess a certain level of data quality required to reduce the corpus size) then the performances would be poorer than expected.

It is also necessary to point out that `Gemma-2B` is not far from `Phi-mini`, although in practice it has half its parameters: 2B and 3.8B respectively, with a performance gap of just 0.6%, corresponding to 2 wrongly predicted answers only[41].

Another surprising result is that `LlaMa-8B` is the worst performing model overall if we limit to prompt the model in the standard way.

Switching now to the ***synthesis***' results, the best overall pipelines are obtained using `Phi-medium` and `Gemma-9B` models, although even `LlaMa-8B` reaches a similar result.
We can observe that models performing worse than others in the thesis step are able to largely bridge the initial gap through the pipeline, reaching the best overall performances (this is the case of both `Phi-medium` and `LlaMa-8B`). `Phi-medium` beats by a small margin (1.5%, corresponding to 5 differently guessed answers).

The smaller models reach lower accuracy values than their correspondent greater options, even not by a large margin. Additionally, `Gemma-2B` beats by 1% `Phi-mini`, despite being almost half the size of the second one.

With these experiments, we can safely assess that, at least on HotpotQA `comparison`, the pipeline works for different families of models and improves by a large margin the baseline. `Phi-mini` obtains the worst performance increase overall, and despite this it is able to score a really good improvement.

5.3 Robustness with respect to different datasets

In the previous section we clearly expressed how our experiments made sure the pipeline's effectiveness on the HotpotQA `comparison` subset.
Now we want to assess whether it is true that this statement could be generalized to other *multi-hop* datasets. We test the same pipeline setting also on the `bridge` partition of the same dataset and on the WikiHop dataset. In this section we will be considering the WikiHop version with summarized contexts, i.e. the 330 questions mentioned in (4.3.1).

Dataset	% correct	`Phi-mini`	`Phi-medium`	`Gemma-2B`	`Gemma-9B`	`LlaMa-8B`
HotpotQA `comparison`	Thesis:	53.4	50.0	52.8	59.7	48.3
	Synthesis:	80.7	89.5	81.8	88.1	87.2
HotpotQA `bridge`	Thesis:	52.1	56.0	55.7	66.5	49.8
	Synthesis:	87.9	90.2	81.3	88.9	91.9
WikiHop	Thesis:	12.7	13.6	17.7	16.9	12.4
	Synthesis:	33.0	40.7	28.3	21.1	37.7

These results confirm the trend that we observed for the `comparison` partition of HotpotQA.
If we consider the `bridge` partition of the same dataset, we can recognize similar *thesis* and *synthesis* accuracy values, although they tend to be better than those of `comparison`. `Phi-mini` and `LlaMa-8B` show nearly identical *thesis* percentages of correct answer in both `comparison` and `bridge` partitions, while `Phi-medium`, `Gemma-2B` and `Gemma-9B` perform better by a certain margin in this second subset. We have to consider that while we just considered 352 questions from `comparison`, the `bridge` partition contains 1000 items. Thus, these differences[42] could be originated:

- the different dataset sizes, since a more various and wide datasets could provide a more reliable value of models' predictive abilities than a smaller one;

[41] The `comparison` partition of the dataset consists in 352 questions.
[42] 6%, 2.9% and 6.8% in `Phi-medium`, `Gemma-2B` and `Gemma-9B` respectively

- by the automatic generation of the alternative option for `bridge`, as described in (3.1.2) that could make its tasks easier to solve than hand-checked ones, in particular for more capable models. This would explain the greater gap that we observe for `Phi-medium` and `Gemma-9B`.

What we can infer from these results is that greater size models (i.e. `Phi-medium`, `Gemma-9B` and `LlaMa-8B`) perform remarkably well, as we observed for `comparison` already. The interesting difference is that this time `Phi-mini` outperforms by a large margin `Gemma-2B`: that (apparently small) 6.6% gap corresponds to 66 wrongly answered questions from the latter model. This leads us to the idea that `Phi-mini` is more capable than `Gemma-2B` at merging multiple sources to determine which is the correct solution. When we ask instead to look at two different sources and compare them, the 2B model performs better than the other one. Maybe this difference is due to the limited abilities of a model of that small size, and for *bridge* type of tasks it may be preferable to use a greater model.

The WikiHop dataset confirms the pipeline effectiveness, despite the reduced accuracy values that can be observed both for the *thesis* and for the *synthesis* percentage of correct answers. In this case, the number of candidate options is not fixed to two and neither is the number of *hops*. Instead, WikiHop presents a number of options up to 60, even though limited to a few questions, and a set of passages with a big variety in terms of number of *hops* (3.3).
Due to this significant difference, the already tested models behave differently than before. While `Phi-mini`, `Phi-medium` and `LlaMa-8B` improve the *thesis* by large margins (each of these improves the baseline more than twice), the `Gemma-2` family of models seem to struggle with this various set of tasks.
Surprisingly, the 2B version overcomes the 9B one by a solid 7.2%.

Below is reported the table of absolute improvements in accuracy observed between the *thesis* and the *synthesis* steps of the pipeline. Results are reported with respect to the considered dataset (`comparison` and `bridge` omit the HotpotQA prefix) and the model used inside the pipeline.

Dataset	Phi-mini	Phi-medium	Gemma-2B	Gemma-9B	LlaMa-8B
comparison	27.3	39.5	29.0	28.4	38.9
bridge	35.8	34.2	25.6	22.4	42.1
WikiHop	20.3	27.1	10.6	4.2	25.3
Overall mean	*27.8*	*33.6*	*21.7*	*18.3*	*35.4*

The first conclusion that we can draw is that for all the cases we considered, our proposed pipeline always causes positive improvements in accuracies; we can consider the mean on all the dataset as a synthetic measure supporting this statement.

Despite its being effective, this level of positive impacts is model and dataset dependent. WikiHop is proved to be more challenging due to its multi-hop variety and the big number of options between which the models have to choose the most proper answer.
Despite its being really challenging, the pipeline often reaches large margins of improvements (more than 20%) for two families of models, `Phi-3` and `LlaMa-3.1`. The `Gemma-2` family clearly struggles in absorbing meaningful information from the pipeline, while being the most capable family when considered the *thesis* performances only.
This could be due to the parametric knowledge of `Gemma-2` models, maybe richer and more various than the "texbook" knowledge of `Phi-3` and of the `LlaMa-3.1` one.
Considered this, we will prefer to analyze the *synthesis*' accuracy values as indicators of the pipeline effectiveness rather than focusing on the relative improvement observed with respect to the *thesis*' ability of finding the correct question at the first try.
The minor tendency of `Gemma-2` to follow the instructions given (i.e. use context to answer) is easily observed when we inspect the responses produced by this in the *antithesis* phase, compared with those of the other two models:

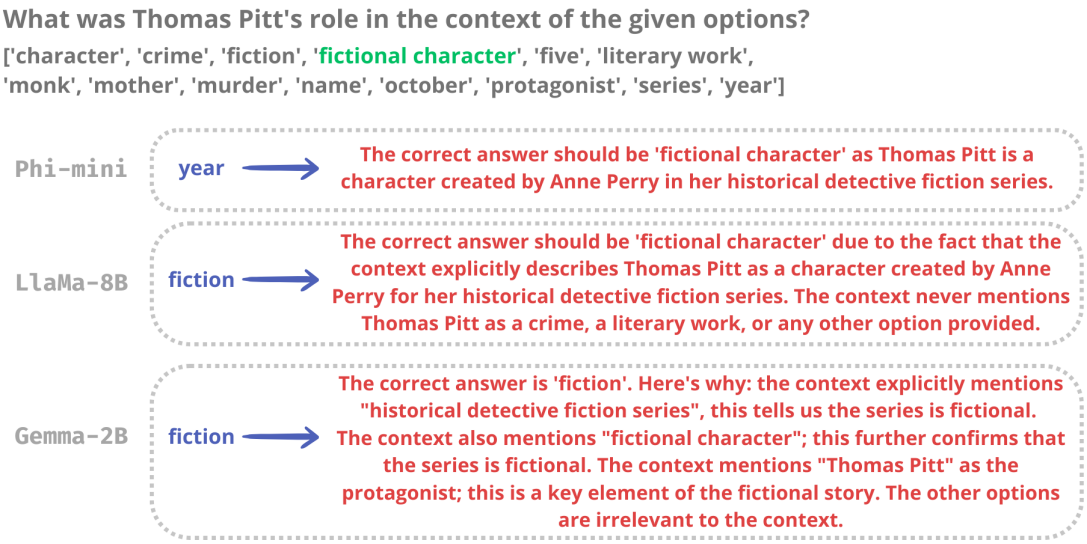


Figure 39: Comparison between the *antithesis*' answers as given by these three families of models; the *thesis*' proposals are colored in blue. Note that the format should follow the one-shot example provided in (4.6.2), thus `Gemma-2B` turns out to be poorer than the other models in faithful instruction following.

It would be untrue to state that `Gemma-2` models' poor ability to replicate faithfully the given few-shot examples (and being more "conversational" rather than context-grounded) is harmful in all the cases. In HotpotQA, where the options are just two, this issue could not be a real problem. In fact, the absolute improvements observed for `Gemma-2` models on HotpotQA are large (although smaller than other models' margins). The necessity to attain to instructions is instead crucial in more complex task.

This further hints that the real improvement given by the pipeline is not given by its multi-step nature, rather by the *reasoned solving process*. By breaking down the problem into units performing multiple checks from different perspectives, even small models (i.e. with less than 20B parameters) can achieve good performances.

5.4 Pipeline variations

In the previous section (5.3) we highlighted how the `Gemma-2` tendency to approach the *antithesis* step differently from what it has been shown from the one-shot example could explain worse prediction abilities (at least on challenging tasks).
The following experiments are made to assess how much the way in which we prompt the model has an impact on the final performances. In practice, we substitute the one-shot example provided and study how much this modification changes the correctness of the final predicted output.

We start from the `comparison` partition of HotpotQA and test the differences between the baseline pipeline settings and the `is`, the `cot` and the `ctx` variants (4.6.2). This comparison is performed for all the models that we already considered.

