ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

TO GENERATE OR TO RETRIEVE: ON THE EFFECTIVENESS OF ARTIFICIAL CONTEXTS FOR BIOMEDICAL QUESTION ANSWERING

Elaborato in
Programmazione di Applicazioni Data Intensive

Relatore
Prof. Gianluca Moro

Co-relatori
Dott. Giacomo Frisoni

Presentata da
Alex Presepi

Seconda Sessione di Laurea
Anno Accademico 2022 – 2023

KEYWORDS

Natural Language Processing
Large Language Models
Augmented Generation
Open-Domain Question Answering
Biomedical Domain

A Wandering Mind Is an Unhappy Mind
- Matthew A. Killingsworth and Daniel T. Gilbert

Sommario

I Large Language Model (LLM) al giorno d'oggi sono applicati in diversi ambiti, particolare attenzione è rivolta ai knowledge-intensive task come l'open-domain question answering (ODQA). La maggior parte delle soluzioni allo stato dell'arte sono basate su pipeline retrieve-then-read, che prima recuperano documenti rilevanti da fonti esterne e poi generano una risposta aumentando il contesto del language model. Questa metodologia si affida agli embedding e alla loro indicizzazione all'interno di vector database, comportando diverse limitazioni. Generate-then-read (GenRead), sostituisce il componente di retrieval con dei LLM generator. Questo approccio ha recentemente superato le soluzioni retrieve-then-read precedenti e la pura generazione senza contesto aumentato, in task come ODQA in dominio generale, fact checking, e dialogue system. Nel dominio biomedico, l'applicazione di tali contributi assume un'importanza cruciale dettata dal gergo settoriale, l'alto numero di entità, il rapido evolversi della conoscenza scientifica, la non tolleranza di allucinazioni, e il bisogno di evidenze a supporto delle inferenze prodotte. Questa tesi esplora il paradigma generate-then-read nel dominio biomedico utilizzando LLM open-source e con un numero di parametri contenuto. Per la generazione di documenti sono implementate soluzioni ensembling, utilizzando diversi LLM addestrati su dataset differenti. Si propone una pipeline in cui inizialmente si generano k interrogativi inerenti al contesto della domanda, seguiti dalla generazione delle risposte da parte di un LLM. Inoltre esaminiamo e confrontiamo diverse strategie per la fase di lettura del documento e risposta, utilizzando chatbot, Fusion-in-Decoder (FiD) e modelli encoder-only. Utilizzando MedMCQA come dateset di domande a scelta multipla, medllama2 come generatore diretto di documenti e LLaMA 2 come chatbot per formulare la risposta, si ottiene un'accuratezza di 39.1%, rispetto a 36.8% di medllama2 e 35.2% di LLaMA 2.

Abstract

Large Language Models (LLMs) nowadays are used to solve more tasks, focusing on knowledge-intensive tasks like open-domain question answering (ODQA). Most state-of-the-art solutions rely on retrieve-then-read pipelines that first retrieve relevant documents from external sources and then generate a response by augmenting the language model context. This methodology is based on embeddings and their indexing in vector databases but has several limitations. Generate-then-read (GenRead) replaces the retrieval component with LLM generators. This approach has recently exceeded previous retrieve-then-read solutions and pure generation without augmented context in tasks such as domain-general ODQA, fact-checking, and dialogue systems. In the biomedical field, these contributions become exceptionally significant due to the specialized terminology, the abundance of entities, the rapid advancement of scientific knowledge, the intolerance of hallucinations, and the necessity for evidence to verify the generated inferences. This thesis explores the generate-then-read paradigm in the biomedical domain using open-source LLM and with a limited number of parameters. Ensembling solutions are implemented for document generation, using different LLMs trained on different datasets. A pipeline is proposed in which k questions related to the context of the query are initially generated, followed by the generation of answers by an LLM. We also examine and compare different strategies for the document reading and response phase using chatbots, Fusion-in-Decoder (FiD), and encoder-only models. Using MedMCQA as the multiple-choice question dataset, medllama2 as the direct document generator, and LLaMA 2 as the chatbot to formulate the answer, an accuracy of 39.1% is achieved, compared to 36.8% for medllama2 and 35.2% for LLaMA 2.

Introduzione

Negli ultimi anni, i Large Language Model (LLM), branca del deep learning e dell'elaborazione del linguaggio naturale (NLP), sono diventati sempre più popolari grazie all'introduzione di chatbot come ChatGPT[1] e Bard,[2] facendo scalpore per la qualità con cui riescono a generare testo in linguaggio naturale. Oltre a produrre testo semanticamente e grammaticalmente corretto, riescono a ragionare, rispondere a domande poste e seguire in modo chiaro le istruzioni fornite. La conoscenza e le abilità risiedono nei loro parametri di addestramento, fase self-supervised in cui vengono sottoposti a enormi quantità di testo non etichettato e non strutturato.

I LLM odierni sono basati su un'architettura transformer [1], fondata sul meccanismo dell'attention. Alcuni studi sostengono che aumentando il numero di parametri le prestazioni migliorano notevolmente, introducendo diverse scaling law come, KM scaling law [2] e la più recente Chinchilla scaling law [3]. Quest'ultima si focalizza sull'aumentare di conseguenza anche il numero di dati di addestramento, processo delicato data l'importanza della qualità dei dati. I language model vengono definiti "large" per l'immensa quantità di parametri che possiedono, generalmente decine o centinaia di miliardi [4], come LLaMA [5] 70 miliardi, GPT-3 [6] e InstructGPT [7] con 175 miliardi e PaLM [8] 540 miliardi.

Ogni giorno, questi modelli vengono sottoposti a prove complesse per verificare la loro efficacia e utilità in una serie di applicazioni, tra cui traduzione, sintesi automatica, classificazione del testo e question answering. Nella comunità scientifica viene data attenzione ai knowledge-intensive task che intrinsecamente richiedono molte informazioni, generali o specifiche di un dominio, per essere portati a termine. In particolare l'open-domain question answering (ODQA) si pone come obbiettivo quello di rispondere a domande che non dispongono di un documento di accompagnamento da cui estrapolare direttamente la risposta.

La maggior parte delle soluzioni allo stato del arte sono basate su pipeline che prima recuperano documenti rilevanti da fonti esterne, e poi generano

[1] `https://openai.com/blog/chatgpt`
[2] `https://bard.google.com/`

una risposta aumentando il contesto del language model, soluzioni note come Retrieval Augmented Generation (RAG). Soluzioni di questo tipo hanno accesso ad informazioni esterne e sono classificate open-book. Il retriever, componente principale, esegue la fase di recupero documenti da fonti come Wikipedia, e si compone di due fasi: (i) fase di preprocessing dividere il testo in chunk ed eseguirne l'embedding ottenendo un vector database, embedding-chunk, (ii) eseguire l'embedding della query ed attraverso metriche di confronto tra vettori, come la similarità coseno, selezionare k chunk più simili alla query. Questo sistema presenta problemi e limitazioni; i chunk di lunghezza fissa (es., 100 parole) possono avere uno scarso potere informativo o essere affetti da rumore, soprattutto nelle pubblicazioni scientifiche dove l'informazione è spesso sparsa, incompleta e non contestuale. Inoltre, la fase di preprocessing dell'intero corpus limita i parametri del retriever e le dimensioni degli embedding, non sfruttando del tutto la conoscenza del mondo o le capacità di deduzione dei LLM, con rappresentazioni di domande e documenti spesso ottenute in modo indipendente, portando a interazioni superficiali tra di loro.