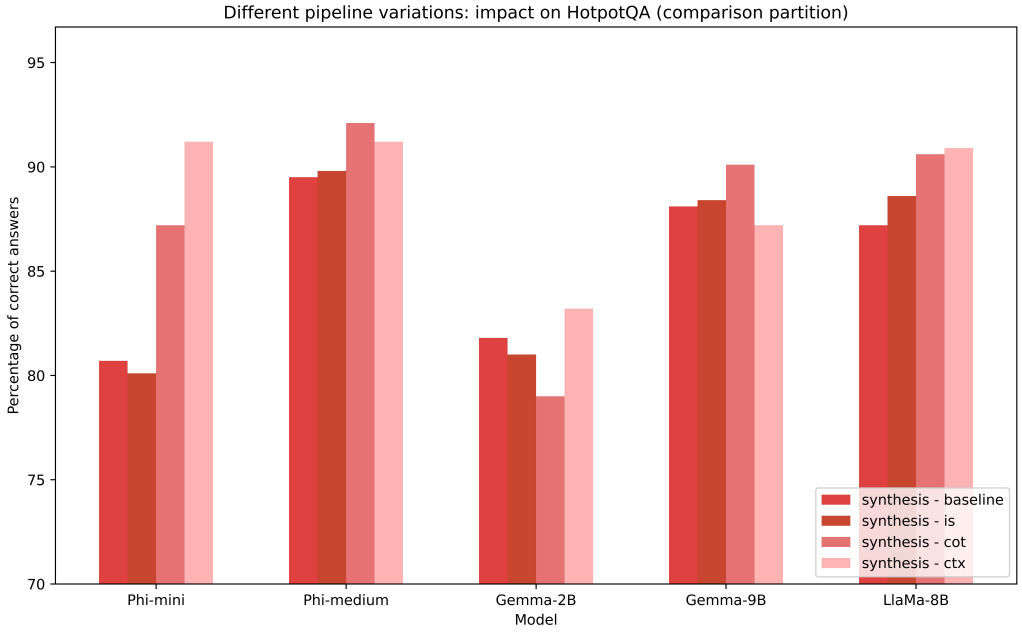


Figure 40: Different percentages of correct answers obtained using a particular pipeline setting **on HotpotQA - `comparison`**. The plot y-axis is reduced (starts from 70%) to highlight the small differences between the different one-shot examples.

The first comment that we can make is that there is a negligible difference (under the 1% for all the models) between the baseline and the `is` variation. We can safely affirm that the level of confidence with which the *antithesis* is provided is not relevant in terms of performance. Thus, the *antithesis* persuades the pipeline through factual knowledge, not by a more confident style of suggestions.

The `cot` variant seems to outperform the baseline when the model with sufficiently big models, i.e. between tlinemdPhi-medium, `Gemma-9B` and between ttinlinemdLlaMa-8B. This improvement lies in the reasoning abilities observed in models with a greater number of parameters[43]. On the other hand, also `Phi-mini` benefices from this "reasoned" style of answer, while `Gemma-2B` is penalized with respect to the baseline. A possible explanation of this discrepancy stands in the pre-training procedure of `Phi-mini`: an entire step of it is dedicated to teaching the model how to logically reason and to attain specific skills (2.3.2). Consequently, `Phi-mini` small size does not influence the quality of the generated answer. This is obviously not the case of `Gemma-2B`, that possesses a diminished capability with respect to the 9B version and is not pre-trained for reasoning purposes.

The `ctx` variant differs from the previous one in two main points:

1. the *antithesis* is induced to quote all the relevant context before choosing the correct option, thus allows the model to split the content extraction and the effective decision process (this is an implicit

[43]This is also confirmed by the Chain-of-Thought paper by Wei et al. [68], where reasoning is described as *an emergent ability of more capable* (i.e. greater size) *models*.

process, that the model has to infer from the one-shot example that is provided);

2. the *synthesis* stage shows two examples of dealing with the *thesis*' and the *antithesis*' opinions on which is the correct answer (in the first one, the two agree; in the second one, they don't), thus instruct explicitly the *synthesis* on how to deal with these two scenarios.

This variation appears to work well (i.e. better than the baseline) on small models such as `Phi-mini` and `Gemma-2B`; this is probably due to the fact that these kinds of LLMs benefit from the division between the context selection and the effective decision. Additionally, instructing their *synthesis* phase on how to deal with agreement and disagreements could also have a positive effect.
Greater models (`Phi-medium`, of 14B parameters, and `Gemma-9B`) still behave better with a more "logically structured" approach as proposed by the `cot` variant: while `ctx` is still a quite valid option (for `Gemma-9B` it worsens the baseline performances, but by a reduced margin of only 0.9%) it seems to *burden* these pipelines in an unnecessary manner. `LlaMa-8B` shows comparable performances of `cot` and `ctx` and is in fact a middle ground between greater and smaller models in terms of number of parameters.
Despite some small differences widely discussed above, these first experiments confirm that our method allows to observe consistent improvements with different pipeline settings and across different models employed, not being dramatically biased by specific prompts.

Switching now to the `bridge` partition, we choose not to run the `is` variant because of its nearly perfect correspondence to the baseline.

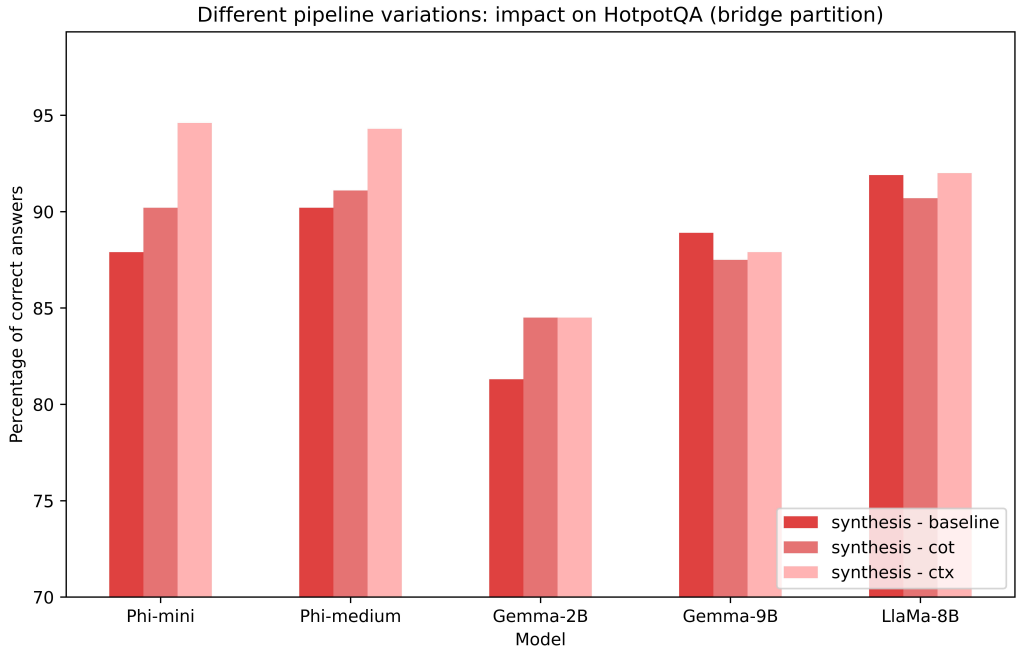


Figure 41: Different percentages of correct answers obtained using a particular pipeline setting **on HotpotQA - `bridge`**. The plot y-axis is reduced (starts from 70%) to highlight the small differences between the different one-shot examples.

On the other hand, we considered again both the `cot` and `ctx` variants. In this partition, the `cot` option seems slightly inappropriate: this is probably due to the different task that is required to perform. The baseline version of the pipeline seems to perform better than the `cot` variant in some cases.
While `comparison` required to compare two sources containing information about two different items and to reason on this content, `bridge` requires to merge correctly multiple sources, but once they are properly merged they are of easier understanding.
The reasoning abilities in this form of tasks is probably less important rather than focusing on the proper information inside the context and properly merge them. This second step is what the `ctx` variant does, and in fact it can be observed that in this subset this solution is the best one for all the tested models (for `Gemma-2B` the two variants achieve equal performances).

Finally, we want to test whether a different pipeline setting could help in improving the performances on the challenging WikiHop tasks.

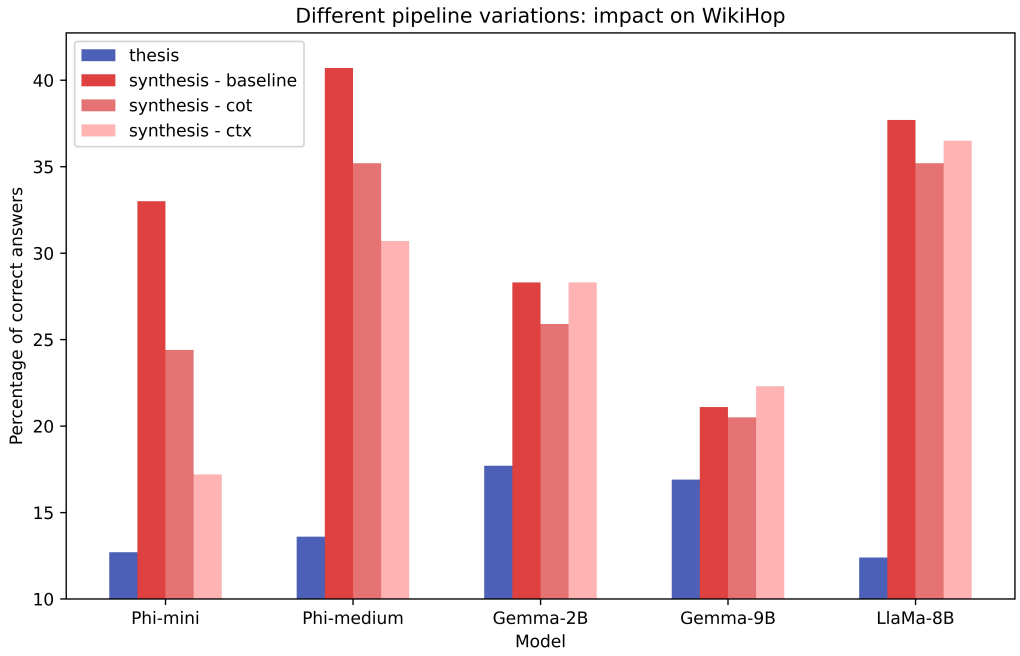


Figure 42: Percentages of correct answers obtained using a particular pipeline setting **on WikiHop**. The *thesis*' correct answers are represented by the blue bars, while the red ones represent the different pipeline variations considered.

For WikiHop, we can comment each family of models separately. The `Phi-3` models show the baseline as the best pipeline option, followed by the `cot` and finally by the `ctx` variations. `Gemma-2` models show similar performances of the baseline pipeline and the `ctx` variation, while `cot` appears worse. `LlaMa-8B` is similar to the previous family, although reaching higher performances for all the pipeline settings.
No pipeline variation is able to consistently outperform the baseline version of it on *multi-hop*, multiple choice tasks like those of WikiHop. The reason could be find in the fact that both `cot` and `ctx` ask the model to consider the context and reason on it, such as depicted in (4.6.2). On the opposite, the baseline version of the pipeline just asks the model to decree the correct answer, thus it can skip unnecessary context selection and go directly to the most proper answer.
In the following example, it is clear that while the baseline option can directly answer, the two other options are burdened by the prompt requirements (via the one-shot example). Since WikiHop is really complex and often presents a variety of options, many of which are semantically related (e.g. `United Kingdom` and `british`), the additional effort that the models have to face is to answer to the prompt's exact instructions.
In the following example, the question clearly ask the `parent taxon` of `Australosuchus`, thus it is not properly true (as `cot` states) that `'animal'` and `'crocodile'` are correct options. The model with `cot` *antithesis* is able to identify the fact that `'crocodilia'` is the correct option, but gets lost in its reasoning process and in the final stage is not able to clearly state that.
On the opposite, `ctx` directly forgets to include `'crocodilia'` as one of the options, probably confusing and merging it with `'crocodile'`.

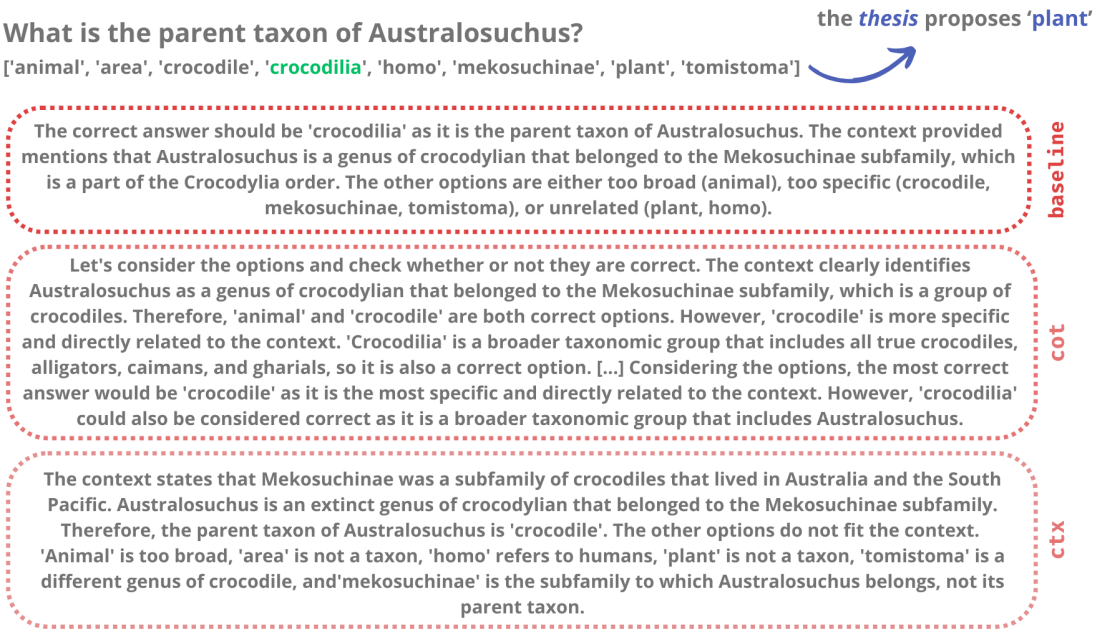


Figure 43: Baseline, `cot` and `ctx` *antitheses* obtained with `Phi-mini`.

Again, `Gemma-2` family exhibits poor abilities in absorbing the few-shot prompting implicit patterns. This allows its models to choose an original and *unscheduled* way of dealing with the problem, i.e. splitting the reasoning process in multiple steps in order to explain why a certain option is the correct one, while briefly explaining why the others are not. This approach is able to reach the correct option in some cases, while it leads the model to strongly support a wrong claim due to an approximate context usage (Figure 39).

`LlaMa-8B` allows the `ctx` variant to properly analyze each option, beating the `cot` one which tends to get struck on motivating why its option is better than the other, rather than consider each alternative independently and finally reach a conclusion.

Summing up, in tasks requiring a careful and concordant answer to a precise question in which the model has to deal with multiple options and long, *multi-hop* contexts, we suggest to let the pipeline do its work without forcing it to explain in a Chain-of-Thought manner why it does so. Additionally, we find that for these type of tasks, indulging in evaluating each relevant part of the context could be harmful[44].

[44]We can imagine the metaphor of an overthinking model: if it already knows the correct answer, it could *get lost* in trying to motivate everything besides that option.

5.5 *Overthinking* can be harmful (even for LLMs)

In the previous sections (4.2, 4.7) we question the effectiveness of the *synthesis* step, that could appear redundant once the *antithesis* gives its feedback.
To assess whether we need the *synthesis*, we extracted the options suggested by the *antithesis* step in order to observe what changes between these two steps. We compare the answers given in these two checkpoints.
In the following table we show the mean improvement across all datasets caused by the *antithesis* step with respect to the *thesis* (first rows) and the residual variation obtained by adding the *synthesis* step (second rows):

Setting	Step	`Phi-mini`	`Phi-medium`	`Gemma-2B`	`Gemma-9B`	`LlaMa-8B`
baseline	**T-A**	26.67	32.57	23.17	18.83	35.40
	A-S	1.13	1.03	2.50	0.57	0.57
cot	**T-A**	27.63	31.30	19.03	17.90	32.97
	A-S	0.23	1.63	0.43	0.43	2.63
ctx	**T-A**	27.97	32.17	26.27	21.43	38.43
	A-S	0.30	0.03	**-2.30**	**-2.57**	**-2.13**

Table 2: Mean accuracy gaps (across HotpotQA partitions and WikiHop) observed between the *thesis* and the *antithesis* steps (**T-A**) and between the *antithesis* and the *synthesis* ones (**A-S**). In bold, we highlighted the negative ones.

The results above are averaged across all datasets; the dataset-specific ones can be found in Appendix F. Despite each dataset's characteristics that we have previously described (5.3) and the model's tendency to behave differently to the same task (5.2), from results in Appendix F it appears quite clear that the pipeline variations condition on the *synthesis*' ability to improve the *antithesis*' one.

Besides the -1.8% of the `Gemma-2B` case on the `bridge` partition of HotpotQA, the *synthesis* stage turns out to be always beneficial if the baseline version of the pipeline is used.
Some performances drop between the *antithesis* and the *synthesis* stages are observed when used the `cot` variant, and this is reasonable due to the fact that the Chain-of-Thought approach to the answer already explores whether each option is the correct one or not. Adding a further step to this already exhaustive computation has the effect of bias towards a wrong answer some correct reasoning chains (even by small margins). This is due to the limited number of *hops* (2) that characterizes HotpotQA: `cot` *antithesis* is already too powerful for easy problems, and the *synthesis* could only "take the model off the road".
The prevalent loss of accuracy between the *antithesis* and the *synthesis* happens for HotpotQA partitions and the usage of the `ctx` variation. The reasons could be similar than the ones already proposed for the `cot` variant. Additionally, we highlight the fact that the `ctx` variant is the only one with two examples in the prompt; this suggests that a stronger guidance on how to deal with disagreement between *thesis* and *antithesis* is not beneficial. It could be interesting to modify this choice and study the `ctx` outcome with only a one-shot example, as in `cot`. This gap appears linked with the `Gemma-2` and `LlaMa-3.1` models employed.

We conclude that the pipeline works, but tends to be negatively affected if asked to *think too much* about a problem that is too simple. Thus, in front of relatively easy problems (small and already selected contexts, 1-*hop* passages, trivial questions given the context) we recommend to stop the `cot` or `ctx` pipeline variations at the *antithesis* stage and retrieve the proposed solution. Since identifying easy problems for the LLM can be a challenging task per se, we leave the use of an automatic detection component for such settings to future work.

Although by a small margin, the *synthesis* improves the prediction accuracy in more challenging scenarios or when we use the baseline version of the pipeline (also suggested for user-friendly coding, because the one-shot example is simpler to create).

5.6 Chain-of-Thought prompting comparison

We previously described our method as a *guided and disentangled Chain-of-Thought* (4.2), because we ask multiple times the same model to reason on which is the most correct answer to a question, providing to each step the opinion of the previous ones. Thus, we want to compare this method with its most natural competitor: the Chain-of-Thought prompting.
From now on, when we refer to `cot` we mean the pipeline variation, while CoT stands for the one-shot prompt as already described in (4.9). In that section we highlight the fact that the one-shot example provided in `cot` and in CoT is nearly the same[45].
The question now is whether our proposed method is able to outperform the accuracy obtained through Chain-of-Thought prompting.

In the following plots we compare the thesis, the CoT and the best *synthesis* accuracy values obtained for different models[46] on HotpotQA. By "the best *synthesis* accuracy values" we mean that we consider the pipeline configuration scoring higher accuracy values for that model and for that dataset. They could be different with respect to the considered model, e.g. the baseline pipeline for `Phi-mini` and the `ctx` variant for `LlaMa-8B`. We allow this to be mixed since not want to test whether a certain pipeline setting outperforms CoT; we just aim at checking whether the pipeline is able to beat CoT in any configuration.