Generate Rather Than Retrieve [9] si discosta dalle soluzioni precedenti utilizzando una pipeline del tipo generate-then-read (GenRead) piuttosto che retrieve-then-read. Si sfruttano i LLM per generare diversi documenti inerenti alla domanda, mettendo in luce che essi possono essere buoni generatori di contesto. La generazione di testo trae maggiore vantaggio dalla conoscenza incorporata nei parametri poiché rappresenta un task più simile alla fase di pre-training dei causal language model. Ciò si differenzia dall'approccio di porre direttamente la domanda, che lascia molta conoscenza inesplorata. Inoltre, l'architettura transformer dei LLM riesce a comprendere relazioni di maggior complessità rispetto all'utilizzo di metriche di confronto tra gli embedding. Questa soluzione si può definire open-book in quanto ricorre a informazioni provenienti da un altro modello. Gli autori hanno utilizzato il modello pre-trained InstructGPT; modelli di queste dimensioni sono poco accessibili dato il costo delle API o le esigenze hardware. GenRead viene esaminato esclusivamente in task che richiedono una comprensione generale del mondo, tra cui dialogue system, fact checking e question answering nei dataset TriviaQA [10], webQ [11] e NQ [12].

Il dominio biomedico, in continuo sviluppo, vanta una particolare attenzione da parte dei ricercatori e i LLM presentano un immenso contributo, dati i numerosi dati presenti, un elevato numero di entità e la complessità del gergo settoriale. Esistono diversi dataset di question answering che comprendono esami di ammissione in campo medico, tra cui MedMCQA [13], composto da domande a scelta multipla. Questa tesi propone diverse soluzioni basate sulla pipeline generate-then-read per il task di ODQA nel dataset MedMCQA. Ci si limita ad usare LLM open-source e con un numero di parametri nel ordine delle

decine di miliardi, modelli pre-trained come LLaMA 2 [14] o anche fine-tuned in ambito biomedico come PMC-LLaMA [15]. Poichè non sempre modelli di questo tipo riescono a generare documenti corposi e di qualità, introduciamo una pipeline che prevede l'utilizzo di due LLM: il primo genera k interrogativi inerenti alla domanda e successivamente il secondo risponde a tali domande, contribuendo così a formare un documento più ampio. Infine é cruciale che durante la fase di lettura e risposta si interpreti bene il documento. Analizziamo reader basati su chatbot, Fusion-in-Decoder (FiD) [16] e modelli encoder-only.

Introduction

In recent years, the Large Language Models (LLMs), a branch of deep learning and natural language processing (NLP), have become increasingly popular thanks to the introduction of chatbots such as ChatGPT[3] and Bard,[4] gaining attention for their ability to generate high-quality natural language text. In addition to producing semantically and grammatically correct text, they are able to reason, answer posed questions, and clearly follow the instructions provided. Their knowledge and abilities reside in their training parameters, a self-supervised phase in which they are exposed to huge amounts of unlabeled and unstructured text.

Contemporary LLMs are based on a transformer architecture [1], based on the mechanism of attention. Some studies argue that significantly improving performance is achieved by increasing the number of parameters, introducing various scaling laws such as the KM scaling law [2] and the more recent Chinchilla scaling law [3]. The latter also focuses on increasing the number of training data accordingly, which is a delicate process given the importance of data quality. Language models are referred to as "large" due to the immense number of parameters they possess, typically tens or hundreds of billions [4], like LLaMA [5] with 70 billion, GPT-3 [6] and InstructGPT [7] with 175 billion and PaLM [8] with 540 billion.

Every day, these models undergo complex tests to verify their effectiveness and usefulness in a number of applications, including translation, summarization, text classification, and question-answering. In the scientific community, attention is given to knowledge-intensive tasks that inherently require a lot of general or domain-specific information to complete. In particular, open-domain question answering (ODQA) aims to answer questions that do not have a supporting document from which to directly extract the answer.

Most state-of-the-art solutions are based on pipelines that first retrieve relevant documents from external sources and then generate a response by augmenting the context of the language model, solutions known as Retrieval

[3] `https://openai.com/blog/chatgpt`
[4] `https://bard.google.com/`

Augmented Generation (RAG). Such solutions have access to external information and are categorized as open-book systems. The retriever, a key component, performs the document retrieval phase from sources like Wikipedia and consists of two steps: (i) a preprocessing phase that divides the text into chunks and obtains an embedding, resulting in a vector database, embedding-chunk, (ii) encodes the query and through vector comparison metrics, such as cosine similarity, select k chunk that is more similar to the query. This system presents problems and limitations; chunks of fixed length (e.g., 100 words) may contain limited information or be affected by noise, especially in scientific publications where information is often sparse, incomplete, and not contextual. Additionally, the preprocessing phase of the entire corpus limits retriever parameters and embedding size, not fully leveraging the knowledge of the world or the deduction capabilities of LLMs, with representations of questions and documents often obtained independently, leading to superficial interactions between them.

Generate Rather Than Retrieve [9] deviates from previous solutions by employing a generate-then-read (GenRead) pipeline instead of a retrieve-then-read approach. The LLMs are used to generate various documents related to the question, highlighting that they can be good context generators. Text generation benefits most from the knowledge embedded in the parameters as it represents a task more similar to the pre-training phase of the causal language model. This approach differs from directly asking the question, which leaves a lot of unexplored knowledge. Furthermore, LLM's transformer architecture can understand more complex relationships compared to the use of similarity metrics between embeddings. This solution can be described as open-book since it relies on information from another model. The authors used the pre-trained InstructGPT model; models of this size are less accessible due to API costs or hardware requirements. GenRead is examined exclusively in tasks that require a general understanding of the world, including dialogue system, fact-checking, and question answering in datasets like TriviaQA [10], webQ [11] and NQ [12].

The biomedical domain, in continuous development, receives particular attention from researchers, and LLMs contribute immensely due to the abundance of data, a high number of entities, and the complexity of the sector's jargon. There are several question-answering datasets that include medical admission exams, such as MedMCQA [13], which consists of multiple-choice questions. This thesis proposes various solutions based on the generate-then-read pipeline for the ODQA task in the MedMCQA dataset. We limit ourselves to using open-source LLMs with a parameter count in the order of tens of billions, including pre-trained models like LLaMA 2 [14] and fine-tuned models in the biomedical domain like PMC-LLaMA [15]. Since such models do not always generate exhaustive and high-quality documents, we introduce a pipeline that involves the use of two LLMs: the first generates k questions related to the que-

ry, and then the second answers these questions, contributing to the formation of a more extensive document. Finally, it is crucial that during the reading and answering phase, the document is interpreted correctly. We examine readers based on chatbots, Fusion-in-Decoder (FiD) [16], and encoder-only models.