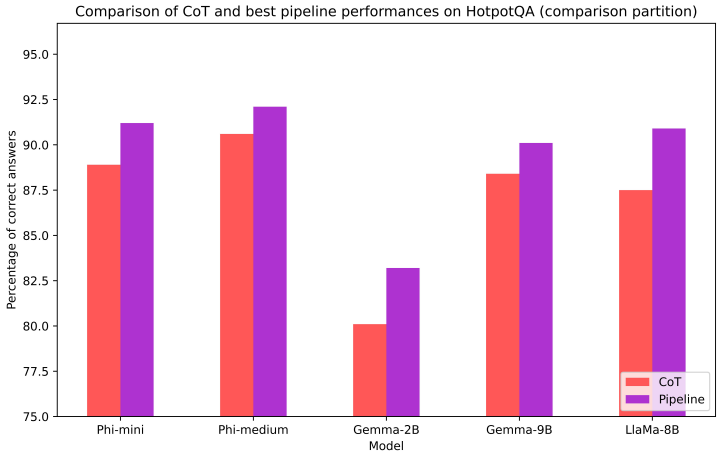


Figure 44: **HotpotQA `comparison` partition**: in red, the Chain-of-Thought answer accuracy values; in purple, the best *synthesis* ones.

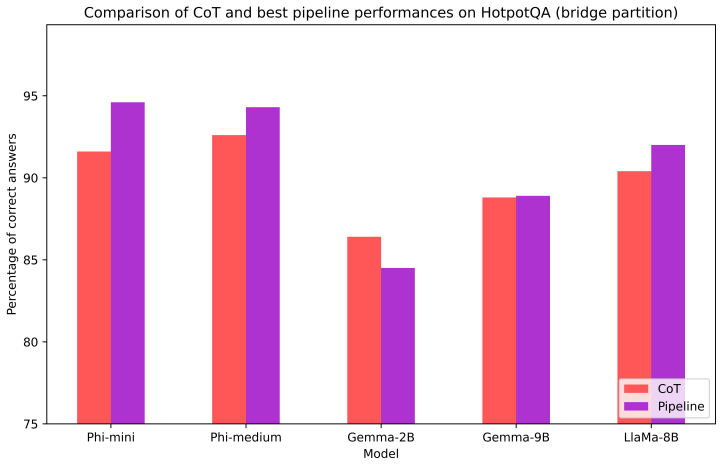


Figure 45: **HotpotQA `bridge` partition**: in red, the Chain-of-Thought answer accuracy values; in purple, the best *synthesis* ones.

[45] Except from the fact that the first one also contains the *thesis*' chosen option, while the second does not.
[46] Despite what we observed about the *antitheses* higher accuracy values than *syntheses* ones for simple tasks, for the sake of consistency we will use the second ones for evaluating the tasks from now on.

On one hand, the differences between the two partitions of HotpotQA are really small and almost always[47] are in favour of the proposed pipeline. On the other hand, we have to consider that these improvements are obtained with two extra steps, while Chain-of-Thought is able to reach similar values in a single step.

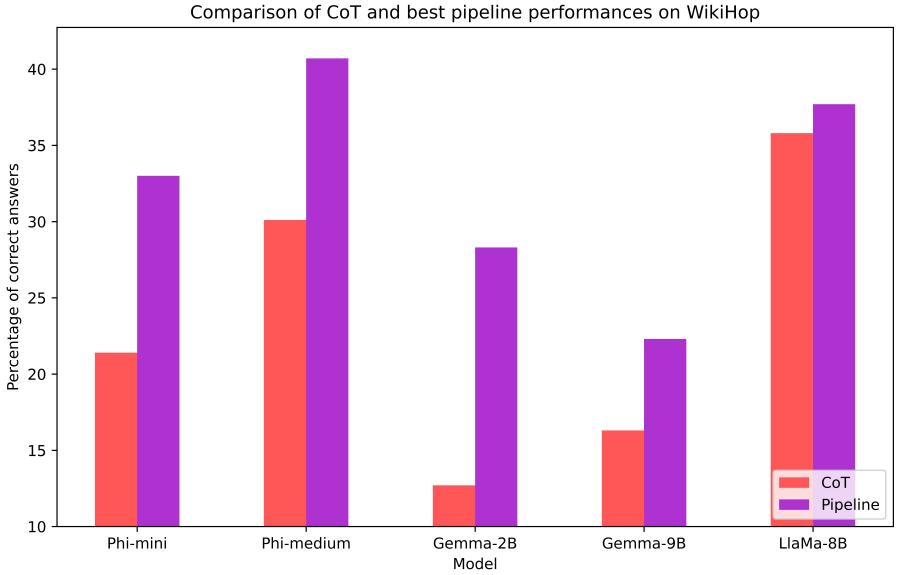


Figure 46: **WikiHop**: in red, the Chain-of-Thought answer accuracy values; in purple, the best *synthesis* ones.

On WikiHop the gap between the two solutions is much higher for some models: a 11.6% on `Phi-mini`, a 10.6% on `Phi-medium` and 15.6% on `Gemma-2B`. On the opposite, `Gemma-9B` baased pipeline outperforms CoT only by 3.8% and `LlaMa-8B` only by 1.9% (for HotpotQA this margin was greater).
The reasons of these gaps stand in the fact that both `Phi-3` and `Gemma-2` models receive a significant benefit from the pipeline, in particular when they face a challenging task like those of WikiHop. The tendency of the `Gemma-2` models of being more conversational rather than context-grounded (39) can be easily spotted from the low performances that these models exhibit when CoT-prompted. By asking them to consider again the *thesis*' proposed option, we are able to obtain large improvements.
`LlaMa-8B` shows instead smaller accuracy differences between the two approaches. This is also due to a performance drop between `LlaMa`'s *antithesis* and *synthesis* stages: if we early exit the pipeline, we would observe greater margins.
Additionally, in the previous section (5.5) we observed that `Gemma-2` and `LlaMa-3.1` family of models tend to favour a more *unstructed* prompting structure; when guided in the *synthesis* stage by providing them two examples, they perform worse than the one-shot scenario. Thus maybe the pipeline (at least for these family of models) is disadvantaged by the burden of having an extra instruction to follow.

Summing up, the pipeline is able to reach higher performances than CoT in almost all the scenarios studied. Certainly, CoT is a more computationally-lightweight than the proposed pipeline, so it is not extremely surprising that it works worse.
However, this is still an interesting finding that highlights how **we have not yet reached the boundary of the gains that can be derived from the reasoning skills of LLMs**.
An interesting frontier of this work consists in studying whether distilling the correctly-executed dialectic dialogues (*thesis - antithesis - synthesis*) in a new, smaller model still outperforms CoT. If positive results are observed, then we could consider to exploit this pipeline as a pre-training/fine-tuning component, rather than an inference strategy. Training models on *thinking tokens* has already been proved as effective by works as the one of Zelikman et al. [81].

[47] Only for `Gemma-2B` for the `bridge` partition CoT is slightly better; but the 84.5% reported for the pipeline is the `ctx`'s synthesis, the corresponding *antithesis* would beat CoT by a 2.8%.

5.7 Context filtering and summarization

Lastly, we want to consider *how much* the relevant passage is provided influences the generated output. The objects of our experiments are consequently the contexts of HotpotQA and WikiHop datasets, that would be manipulated and given as inputs to the pipeline. Their outputs are compared with those produced by using the original context in order to check whether there are some improvements.
WikiHop, that has already been summarized in order to reduce the context length (and stay inside the 4K limit), is considered in these experiments only as a subsection of it. We discard all the original contexts exceeding the threshold[48] and we summarize (4.3.1) and filter (4.3.2) the passages in this subset only.

Due to what we observe on `Gemma-2` models (39), we run the following experiments on the `Phi-3` and `LlaMa-3.1` model families only. We retain that this would be sufficient to assess whether there is an effective improvement in the pipeline's prediction accuracy.

Dataset partition	Pipeline	Passage	`Phi-mini`	`Phi-medium`	`LlaMa-8B`
`comparison`	thesis	original	53.4	50.0	48.3
		filtered	**54.9**	46.4	48.2
	baseline	original	80.7	89.5	87.2
		filtered	67.0	83.0	78.5
	`cot`	original	87.2	92.1	90.6
		filtered	58.2	74.2	79.7
	`ctx`	original	91.2	91.2	90.9
		filtered	57.0	70.3	75.5
`bridge`	thesis	original	52.1	56.0	49.8
		filtered	9.7	10.1	7.4
	baseline	original	87.9	90.2	91.9
		filtered	**96.1**	**94.7**	**93.8**
	`cot`	original	90.2	91.1	90.7
		filtered	**91.0**	**94.5**	**93.6**
	`ctx`	original	91.9	90.7	92.0
		filtered	90.3	**91.0**	**92.6**

Table 3: Context filtering results using (4.3.2) on **HotpotQA**. In bold we highlighted the experiments that outperform the non-filtered context ones.

This table contains interesting results. Despite being partitions of the same dataset, `comparison` behaves very differently from `bridge`.

While the filtered version of `bridge` scores terrible accuracy values in the thesis step, the other partition slightly improves the percentage of correctly guessed options. We think that this happens due to the two tasks' different nature.

Instead, `comparison` asks to compare two independent sources sharing some common content that has to be extracted, thus filtering the original context could help in focusing on the relevant parts of both these sub-passages already in the *thesis* stage. On the opposite, when the output of the filtering process is used in the pipeline, the performances worsen with respect to the original one. We can also notice that greater models (i.e. `Phi-medium` and `LlaMa-8B`) lose less in accuracy terms than smaller ones (i.e. `Phi-mini`). This is caused by an aggressive selection that the filtering stage performs in some cases. While helpful in neglecting irrelevant information, sometimes it also lefts out some important details useful to correctly answer the question, as showed in the example in Figure 47.

[48] Practically, we use 3.5K as threshold in place of the true 4K one in order to allow the instructions to fit in.