Contents

List of Figures

Chapter 1

Theoretical Framework

This chapter overviews fundamental concepts within LLMs, explaining how computers interpret words and sentences, other than analyzing today's architecture and features.

1.1 Embeddings

Humans speak in words and sentences, but computers can only interpret and process numbers. The main aim of NLP, a subfield of Artificial Intelligence (AI), is to enable machines to comprehend and generate human language in a valuable way, focusing on understanding the context, not only single words. Tokenization, a preprocessing phase where words are broken down into single characters, subwords, or whole words called tokens, results in a sequence that is typically longer than the original input text, and each token is assigned a numerical ID. Tokens allow computers to effectively work with text data and allow for word embeddings and sentence embedding techniques.

Word embedding The mapping of words to numerical vectors is called word embedding, and these representations exist within a high-dimensional space that captures the features of the word, and each feature is one new axis. Similar words should be close, and different words should be far away. Coordinates represent the properties of words such as gender, size, and age, while others represent properties that a human may not understand.

Sentence embedding However, word embedding cannot capture complex semantic relationships in natural language sentences. Furthermore, the process of embedding each word can be computationally expensive. To address these limitations, we require sentence embedding, which coherently associates every

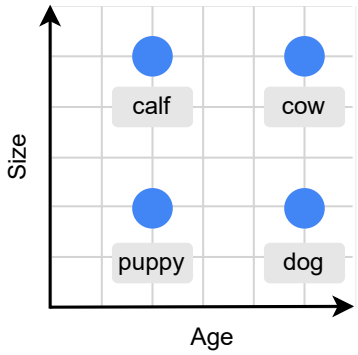


Figure 1.1: Example of word embeddings for 'calf,' 'cow,' 'puppy,' and 'dog' plotted in a two-dimensional space, with one axis representing size feature and the other representing age.

sentence with a vector of numbers. Similarly to word embedding, sentences with a similar meaning are close, and sentences with a different meaning are far away. This is possible through neural network techniques such as recurrent neural networks (RNNs) or transformers.

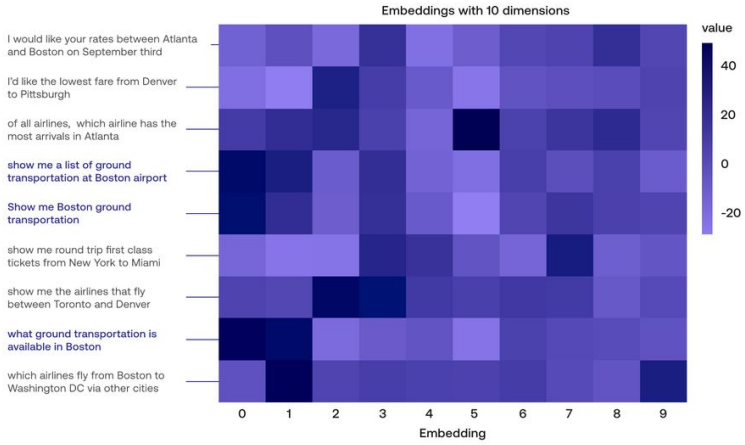


Figure 1.2: Some sentence embedding obtained by Cohere embedding model and compressed in 10-dimensional space using a dimensionality reduction algorithm. Vectors are represented using a color scale, and sentences with similar colors also share similar meanings.
From `https://docs.cohere.com/docs/text-embeddings`

Now, it is essential to determine if two words or sentences are similar. We can calculate similarity metrics with vector embedding, such as cosine similarity 1.1, which measures the cosine of the angle between two vectors. The value ranges from -1 to 1, where -1 indicates perfect dissimilarity, 0 indicates no similarity, and 1 indicates equal vectors. It is widely used for comparing word

or document embeddings.

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|} \tag{1.1}$$

Where $A \cdot B$ represents the dot product between vectors A and B, while $|A|$ and $|B|$ denote the Euclidean norms of vectors, used to normalize the dot product by the magnitudes of the vectors.

1.2 Generative AI and Large Language Model

Language Models (LMs), based on neural networks, have a structure that estimates the probability distribution of sequences of tokens and are predictors of the subsequent token in a sequence. The sentence *"I am going to the park"* is more probable than *"am going I the park to"*.
Any artificial intelligence model that generates novel data, information, or documents, such as images and explanations of complex concepts or stories, can be considered a Generative AI model. In the training phase, it learns from an existing dataset and then creates new data consistent with that context, with similar characteristics.

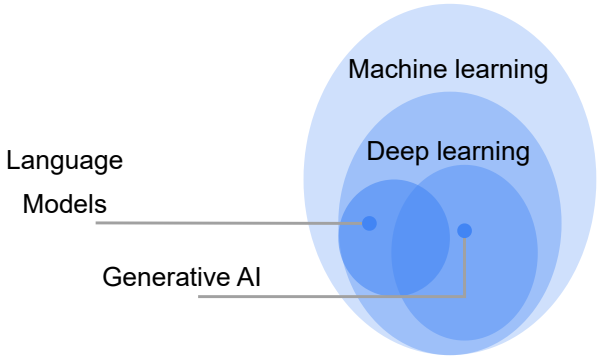


Figure 1.3: Relationships diagram between machine learning, deep learning, language models, and generative AI.

LMs for Generative AI achieve this by recursively asking for the most likely next word given all our previous words. This natural language generation technique is autoregressive because text generation depends on the preceding tokens and allows the model to produce text that maintains coherence and continuity.
These models leverage advanced deep learning neural network architectures like RNNs or modern transformers.

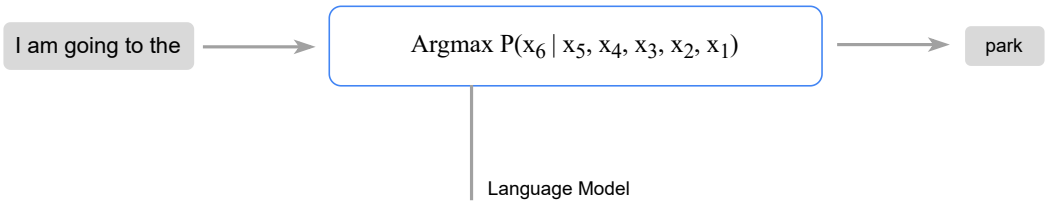


Figure 1.4: Representation of autoregression, iteratively predicting the most probable next word.

The core process is self-supervised training, a learning paradigm that leverages the data's inherent structure to generate training labels, enabling models to find and learn statistical patterns in massive unannotated text, reducing dependence on manually annotated data, which is both limited in quantity and often costly to obtain. Numerous text datasets containing billions or trillions of tokens are available, sourced from books, web pages, code, scientific articles, and more. The LLM input text is known as a prompt, and the space in the number of tokens available for it is called the context window, typically large enough for a few thousand words but differs from model to model.
Important studies demonstrate how scaling the number of parameters significantly increases the performance of an LM, introducing scaling laws such as the KM scaling law [2] and the Chinchilla scaling law [3]. It is observed that as LM reach larger dimensions, they "unlock" new capabilities, often referred to as *emergent abilities* [17], which are not present in smaller models and are not directly trained for. These include mathematical skills, complex reasoning, and one of the most important, In-Context Learning (ICL), the ability to learn from analogies and task examples. However, recent research has shed light on the fact that *emergence* is often caused by the wrong choice of scales[1] or evaluation metrics [18]. These abilities are not limited to larger models, as their performance experiences a smooth improvement with increasing parameters and appear to *emerge* only under metrics that non-linearly or discontinuously scale the error rate per token of any model. So, *emergent abilities* may not be a fundamental property of the scaling model parameters.
LLMs are used for many tasks, including text generation, summarization, classification, translation, question answering, named entity recognition, etc. However, despite their remarkable performance, they present various risks and limitations. Training data are mainly web pages containing numerous questions followed by answers. Many of these are correct, but there are also many false facts and wrong answers to common sense questions. Due to the massive

[1] `https://betterprogramming.pub/there-are-no-emergent-abilities-in-llms-2bb42e17ce7e`

volume of this data, it is almost impossible for the researchers to control the content to which LLMs are exposed during training. Therefore, text generation may occasionally exhibit undesirable behavior, such as reproducing harmful or biased content and generating hallucinations by producing non-existent or false facts.

1.2.1 Transformers

To accomplish the ambitious objective set by NLP, researchers previously relied on RNNs. However, these neural networks had several limitations, including their inefficiency with long text inputs and the inherent sequential structure that hindered efficiency. In 2017, Google introduced Transformers [1], a novel architecture paradigm to address these restrictions.

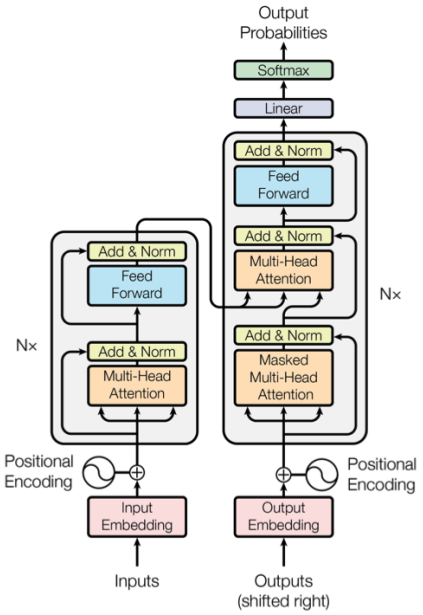


Figure 1.5: Transformers architecture.
From [1]

The encoder and decoder, consisting of N stacked layers, are the main components and work together (Figure 1.5). Initially, a tokenized input embedding is generated, a parallel process that also needs positional embeddings to preserve the information about the word order, which is critical for natural language understanding, and then is passed to the encoder. The encoder transforms a sequence of input symbol representations $(x_1, ..., x_n)$ into a sequence of continuous representations $z = (z_1, ..., z_n)$. With these representations, the decoder proceeds to generate an output sequence $(y_1, ..., y_m)$ in an autoregressive

way. Each encoding layer has two sub-layers: one for multi-head self-attention and another for a feed-forward network. At the same time, the decoder has an additional multi-head attention layer to perform cross-attention with the encoder stack output.

The paper's title, "Attention is all you need", is clear about the concept behind the architecture, the attention:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_K}}) \tag{1.2}$$

The query Q represents the input token, information description about what you are searching for, K are the keys, a feature used to compare with other elements, and V are the values; query, keys, and values are all matrices. The output is a weighted sum of the values based on the attention scores, where each attention score represents the relevance or similarity between a query and a key.

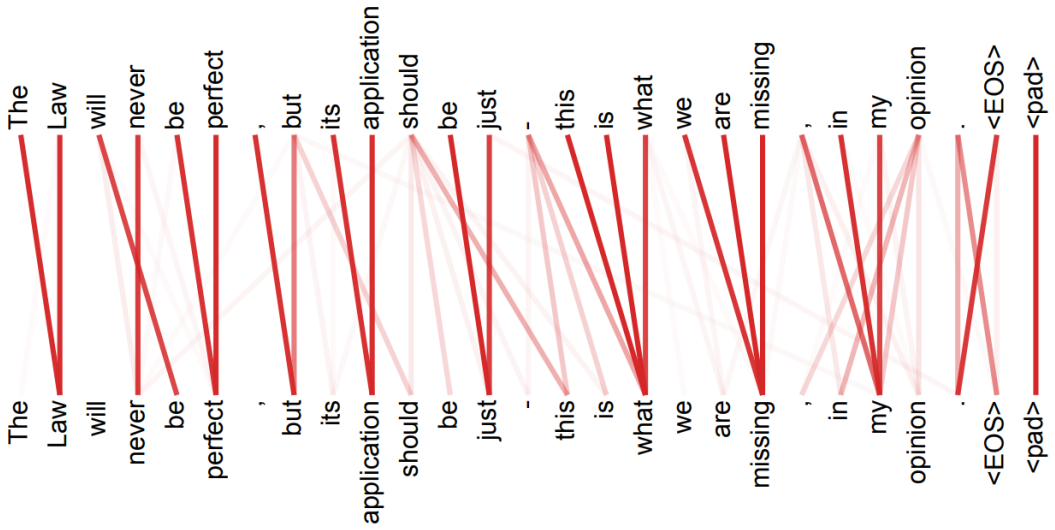


Figure 1.6: Attention map, an illustration of weights between each word and every other word. The color intensity indicates the relationship weight.

Multi-head attention, with several layers running in parallel, allows the model to capture diverse patterns and relationships within the input, where each head focuses on different aspects. The architecture presents this in three ways:

1. *Cross-attention*: mechanisms where the queries come from previous decoder layers while the memory keys and values come from the output of the encoder, allowing it to capture the relevant information from the encoder's output while generating the target sequence.

2. Encoder *self-attention*: in each layer, all keys, values, and queries are derived from the preceding output, allowing every position to attend to all positions in the previous encoder layer.

3. Decoder *self-attention*: allow each position in the decoder to attend to all positions, including itself and those that come before it.

Self-attention allows the transformer architecture to learn the relevance and context of all words in a sentence. This ability to find deep relationships across the whole input significantly improves language encoding, bringing transformers to take an important gap with the previous architecture.
The attention weights, randomly initialized, are learned during model training.

1.2.2 Pre-training and Fine-tuning

The architecture model is initialized with random weights called parameters that require a training phase to learn and understand the natural language. At inference time, only the initial token is the input, passed once to the encoder and then to the decoder, which autoregressively generates a sentence. Thus, the keys and values in the decoder's cross-attention layer, originating from the encoder, remain consistently the same. At training time, the decoder's input differs and depends on the task.
To avoid the need to start the process from scratch each time to understand the semantics of the language, GPT [6] and BERT [19] propose two stages of development to make training less resource-intensive and less expensive.