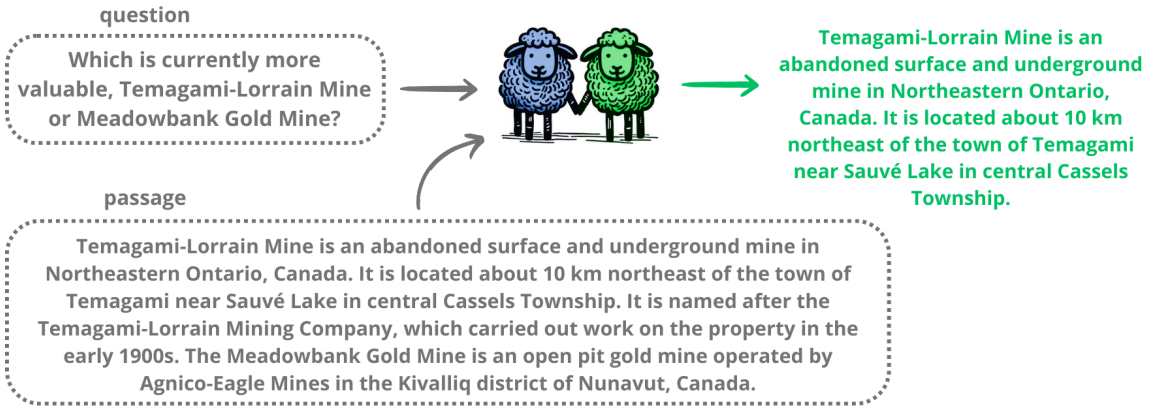


Figure 47: `Phi-mini` filtering process (4.3.2) on a problem of HotpotQA - `comparison`.

In this case, our proposed filtering process discards completely the sentence that talks about `Meadowbank Gold Mine`, causing the *antithesis* step[49] to correctly question that:

```
The context provided does not contain any information about the value of Temagami-Lorrain
Mine or Meadowbank Gold Mine. Therefore, it is impossible to determine which is
currently more valuable based on the given context.
```

A similar problem also affects the other partition of the dataset, but this time in the opposite direction. `bridge` questions require the model to merge multiple sources in order to understand how information present in different passages is related. It is now easy to imagine which is our filtering approach's enormous fatal flaw: if the linking part is left out, then the passage does not make sense anymore. Consider for example:

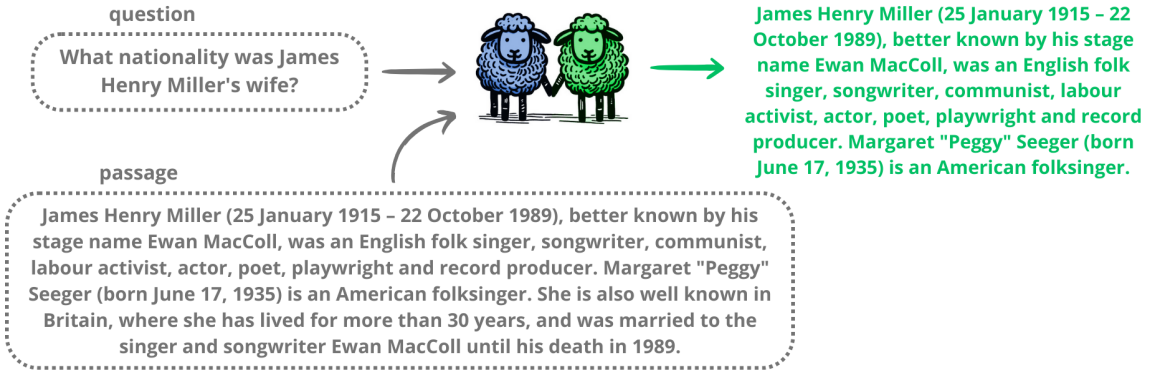


Figure 48: `Phi-mini` filtering process (4.3.2) on a problem of HotpotQA - `bridge`.

Understandably, the *thesis* does not recognize any James Henry Miller's wife from the given context. This is the reason why the accuracy scores of the first tentative answer are terribly low in this partition. The *antithesis* step is able to recognize the fact that the additional information given in the context should be exploited in some way, and makes a guess on why that information is provided:

```
The correct answer should be 'American' because the context states that Margaret "Peggy"
Seeger, who was James Henry Miller's wife, is an American folksinger.
```

No part of the given context says so, but the model knows that it has to search for a `wife` in the passage, thus it is reasonable that it is `Margaret "Peggy" Seeger`. And she is American.

Instead of worsening the output quality, getting rid of additional sentences not relevant for the context seems to give additional robustness to the pipeline's performances. The smaller gaps are again observed for `ctx`, probably for the same reasons we discussed in (5.5).

[49] Output obtained with `Phi-mini`.

In section (4.3.2) we specified that in order to compare WikiHop's original passages and eventually filtered ones, we would have to restrict the dataset to questions relative to passages with less than 4K tokens[50]. We also already mentioned that this subset is made of 92 questions only. We exploit the fact that we had run some tests (5.3) on the summarized versions of WikiHop (4.3.1) to extract the passages corresponding to those subset of questions.
We start again from the original context and filter it with the approach described in section (4.3.2) and also used on HotpotQA. In this way, we have at disposal three different versions of the same passage. We feed these alternatives to the pipeline and compare their results.
This time we are going just to consider the baseline pipeline setting due to what we have observed in (5.4). Some experiments carried out to observe other pipeline settings' behaviour on WikiHop confirmed that the baseline one is the best option.

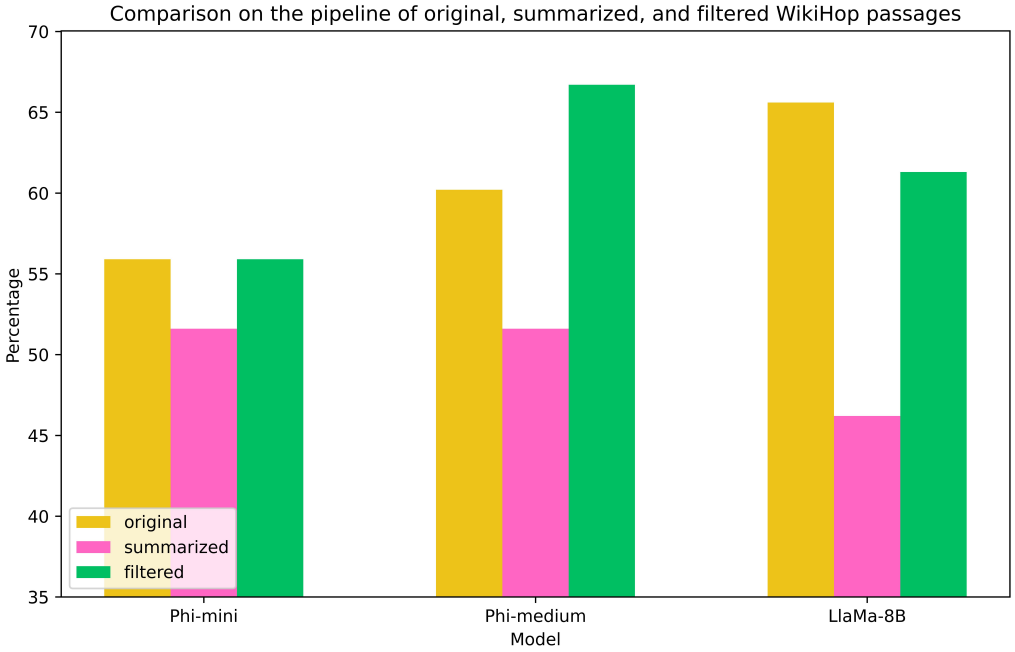


Figure 49: Percentage of answers guessed correctly using the baseline version of the pipeline on WikiHop. The different context versions are colored: in yellow, the original context; in orange, the summarized version; in green, the PECoRe filtered version (4.3.2).

The interesting fact that we can assess is that the three models behave quite differently except for the terrible summarization performances, shared by all of them.
For the `Phi-3` family of models, the original context and the filtered one perform comparably or the second one performs better; to us, this is due to the implicit abilities of LLMs to link the question and the *decontextualized* information in the context, as done in (5.7).
The accuracy gap between the original context and the filtered one in `LlaMa-8B` is likely due to its reduced capability to make implicit connections, such as those mentioned above.
Consider for example the question

```
In which language were the original works that discuss 'fires were started' written?
```

associated with the following filtered context:

```
The Blitz, from the German word "Blitzkrieg" meaning 'lightning war', was the name used
by the British press to describe the heavy air raids carried out over Britain in 1940 and
1941, during the Second World War. Frank Humphrey Sinkler Jennings (19 August 1907
24 September 1950) was an English documentary filmmaker and one of the founders of the
Mass Observation organisation. Jennings was described by film critic and director
Lindsay Anderson in 1954 as: "the only real poet that British cinema has yet produced.
The film uses actual firemen (including Cyril Demarne) rather than professional actors.
```

[50]Practically speaking the threshold is set to 3.5K.

And here we report the Phi-medium and LlaMa-8B answers:

- **Phi-medium:**

  ```
  The correct answer should be 'english' due to the context mentioning that Frank
  Humphrey Sinkler Jennings was an English documentary filmmaker and the film uses
  actual firemen rather than professional actors. The context does not provide any
  information about the original works discussing 'fires were started' being
  written in French, German, or Latin.
  ```

- **LlaMa-8B:**

  ```
  The correct answer should be 'latin' due to the fact that the context does not
  provide any information about the language in which the original works that discuss
  'fires were started' were written. However, the context does mention the Blitz and
  Frank Jennings, but it does not provide any information about the language of the
  original works. However, considering the context, it is more likely that the correct
  answer is not among the options provided. The context is about the Blitz
  and a documentary filmmaker, but it does not provide any information about
  the language of the original works that discuss 'fires were started'. But if I had to
  choose from the options provided, I would say that the correct answer is 'latin' because
  it is a more general and ancient language that could be related to the topic of 'fires
  were started', but it is not a definitive answer.
  ```

Despite the fact that the context is particularly *cryptic* even to humans, we can clearly see that Phi-medium is able to link fires to firemen and consequently to Frank Jennings, who speaks english. On the opposite, LlaMa-8B does not recognize this link and also assumes the wrong option, latin, which is evidently not the correct one also because of the fact that the 'fires were started' is expressed in english words.