Pre-training This training is run once without a specific objective and is resource-intensive and expensive. It is necessary to ensure that the model understands the semantics and grammar of a language and the meaning of words and thus acquires the ability to manipulate it. Various tasks self-supervised are performed during pre-training, like:

1. a token in a sentence is masked in decoding, and the model has to find the right word to insert based on the context

2. mixed the sentence token, the model has to reconstruct the original sequence

3. given two sentences, understand if one can follow the other

These tasks are performed with the constraint that decoding attention can only consider tokens generated up to that point. Any subsequent tokens are masked to preserve the autoregressive nature. Pre-training requires a large dataset,

generally trillion tokens of unlabeled text, and the quality and diversity of these data influence the final performance.

Fine-tuning Taking a pre-trained model, fine-tuning consists of training it for a specific task, and a model can be fine-tuned several times, producing different models. The training data consists of input-output samples for tasks such as summarization and QA, and it requires a smaller dataset than pre-training, resulting in less expensive supervised training. Moreover, fine-tuning can be utilized to enable a pre-trained model to acquire knowledge in specific domains like biomedicine or law. ICL can be useful in teaching simple tasks to an LLM with limited results and without updating model parameters, which is essential when we do not have direct access to the model but can only do inference. On the contrary, the feature-based approach focuses on embedding created by the pre-trained model and using them as input features to train a classification model. In contrast to the two previous methods, real fine-tuning updates something in the pre-trained model, such as:

1. Add output layers (Fine-tuning i): the parameters of the pre-trained model are frozen, and only newly added output layers are trained.

2. Updating all layers (Fine-tuning ii): even though fine-tuning requires fewer resources than pre-training, the larger the model, the more expensive it is to update its layers.

3. Parameter-efficient fine-tuning (PEFT): it consists of several parameter-efficient mechanisms that train only a small number of parameters with performance comparable to full fine-tuning.

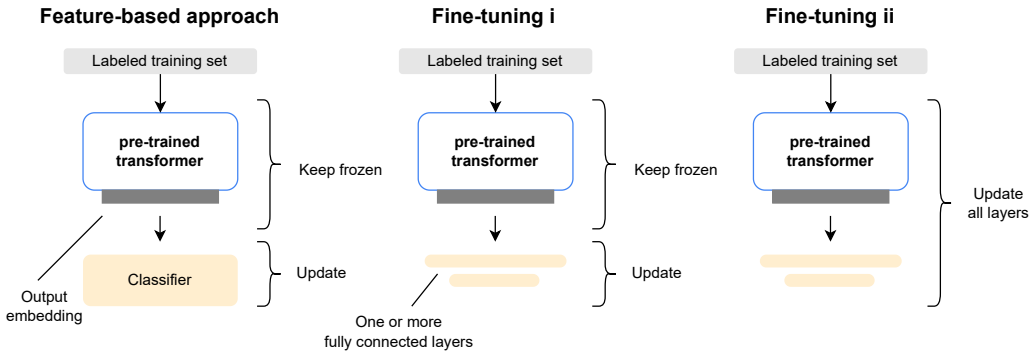


Figure 1.7: Illustration of different trained layer in feature-based approach, finetuning i and finetuning ii.

1.2.3 Encoder-Decoder, Encoder-only, Decoder-only

The advent of self-attention transformer models has given rise to three distinct architectural configurations that play a pivotal role in various language understanding and generation tasks. These configurations include encoder-only, encoder-decoder, and decoder-only models derived from the foundational transformer architecture.

Encoder-only This model takes the input sequence of text and gives in the output a fixed-length vector that represents the text (embedding) and can be a vector for the entire sequence or a different vector for each word. The vectors describe the meaning of the sequence, and this can be used for several downstream tasks focused on the context meaning or for labeling each word in the text, such as sentiment analyses or named entity recognition. It is common to execute fine-tuning to this type of model to be able to solve specific tasks. The encoder-only model cannot generate text but can only understand it by performing bidirectional attention, where each token can pay attention to any other token within the provided sentence.

Encoder-decoder This type of model, also called sequence-to-sequence (seq2seq), is the original transformer architecture, composed of both the encoder that takes an input sequence and generates an embedding and the decoder that generates a sequence one token at a time. The most common tasks this model solves are text translation and summarization, and the famous models are T5 or BART. The main difference between the encoder-decoder models and encoder-only or decoder-only is the cross-attention between the two components to merge the source and target sequences.

Decoder-only Many recent LLMs are based on decoder-only architecture for text generation, similar to encoder-decoder but without the first component. These models, which are more straightforward and have notably fewer parameters, take an input sequence and start to produce the next most probable token in an autoregressive manner until it generates an end token.

1.2.4 Prompt Engineering

Prompting is the way to control an LLM where the instruction is given as a text prefix, for example, "*Translate the following text in Greek:*". The model takes only a natural language description of the task and completes the sentence, generating the corresponding Greek sentence. There are different methods for prompting an LLM:

- *Zero-shot*: the instructions and the data are specified in a structured prompt
- *One-shot*: in addition to data and task description, at the beginning of the prompt is preset one example of the task and the correct answer
- *Few-shot*: like one-shot but with few examples of the task
- *Chain-of-thought (CoT)*: enhances complex reasoning abilities by trying to generate intermediate reasoning steps (e.g., *Let's think step by step*). It is possible to combine it with few-shot prompting to get a better result.

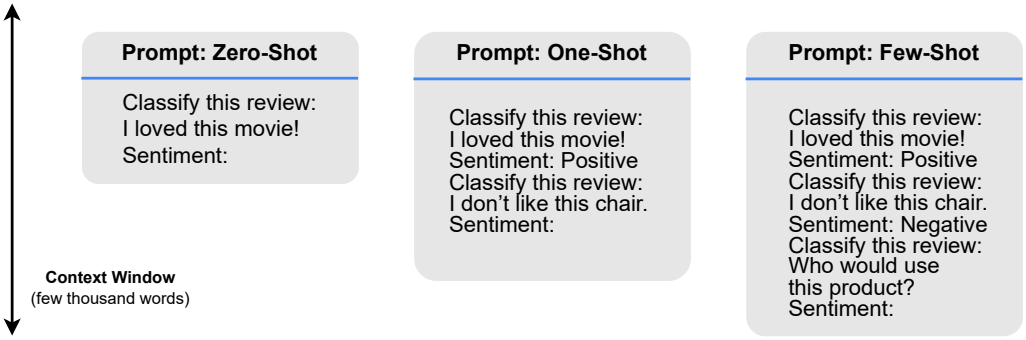


Figure 1.8: Example of zero-shot, one-shot, and few-shot prompting to solve a sentiment analysis task.

Prompting is a general technique used to solve a lot of problems (Figure x), but it is hard to control the generated output because there isn't a distinction between instruction and input. To address this, many LLMs are fin-tuned with a template of the prompt, which distinguishes various sections such as:

- *System*: used to indicate a general instruction or a context to the LLM and help to clarify the following input.
- *Instruction / User*: used to indicate a sequence of specific instructions that must be followed by an LLM and the input data
- *Assistant*: after this, the LLM generates the text output

Prompt engineering leverages the ICL ability of an LLM, but it's also possible to do instruction-tuning, which allows more control over text generation for different tasks.