We find that summarizing the context is not the best possible solution to deal with long contexts with non-relevant information included in them. When we prefer not to provide the original context, we can filter the sources using a method such as the one we proposed in section (4.3.2). This is also proved to improve overall performances when paired with models possessing good reasoning abilities such as the Phi-3 ones.
However, we suggest to use this form of pre-processing only when necessary. In cases such as HotpotQA, in which the context is already selected to be relevant, neglecting some parts of it could lead to a drop in the generation accuracy (such as in comparison). Additionally, we suggest also to prefer the baseline pipeline setting, since both the cot and ctx variants are built to deal with not-selected context (and thus they tend to focus on the relevant parts before answering, but this time the first operation has been already done).

6 Conclusions and future directions

The results exposed in the final section highlight the effectiveness of adopting a dialectic approach to face challenging multiple-choice question answering tasks.

An established way of dealing with this task is the Chain-of-Thought approach (4.9), which proposes to prompt a single model to produce a multi-step reasoning chain. This is supposed to let the model focus more on each sub-task present on the problem and to motivate its answer by producing natural language explanations.
Another way of facing the same problem is given by the self-refinement approaches (2.9.1, 2.9.2), that instead decompose the problem into multiple, iterative refinements made by the same large language model or by a specialized corrector.
Our method is a third option between these two, since it consists in dialectic setting involving three actors called *thesis*, *antithesis* and *synthesis* (4.2) that are asked to incrementally improve the previous steps' outputs. The latter steps are also asked to reason before answering, making them CoT-units and allowing us to refer to our method as a sort of *guided and disentangled Chain-of-Thought* (4.2).

However, we retained improper to define this pipeline as a Chain-of-Thought variation. Even though the pipeline exploits a single model, it involves three actors that check the answers' correctness, each one with a different role. In this sense we found reasonable not to classify this method neither as a Chain-of-Thought approach, nor as a self-refinement one.
Reflexion (2.9.3) was also tested by its creators in its CoT-augmented version on one of the two datasets that we considered (3.1.1) employing `GPT-4`; their results never reach the 80% of correct answers despite the high number of trials employed [59]. Our method was instead able to beat this threshold even with a 4B model (i.e. `Phi-mini`). Experiments performed in section (5.6) showed how the proposed pipeline was also able to consistently outperform the Chain-of-Thought alternative.

Our contributions are the following:

- We proposed a dialectic pipeline that is found effective for improving results in *multi-hop* question answering tasks (Figure 46). The margins of accuracy improvements are large and superior to the ones observed when Chain-of-Thought prompting is exploited to solve the problem.
- We assessed the robustness of this method with respect to different families of models and different datasets, proposing different *multi-hop* sub-problems and different number of *hops*. We found that greater models are generally more capable of dealing effectively with *multi-hop* sources. Additionally, a greater number of *hops* (e.g. in WikiHop) is correlated with worse model performances overall; however, this is also the case in which our proposed method scores the biggest improvements, both with respect to the baseline (5.3) and to the CoT-only approach (Figure 46).
- We tried different pipeline configurations (4.6.2) showing the model different ways of handling the context. We found that this helps in tasks with a limited number of *hops*, while it tends to confuse the model when the content to analyze is wide and complex; for this latter scenario, the baseline version is found to be the most effective one.
- The pipeline works best for models which are really careful in following instructions. Even though we considered all instruction-tuned models, we found that `Gemma-2` models tend to be creative when not requested, disobeying the few-shot example and performing the task with a different approach (39). Since this turns out to be poorer in performance terms, we can confirm that the improvement that the pipeline causes is not only due to additional computation (i.e. the fact that more steps are performed) but also to what is done in these steps.
- More capable models benefit less by the addition of the *synthesis* step (Table 2) and the `ctx` pipeline variation (4.6.2) causes a loss of accuracy when it has to face easy tasks (Appendix F). Since other settings do not cause this phenomenon, our opinion is that *overthinking* harms the process, thus an early stopping should be considered.
- When the context is long and noisy we can filter it to achieve greater performances (5.7). We exploited a modification of MIRAGE (4.3.2) and compared it with the results obtained with the summarized and the original context. What we found is that filtering enhances the predictions of

models with good deductive abilities (5.7), while summarization is the worst option among the tested ones.

This work is a first step in a wider set of experiments that we would like to test. Up to now, we have only considered open-source and relatively small (all under 20B) models; however, the prevalence of methods summarized in the literature review part (2) are tested on `GPT`-like decoder-only models. Although we proved the consistence of our pipeline across multiple families of open-source models (5.2), it would be interesting to run some tests to ensure that our method is widely applicable to proprietary LLMs too. Additionally, experimenting with the dialectic pipeline using various agents in the roles of thesis, antithesis, and synthesis to introduce diversity in skills presents an interesting avenue for future work.

We would like also to assess whether the pipeline works for other datasets. In our analysis we focused on *multi-hop* datasets, since they merge the two tasks of relevant content extraction and reasoning. We judged important to propose a method that is able to face this two tasks jointly instead of separately. However, studying the pipeline's effectiveness on these two tasks separately (e.g. datasets like GSM8K commonly allow to study only mathematical reasoning, while there is a wide literature of RAG-only datasets from which we can choose one to run relevant context extraction only) could still be a significant contribution to the research community.

Switching now the focus on possible pipeline inner improvements, we already observed that the number of *shots* provided to the prompt could be influential to the *synthesis*' improvements. Consequently, we could test both the zero-shot version of the pipeline and whether decreasing to one the number of `ctx` synthesis' examples could achieve better results.

In general, more effort could be made in trying to reduce the negative gap between the *antithesis* and the *synthesis* steps of the pipeline; a preliminary analysis of the observed "in-between change of mind" should lead pipeline modifications and hopefully improvements.

Regarding context filtering process, we ran all these tests by keeping fixed the percentage of relevant tokens (`p = 5`%) that the method has to select. It could be interesting to benchmark multiple proportions and observe whether there is a correlation between the datasets' *hops complexity* (i.e. the number of *hops* in the passages) and the degree of selection (i.e. the value of `p`) applied to its context.
The comparison between the pipeline's performances given the original, the summarized and the filtered context as inputs is performed only for passages up to 4K tokens. This is simply due to our choice of models; they have a limited context window that bounds their processing capacity. A separate study could be performed with the same models possessing a higher context length (e.g. `Phi-3` has also the `128k` models' versions) in order to study WikiHop more broadly, in order to draw more general conclusions on how the context conditions the pipeline's output.
A simple, final test that should be performed is the pipeline's behaviour when the model is not given. We ran some tests (not showed in this work) that confirm the pipeline's improvement with respect to the *thesis* step even when asked to rely only on the models' parametric knowledge. Expanding those experiments could allow us to leverage the pipeline both for factuality and reasoning tasks.

Lastly, it would be interesting to test whether a distillation of correctly-executed pipelines [60] could be a significant form of pre-training for LLMs.

References

[1] Marah Abdin et al. *Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone*. 2024. arXiv: 2404.14219 [cs.CL]. URL: https://arxiv.org/abs/2404.14219.

[2] G Apollinari et al. "High-Luminosity Large Hadron Collider (HL-LHC) : Preliminary Design Report". In: (2015). DOI: 10.5170/CERN-2015-005.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL]. URL: https://arxiv.org/abs/1409.0473.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL]. URL: https://arxiv.org/abs/1409.0473.

[5] Angels Balaguer et al. *RAG vs Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture*. 2024. arXiv: 2401.08406 [cs.CL]. URL: https://arxiv.org/abs/2401.08406.

[6] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: 2004.05150 [cs.CL]. URL: https://arxiv.org/abs/2004.05150.

[7] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[8] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[9] Manish Chablani. *Sequence to sequence model: Introduction and concepts*. 2017. URL: https://towardsdatascience.com/sequence-to-sequence-model-introduction-and-concepts-44d9b41cd42d.

[10] Jifan Chen, Eunsol Choi, and Greg Durrett. *Can NLI Models Verify QA Systems' Predictions?* 2021. arXiv: 2104.08731 [cs.CL]. URL: https://arxiv.org/abs/2104.08731.

[11] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. 2021. arXiv: 2110.14168 [cs.LG]. URL: https://arxiv.org/abs/2110.14168.

[12] Dorottya Demszky, Kelvin Guu, and Percy Liang. *Transforming Question Answering Datasets Into Natural Language Inference Datasets*. 2018. arXiv: 1809.02922 [cs.CL]. URL: https://arxiv.org/abs/1809.02922.

[13] Yiran Ding et al. *LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens*. 2024. arXiv: 2402.13753 [cs.CL]. URL: https://arxiv.org/abs/2402.13753.

[14] Jesse Dodge et al. *Fine-Tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping*. 2020. arXiv: 2002.06305 [cs.CL]. URL: https://arxiv.org/abs/2002.06305.

[15] Abhimanyu Dubey et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

[16] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. *Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning*. 2017. arXiv: 1702.03118 [cs.LG]. URL: https://arxiv.org/abs/1702.03118.

[17] J. R. Firth. "Studies in Linguistic Analysis". In: *In J. R. Firth, editor, Studies in Linguistic Analysis*. Oxford, UK: Basil Blackwell, 1957.

[18] GeeksForGeeks. *One Hot Encoding in Machine Learning*. 2024. URL: https://www.geeksforgeeks.org/ml-one-hot-encoding/.

[19] Google. *Gemma 2 release*. July 2024. URL: https://huggingface.co/collections/google/gemma-2-release-667d6600fd5220e7b967f315.

[20] google. *SentencePiece GitHub page*. URL: https://github.com/google/sentencepiece.

[21] Gopal Goyal. *Sliding Window Technique Grouped-Query Attention . Mistral 7B*. 2023. URL: https://medium.com/@gopalgoyal612002/mistral-llm-architectural-details-8dc0447fea62.

[22] Suriya Gunasekar et al. *Textbooks Are All You Need*. 2023. arXiv: 2306.11644 [cs.CL]. URL: https://arxiv.org/abs/2306.11644.

[23] Sanda Harabagiu and Andrew Hickl. "Methods for Using Textual Entailment in Open-Domain Question Answering". In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. Ed. by Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle. Sydney, Australia: Association for Computational Linguistics, July 2006, pp. 905–912. DOI: 10.3115/1220175.1220289. URL: https://aclanthology.org/P06-1114.