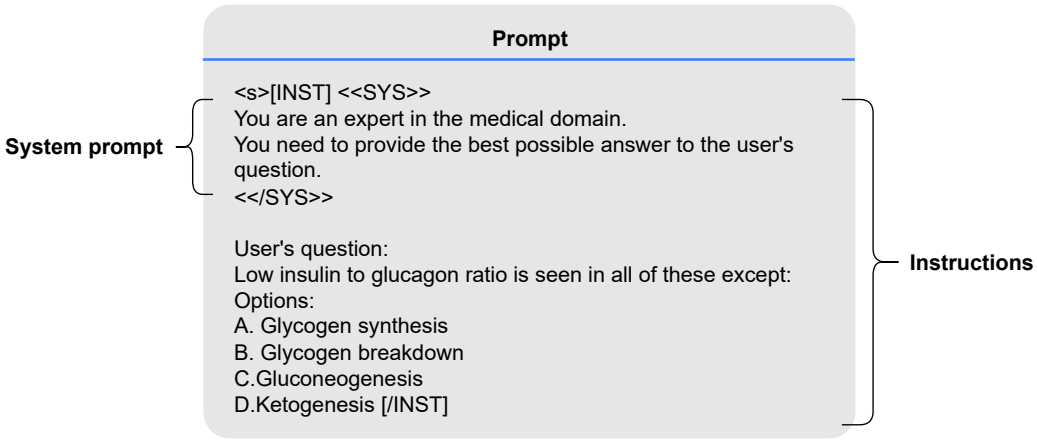


Figure 1.9: In LLaMA2-chat is possible to indicate sections among different tags, "[SYS] text [/SYS]" for system prompt and "[INST] text [/INST]" for instruction prompt.

1.3 Open and closed book question answering

The question-answering (QA) is a common task that involves prompting an LLM with a question and expecting it to produce a correct and coherent response. QA has two sub-categories, closed-domain question-answering (CDQA), which covers questions confined to a specific domain or topic and the system has access to a limited set of documents, and open-domain question-answering (ODQA), where the question can cover diverse subject areas and it is necessary general and vast knowledge to answer. In this last category, an additional distinction can be made:

1. *closed-book question-answering*: the system can only rely on internal knowledge sources represented by the dataset used to train the LLM.

2. *open-book question-answering*: the system has access to extensive external knowledge sources like databases or the internet and can search and retrieve useful information. Typically, these are used to augment the context of the LLM.

Focusing on open-book QA, the sources from which to retrieve relevant information are typically stored in a vector database, and the search occurs through similarity metrics between question and document chucks. Instead, the information is retrieved within some LLMs and utilized with different readers in the thesis. Our QA system, which utilizes LLMs for document generation and reading, combines elements of both open-book and closed-book approaches. We can think of multiple heads collaborating to answer a question.

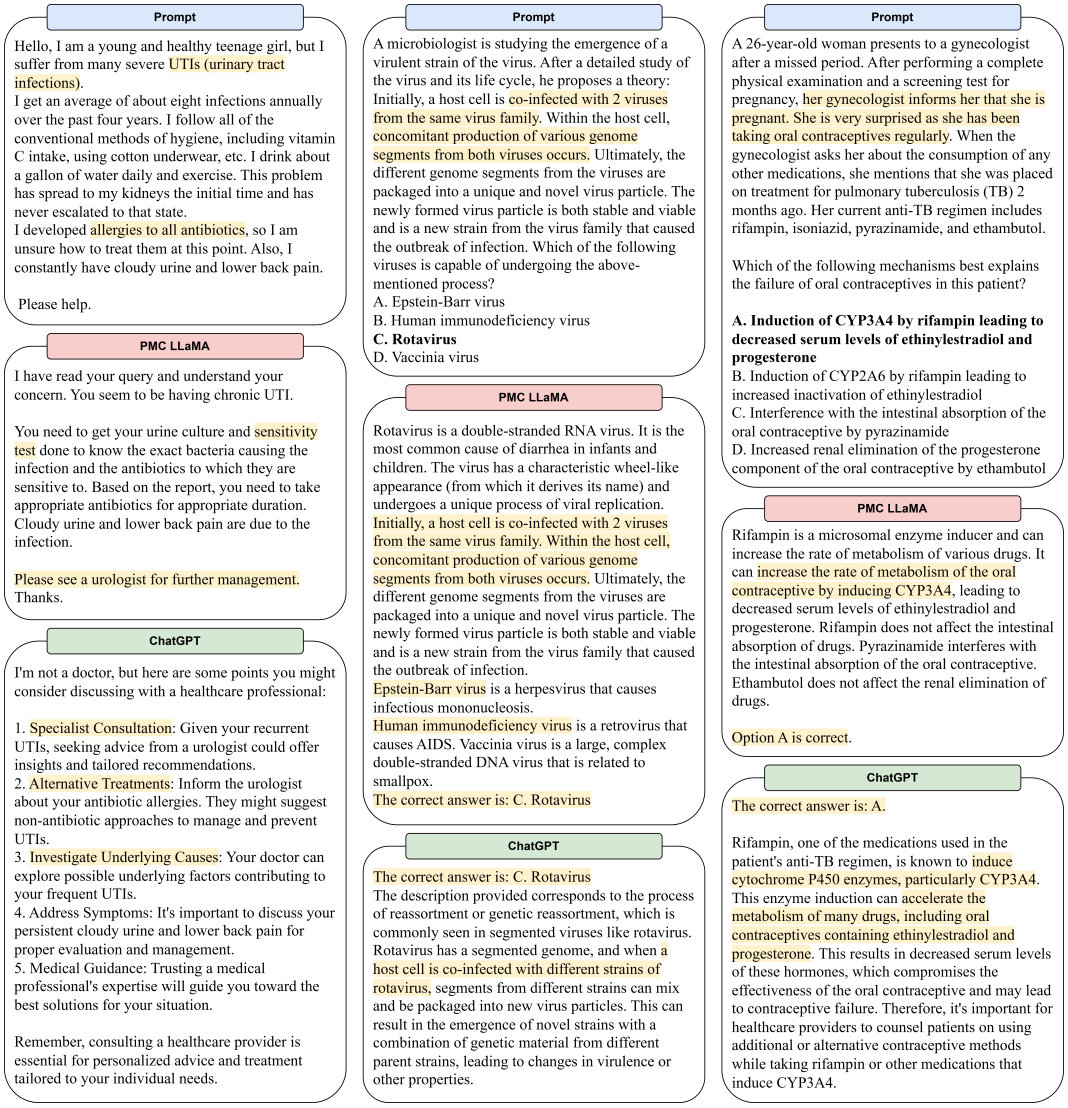


Figure 1.10: An example of a QA task on the left and two examples of MCQA tasks on the right, along with the answers provided by PMC-LLaMA and ChatGPT.
From [15]

Chapter 2

Related Work

This chapter clarifies the existing solution to retrieve information for ODQA and describes the LLMs used in this thesis.

2.1 Retrieve-then-read

Mainstream methods for solving ODQA are based on a retrieve-then-read pipeline that first retrieves relevant passages for a given question from a database, and then they are passed to a reader, which predicts the final answer. Recent work has focused on the retriever or the reader to improve this pipeline, such as the sparse retriever Best Match 25 (BM25) [20] and the dense passage retriever (DPR) [21]. Extractive readers select passages or sentences that directly contain the answer, such as DPR-reader [21] while generative readers produce the answer, based on the context, in a sequence-to-sequence manner, such as Fusion-in-Decoder (FiD) [16] and Retrieval-Augmented Generation (RAG) [22].

2.1.1 Dense retriever

In contrast to BM25, which ranks documents based on their relevance by considering TF-IDF and document length, DPR introduces neural network-based methods to obtain text embeddings using a bi-encoder (also called two-tower) architecture. This fixed-length vector, also called a dense vector, can capture complex semantic relationships between queries and documents.

1. **Encoding**: the first step is encoding $E_P(\cdot)$ once M text passage to a d-dimensional vector space (M can be very large, e.g., millions of passages) and building the document index. Then, at run-time, the query is encoded $E_Q(\cdot)$ to a d-dimensional vector (note E_P and E_Q are different encoders).

2. **Similarity score**: to select k relevant passages is calculated the dot product similarity between query and passages embedding and the closest are selected:

$$sim(q, p) = E_Q(q)^T E_P(p) \tag{2.1}$$

3. **Encoders training**: the encoders have to generate embeddings so that relevant pairs of questions and passages are closer than irrelevant pairs. The training dataset D comprises m instances, each consisting of a query q_i, one relevant passage p_i^+, and n irrelevant passage $p_{i,j}^-$:

$$D = \{q_i, p_i^+, p_1^-, \ldots, p_{i,n}^-\}_{i=1}^m \tag{2.2}$$

The loss function to optimize is defined like this:

$$L(q_i, p_i^+, p_{i,1}^-, \ldots, p_{i,n}^-) = -\log \frac{e^{sim(q_i, p_i^+)}}{e^{sim(q_i, p_i^+)} + \sum_{j=1}^n e^{sim(q_i, p_{i,j}^-)}} \tag{2.3}$$

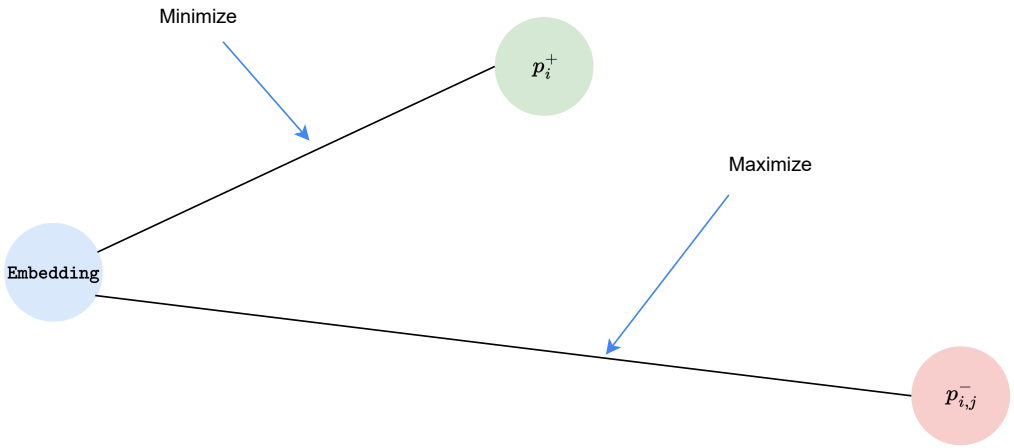


Figure 2.1: Dense retriever encoders training has to maximize the distance between the embedding and the irrelevant passages ($p_{i,j}^-$) and minimize the distance with relevant passages (p_i^+).

2.1.2 Fusion-in-Decoder

The Fusion-in-Decoder (FiD) [16] reader is based on a seq2seq (encoder-decoder) model pre-trained on unsupervised data, such as T5 [23]. After recovering k relevant passage, these are provided together with the input question, and the model generates the answer:

1. **Encoding**: each relevant passage and its title are concatenated with the question and then are encoded independently, producing k embedding per question. This allows scaling the k passages, and the computation time grows linearly with this number.

2. **Fusion**: the concatenation of the resulting representations is passed to the cross-attention layer of the decoder, where the evidence merges (fusion-in-decoder) to generate the final answer.

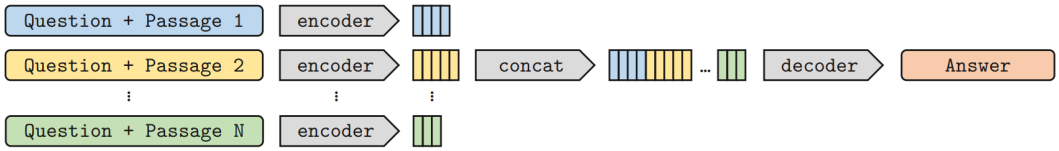


Figure 2.2: Architecture of the Fusion-in-Decoder method.

2.1.3 FiDO

The FiD architecture, based on minimal changes to the T5 model, has a suboptimal allocation of resources for retrieval-augmented tasks. This assigns a significant portion of computational resources to the encoder, while the primary bottleneck in inference time is memory bandwidth limitations in the decoder. This is mainly because the size of the encoder and decoder is similar, but a large number of passages are fed to the encoder, and the decoder performs multi-head cross-attention over a large input sequence. FiD optimized (FiDO) [24] proposes two changes to the architecture to alleviate memory bandwidth constraints:

1. using multi-query attention[1] instead of multi-head attention, eliminates the memory-bandwidth bottleneck

2. rebalance the encoder and decoder size, reducing the encoder to extract information from passages and using a larger decoder for reasoning and generating the answer.

[1]Multi-query attention [25] is an approach to reducing the keys and values matrix size by sharing a single set of keys and values in different attention heads.