[24] Hangfeng He, Hongming Zhang, and Dan Roth. *Rethinking with Retrieval: Faithful Large Language Model Inference*. 2022. arXiv: 2301.00303 [cs.CL]. URL: https://arxiv.org/abs/2301.00303.

[25] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[26] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG]. URL: https://arxiv.org/abs/1606.08415.

[27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML]. URL: https://arxiv.org/abs/1503.02531.

[28] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. 2018. arXiv: 1801.06146 [cs.CL]. URL: https://arxiv.org/abs/1801.06146.

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

[30] HuggingFace. *Guidance documentation*. https://huggingface.co/docs/text-generation-inference/conceptual/guidance. Accessed: 2024-08-27. 2024.

[31] Inseq Team. *Inseq: Interpretability for Sequence-to-Sequence Models*. https://github.com/inseq-team/inseq. Accessed: 2024-09-10. 2024.

[32] Albert Q. Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL]. URL: https://arxiv.org/abs/2310.06825.

[33] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, 2009. ISBN: 9780131873216 0131873210. URL: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

[34] Daniel Kahneman. *Thinking, Fast and Slow*. New York: Farrar, Straus and Giroux, 2011.

[35] Subbarao Kambhampati. "Can large language models reason and plan?" In: *Annals of the New York Academy of Sciences* 1534.1 (Mar. 2024), 15–18. ISSN: 1749-6632. DOI: 10.1111/nyas.15125. URL: http://dx.doi.org/10.1111/nyas.15125.

[36] Subbarao Kambhampati. *On the Role of LLMs in Planning (ICML 2024 Tutorial)*. https://www.dropbox.com/scl/fi/gul511qacx58i5esrqi92/ICM2024-Tutorial.pdf?rlkey=mmv77ou4qyxi0a6lol3m6ug80&dl=0. Accessed: 2024-08-27. 2024.

[37] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

[38] Kiseung Kim and Jay-Yoon Lee. *RE-RAG: Improving Open-Domain QA Performance and Interpretability with Relevance Estimator in Retrieval-Augmented Generation*. 2024. arXiv: 2406.05794 [cs.CL]. URL: https://arxiv.org/abs/2406.05794.

[39] Takeshi Kojima et al. *Large Language Models are Zero-Shot Reasoners*. 2023. arXiv: 2205.11916 [cs.CL]. URL: https://arxiv.org/abs/2205.11916.

[40] Steve Jerome Lawrence. *Learning position with Positional Encoding*. 2023. URL: https://www.scaler.com/topics/nlp/positional-encoding/.

[41] Quentin Lhoest et al. "Datasets: A Community Library for Natural Language Processing". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Heike Adel and Shuming Shi. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184. DOI: 10.18653/v1/2021.emnlp-demo.21. URL: https://aclanthology.org/2021.emnlp-demo.21.

[42] Yanyang Li et al. *Making Long-Context Language Models Better Multi-Hop Reasoners*. 2024. arXiv: 2408.03246 [cs.CL]. URL: https://arxiv.org/abs/2408.03246.

[43] Nelson F. Liu et al. *Lost in the Middle: How Language Models Use Long Contexts*. 2023. arXiv: 2307.03172 [cs.CL]. URL: https://arxiv.org/abs/2307.03172.

[44] Aman Madaan et al. *Self-Refine: Iterative Refinement with Self-Feedback*. 2023. arXiv: 2303.17651 [cs.CL]. URL: https://arxiv.org/abs/2303.17651.

[45] Sourab Mangrulkar et al. *PEFT: State-of-the-art Parameter-Efficient Fine-Tuning methods*. https://github.com/huggingface/peft. 2022.

[46] Vaibhav Mavi, Anubhav Jangra, and Jatowt Adam. "Multi-hop Question Answering". In: *Foundations and Trends® in Information Retrieval* 17.5 (2024), pp. 457–586. ISSN: 1554-0669. DOI: 10.1561/1500000102. URL: http://dx.doi.org/10.1561/1500000102.

[47] Meta. *Llama 3.1 release*. August 2024. URL: https://huggingface.co/collections/meta-llama/llama-31-669fc079a0c406a149a5738f.

[48] Microsoft. *Phi 3 release*. June 2024. URL: https://huggingface.co/collections/microsoft/phi-3-6626e15e9585a200d2d761e3.

[49] Norman Mu et al. *Can LLMs Follow Simple Rules?* 2024. arXiv: 2311.04235 [cs.AI]. URL: https://arxiv.org/abs/2311.04235.

[50] OpenAI. *tiktoken: Fast BPE tokenization for OpenAI models*. https://github.com/openai/tiktoken. Accessed: 2024-09-07. 2023.

[51] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL]. URL: https://arxiv.org/abs/2203.02155.

[52] Jirui Qi et al. "Model Internals-based Answer Attribution for Trustworthy Retrieval-Augmented Generation". In: *ArXiv* abs/2406.13663 (June 2024). URL: https://arxiv.org/abs/2406.13663.

[53] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018. URL: https://api.semanticscholar.org/CorpusID:49313245.

[54] Jack W. Rae et al. *Scaling Language Models: Methods, Analysis Insights from Training Gopher*. 2022. arXiv: 2112.11446 [cs.CL]. URL: https://arxiv.org/abs/2112.11446.

[55] Rafael Rafailov et al. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. 2024. arXiv: 2305.18290 [cs.LG]. URL: https://arxiv.org/abs/2305.18290.

[56] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2023. arXiv: 1910.10683 [cs.LG]. URL: https://arxiv.org/abs/1910.10683.

[57] Alexandre Ramé et al. *WARP: On the Benefits of Weight Averaged Rewarded Policies*. 2024. arXiv: 2406.16768 [cs.LG]. URL: https://arxiv.org/abs/2406.16768.

[58] Gabriele Sarti et al. "Quantifying the Plausibility of Context Reliance in Neural Machine Translation". In: *The Twelfth International Conference on Learning Representations (ICLR 2024)*. Vienna, Austria: OpenReview, May 2024. URL: https://openreview.net/forum?id=XTHfNGI3zT.

[59] Noah Shinn et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. 2023. arXiv: 2303.11366 [cs.AI]. URL: https://arxiv.org/abs/2303.11366.

[60] Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. *Distilling Reasoning Capabilities into Smaller Language Models*. 2023. arXiv: 2212.00193 [cs.LG]. URL: https://arxiv.org/abs/2212.00193.

[61] Jianlin Su et al. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. 2023. arXiv: 2104.09864 [cs.CL]. URL: https://arxiv.org/abs/2104.09864.

[62] Gemma Team et al. *Gemma 2: Improving Open Language Models at a Practical Size*. 2024. arXiv: 2408.00118 [cs.CL]. URL: https://arxiv.org/abs/2408.00118.

[63] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288.

[64] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971.

[65] Sik-Ho Tsang. *Brief Review — GLU Variants Improve Transformer*. 2023. URL: https://sh-tsang.medium.com/brief-review-glu-variants-improve-transformer-9ee943115ab.

[66] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[67] Xuezhi Wang et al. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. arXiv: 2203.11171 [cs.CL]. URL: https://arxiv.org/abs/2203.11171.

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

[69] Jason Wei et al. *Finetuned Language Models Are Zero-Shot Learners*. 2022. arXiv: 2109.01652 [cs.CL]. URL: https://arxiv.org/abs/2109.01652.

[70] Jason Wei et al. *Finetuned Language Models Are Zero-Shot Learners*. 2022. arXiv: 2109.01652 [cs.CL]. URL: https://arxiv.org/abs/2109.01652.

[71] Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. *Constructing Datasets for Multi-hop Reading Comprehension Across Documents*. 2018. arXiv: 1710.06481 [cs.CL]. URL: https://arxiv.org/abs/1710.06481.

[72] Sean Welleck et al. *Generating Sequences by Learning to Self-Correct*. 2022. arXiv: 2211.00053 [cs.CL]. URL: https://arxiv.org/abs/2211.00053.

[73] Jason Weston and Sainbayar Sukhbaatar. *System 2 Attention (is something you might need too)*. 2023. arXiv: 2311.11829 [cs.CL]. URL: https://arxiv.org/abs/2311.11829.

[74] Wikipedia contributors. *Catastrophic interference — Wikipedia, The Free Encyclopedia*. [Online; accessed 17-August-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Catastrophic_interference&oldid=1237425828.

[75] Wikipedia contributors. *Dialectic — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Dialectic&oldid=1244221571. [Online; accessed 7-September-2024]. 2024.

[76] Brandon T. Willard and Rémi Louf. *Efficient Guided Generation for Large Language Models*. 2023. arXiv: 2307.09702 [cs.CL]. URL: https://arxiv.org/abs/2307.09702.

[77] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Qun Liu and David Schlangen. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6.

[78] Ruibin Xiong et al. *On Layer Normalization in the Transformer Architecture*. 2020. arXiv: 2002.04745 [cs.LG]. URL: https://arxiv.org/abs/2002.04745.

[79] Zhilin Yang et al. *HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering*. 2018. arXiv: 1809.09600 [cs.CL]. URL: https://arxiv.org/abs/1809.09600.

[80] Kayo Yin and Graham Neubig. *Interpreting Language Models with Contrastive Explanations*. 2022. arXiv: 2202.10419 [cs.CL]. URL: https://arxiv.org/abs/2202.10419.

[81] Eric Zelikman et al. *Quiet-STaR: Language Models Can Teach Themselves to Think Before Speaking*. 2024. arXiv: 2403.09629 [cs.CL]. URL: https://arxiv.org/abs/2403.09629.

Appendix A PECoRe invocation for context filtering

The context filtering process has its core in the following function:

```
def run(question, passage, p):
    tokens = tokenizer.tokenize(passage)
    invoke_pecore(passage, question, p)
    return select_passages(passage, question, p, tokens)
```

where `invoke_pecore(passage, question, p)` invokes PECoRe [58] and outputs the CCI scores of each context token:

```
def invoke_pecore(passage, question, p):
    pecore_args = AttributeContextArgs(
        model_name_or_path="microsoft/Phi-3-mini-4k-instruct",
        attribution_method="saliency",
        attributed_fn="contrast_prob_diff",
        context_sensitivity_metric="kl_divergence",
        context_sensitivity_std_threshold=1,
        context_sensitivity_topk = find_top_p(passage, p),
        attribution_std_threshold=None,
        attribution_topk=None,
        input_current_text=question,
        input_context_text=text_passage,
        contextless_input_current_text="""             <|system|>
            You are a helpful assistant that provide concise and accurate answers.<|end|>
            <|user|>
            {current}<|end|>
            <|assistant|>""",
        input_template="""<|system|>
            You are a helpful assistant that provide concise and accurate answers.<|end|>
            <|user|>
            {context}

            {current}<|end|>
            <|assistant|>""",
        contextless_output_current_text="""            {current}""",
        output_template="{current}",
        special_tokens_to_keep=['<|system|>', '<|end|>', '<|assistant|>', '<|user|>'],
        decoder_input_output_separator="",
        show_viz=False,
        save_path=None,
        viz_path=None,
        generation_kwargs={'max_new_tokens': 50},
    )

    out = attribute_context_with_model(pecore_args, inseq_model)
    return out
```

That are subsequently used to select only the sentences containing at least one influential token.

Appendix B Usage of `guidance` framework for multiple-choice questions

Example of usage of the `guidance` framework for constraining the model to output exactly one of the options given a multiple-choice question:

1. import the `guidance` model (and the corresponding tokenizer):

```
from guidance import models
guidance_model = models.Transformers(model, tokenizer)
```

2. define the desired prompt:

```
prompt = """
    You are a helpful AI assistant.
    You are given a question and the relevant context to answer it.
    Answer briefly to the question with just one of the given options.

    Question: Which year is Halley's Comet expected to return to the solar system?

    Options: [2110, 2045, 2086, 2061]

    Context: Astronomers have now linked the comet's appearances to observations
    dating back more than 2,000 years. Halley was last seen in Earth's skies in 1986
    and was met in space by an international fleet of spacecraft.
    It performs a regular 76-year journey around the Sun.

    Assistant:
"""
```

3. mask everything that is not inside the grammar (i.e. not one of the provided options) using the `select` function of the `guidance` framework:

```
from guidance import select
answer = guidance_model + prompt + select([2110, 2045, 2086, 2061])
```

Appendix C Thesis

Code used for the *thesis'* generation:

```
from guidance import models, select

guidance_model = models.Transformers(model, tokenizer, temperature=0.0)

def create_message_thesis(question, options, context):

    messages = [
        {"role": "system", "content": """
        You are an helpful AI assistant.
        You have to provide helpful answers to the user's questions based on the context:
        """ + context},
        {"role": "user", "content": user_content}
    ]

    user_content = "Answer to the following question: " + question +
                   " providing one of these options as answer: " + options +
                   "Assistant: "

    return messages

def thesisGeneration(question, options, context):
    prompt = create_message_thesis(question, options, context)
    answer = guidance_model + prompt + select(options)
    return answer
```

Appendix D Antithesis

We used the following function to generate the *antithesis*:

```
def antithesisGeneration(question, options, thesis, context):
    prompt = create_message_antithesis(question, thesis, options, context)
    output = pipeline(prompt, **generation_args)
    return output[0]['generated_text']
```

where `create_message_antithesis()` is the function responsible to describe how we want the task to be performed, both using instructions and a concrete example that is also augmented with the new task:

```
def create_message_antithesis(question, options, thesis, context, one_shot_example):

    messages = [
        {"role": "system", "content": """
        You are an helpful AI assistant. You are asked to determine the most correct answer for
        a given question, provided a set of possible options. You also have at disposal a first
        tentative answer that you are required to check with respect to the question and the
        relevant context. Your goal is to decree which is the most correct answer to the question
        between the available options.

        Here's an example of how to do it:
        """},
        {"role": "user", "content": one_shot_example
        },
        {"role": "system", "content": "Now do the same for the following question:"},
        {"role": "user", "content": user_content}
    ]
```

```
    user_content = "Question: " + question + "\n Options: " + options +
                   "\n Candidate answer: " + candidate + "\n Context: " + context +
                   "\n Assistant: \n"

    return messages
```

where `one_shot_example` is the concrete example of expected behaviour of the *antithesis'* step of the pipeline.

Appendix E Synthesis

Similarly as before, we report here the generation prompt:

```
def create_message_presynthesis(question, thesis, antithesis, options, context,
                                few_shot_example):
    messages = [
        {"role": "system", "content": """
        You are an helpful AI assistant. You are asked to determine the most correct answer
        for a given question, provided a set of possible options.
        You also have at disposal a first tentative answer and a suggestion on which is the
        correct answer. Your goal is to decree which is the most correct answer to the
        question between the available options according to the context.

        Here's an example/a few examples of how to do it:
        """},
        {"role": "user", "content": few_shot_example
        },
        {"role": "system", "content": "Now do the same for the following question:"},
        {"role": "user", "content": user_content}
    ]

    user_content = "Question: " + question + "\n Options: " + options +
                   "\n Candidate answer: " + thesis + "\n Suggestion: " + antithesis +
                   "\n Context: " + context + "\n Assistant: \n"

    return messages
```

Appendix F Relative improvements between different pipeline steps

Expanded table showing the relative improvements between different pipeline steps:

Dataset	Setting	Step	Phi-mini	Phi-medium	Gemma-2B	Gemma-9B	LlaMa-3.1-8B
WikiHop	baseline	T-A	+20.1	+25.1	+16.5	+5.1	+25.9
		A-S	+0.2	+2.0	+5.9	+0.9	+0.6
	cot	T-A	+7.8	+18.6	+1.0	+2.7	+17.4
		A-S	+3.9	+3.0	+2.4	+0.9	+5.4
	ctx	T-A	+4.2	+15.6	+16.6	+8.7	+28.0
		A-S	+0.3	+1.5	**-5.9**	**-3.3**	**-3.9**
bridge	baseline	T-A	+35.3	+33.7	+27.4	+22.2	+41.9
		A-S	+0.5	+0.5	**-1.8**	+0.2	+0.2
	cot	T-A	+40.4	+34.4	+30.5	+21.8	+40.0
		A-S	**-2.3**	+0.7	**-1.7**	**-0.8**	+0.9
	ctx	T-A	+41.9	+38.0	+33.5	+25.8	+44.4
		A-S	+0.6	+0.3	**-4.7**	**-4.4**	**-2.2**
comparison	baseline	T-A	+24.6	+38.9	+25.6	+29.2	+38.4
		A-S	+2.7	+0.6	+3.4	+0.6	+0.9
	cot	T-A	+34.7	+40.9	+25.6	+29.2	41.5
		A-S	**-0.9**	+1.2	+0.6	+1.2	+1.6
	ctx	T-A	+37.8	+42.9	+28.7	+29.8	+42.9
		A-S	+0.0	**-1.7**	+1.7	**-2.3**	**-0.3**

Ringraziamenti

Per prima cosa, un grande grazie va al prof.Bortolussi, che ha sostenuto con entusiasmo questa idea dal primo minuto e che l'ha vista diventare realtà in questa tesi, proiettando ancora questo lavoro di ricerca verso nuove prospettive.
Impossibile non ringraziare anche Gabriele, che con infinita pazienza mi ha guidata nei meandri di questa materia. Lavorare con lui è stata per me la conferma del percorso che ho scelto per i prossimi tre anni della mia vita: mi ha mostrato la curiosità, la sfida e la gioia della ricerca, e per questo gli sarò sempre grata. Se non mi sono sentita un *nano* sviluppando questa tesi al loro fianco è solo perché mi hanno *presa sulle loro spalle*, come un famoso detto dice.

Grazie alla mia famiglia, che mi è stata accanto per tutta la strada che mi ha portata fino a questo traguardo. Di fronte a tante persone che partono nel sentiero della vita con lo zaino colmo di insicurezze e dolori non propri, voi siete stati capaci di insegnarmi il linguaggio dell'affetto, *stretta in libera sorte*. Spero di essere in grado di donare l'amore a chi incontrerò nella mia vita come me l'avete trasmesso.

Grazie agli amici di sempre, Elia, Riccardo, Carla e Marco, che mi hanno conosciuta in fasi in cui ero molto diversa e hanno continuato a volermi bene e starmi accanto mentre cambiavamo, nonostante i diversi ritmi e luoghi delle nostre vite. Un grazie particolare ad Elia, che è parte fondante di ciò che ritengo "casa"; sono davvero grata di averti incontrato.

Grazie a Davide, che ha sempre creduto in me molto più di quanto ci credessi io, sapendo accogliere i miei momenti di difficoltà con cuore gentile e gioendo insieme a me di ogni piccola conquista, nonostante tutto.

Grazie anche a LAM, che è stata per me primavera dopo il periodo COVID e che mi ha dimostrato che le persone hanno il potenziale di cambiare le cose; che se ci impegniamo, 1+1 fa davvero 3. Grazie per avermi sopportata e per avermi regalato la prova che il limite di ciò che riteniamo possibile è facilmente superabile se ci si circonda delle persone giuste.

Grazie ad AI2S e alle persone che ne fanno parte: abbiamo appena iniziato a tracciare gli orizzonti di tanti piani e di tanti nuovi modi di stare insieme, e non vedo l'ora di scoprirli tutti.

Grazie al gruppo "C5" (diventato "5C") e ai *data scientist*: sarebbe stato tutto drammaticamente meno divertente e significativo se non ci fossimo conosciuti. Ci siamo incontrati in un momento in cui pensavo che la mia vita fosse già piena a sufficienza e mi avete dimostrato che l'affetto è sicuramente moltiplicativo, non additivo; siete delle bellissime persone che spero di non perdere.

Grazie poi in ordine sparso a tante cose: all'edificio L e all'aula studio in C5, ai 2.3 kg di pasta frolla, a via Scussa, alle passeggiate che terminano troppo tardi, alle serate davanti al pianoforte, a chiunque abbia inventato "briscola a 5", al cestino di ORFEO, ad Harry Potter, all'armocromia, alle LAM-gite, agli audio(libri) su WhatsApp, ad Italo Calvino.