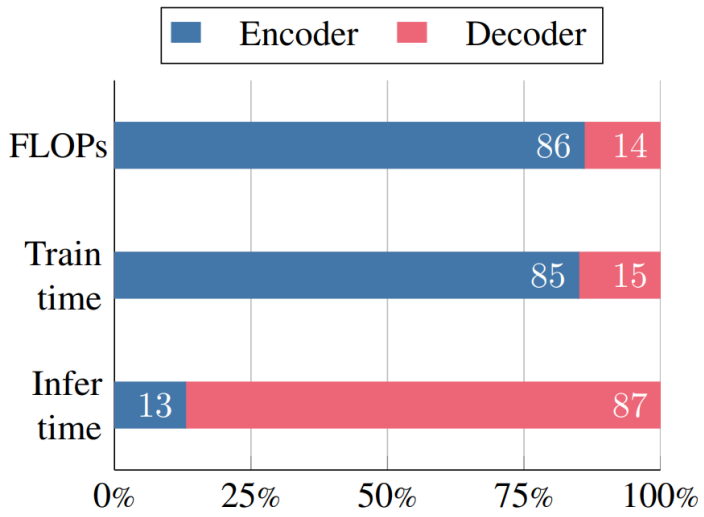


Figure 2.3: Shows the percentage of Floating Point Operations (FLOPs) in forward pass, training time, and inference time for the encoder and decoder for a Fusion-in-Decoder model with 40 retrieved passages and batch size 24. The vast majority of FLOPs and training time originate from the encoder, but the decoder is much more expensive for inference.
From [24]

2.1.4 RFiD

The FiD architecture assigns equal importance to all passages, using only a cross-attention layer to establish correlations between the decoder and encoders. Rational FiD (RFiD) [26] determines which passages contain the answer and generates different embeddings for these and irrelevant passages. The input is the same as FiD, question, and k passages.

1. **Rational passage**: includes at least one answer span from all golden answers. The encoder classifies this, presenting an additional binary classifier on top of the first token's hidden states.

2. **Decoding**: additional embeddings are appended to the encoder result and fed to the decoder to produce the final answer.

2.1.5 DPR-FiD

The original DPR utilized an extractive reader from the retrieved passages to produce the answer. Some studies [16, 27] propose a DPR-FiD pipeline that merges dense retriever with generative reader fusion-in-decoder.

1. **Retriever**: initially, query and passage are encoded, respectively, with $E_Q(\cdot)$ e $E_P(\cdot)$ encoders and then are retrieved the top-k passages based on the dot product similarity score between the generated dense vectors.

2. **Reader**: the FiD reader takes input query q and the previous k passages p, encodes each pair $q - p_i$, and passes the concatenation to the decoder to generate the answer.

2.2 Generate rather than retrieve

Generate-then-read (GenRead) [9] is an alternative to retrieval-then-read pipeline that does not access an external source but only relies on the LLM and its internal knowledge. Two methods are provided:

1. **Zero-shot**: query the LLM to generate a background document supporting a question and then augment the context window with these and ask the LLM to answer the question.

2. **Supervised settings**: the first step is fine-tuning a FiD reader with multiple background documents generated using clustering-based prompts. Subsequently, documents are generated similarly at inference time, and the trained reader is used to answer.

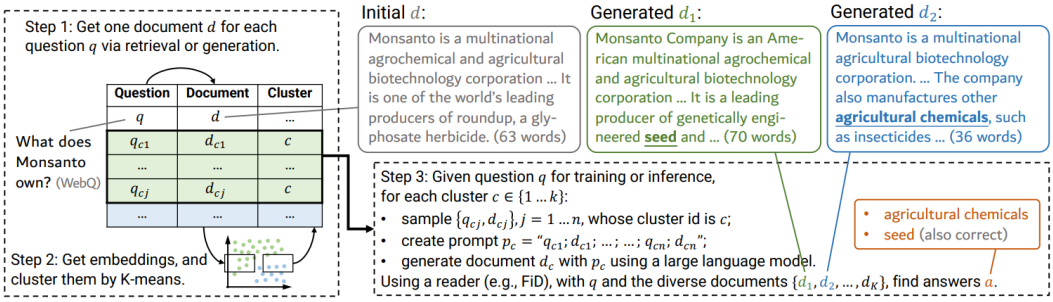


Figure 2.4: An overall framework of clustering-based prompting method. It leverages distinct question-document pairs sampled from each embedding cluster as in-context demonstrations to prompt a large language model to generate diverse documents, then read the documents to predict an answer.

From [9]

The clustering-based prompts method (Figure 2.4) is proposed to generate different documents:

1. Generate one document for each question with the zero-shot method or alternatively retrieve it.

2. Encode the pair document-question and use K-means to cluster all embeddings into K sets.

3. For each set, select n random samples and use the document-question pair of each to create a prompt with an in-context demonstration.

In GenRead, InstrucGPT is used as LLM and is tested on general question-answer datasets. Both LLM and embedding model use non-open-source technology. With a smaller open-source model, knowledge is more limited, and generating long documents is not always easy. Moreover, the biomedical domain with multi-choice question-answering has not been analyzed.

2.3 LLMs

Here is a brief overview of the pre-trained and fine-tuned LLMs utilized and evaluated in this thesis. As there is no precise definition of LLM, in this project, we categorize models with billions of parameters as "Large".

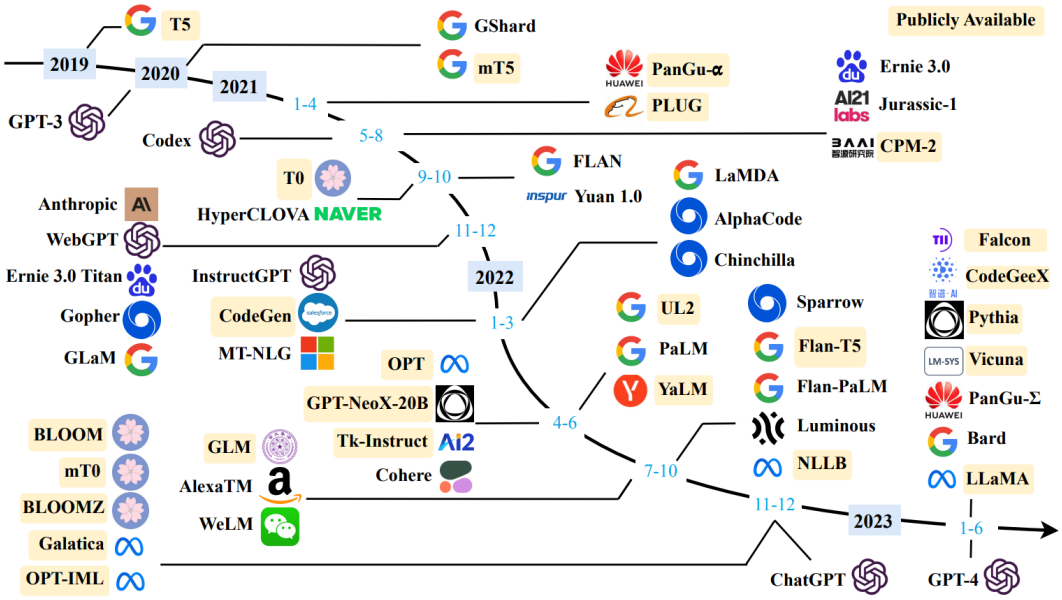


Figure 2.5: A timeline of existing large language models in recent years. LLMs yellow marked have publicly available model checkpoints and are reported only, which include publicly reported evaluation results.

From [4]

2.3.1 LLaMA 2

LLaMA 2 [14] is a general-purpose pre-trained model on publicly available datasets exclusively, without resorting to proprietary datasets. The training

data sources include Wikipedia, books, ArXiv, Github, and StackExchange. In particular, the last two data types commonly raise ICL abilities. We use the 7B parameters model, chat fine-tuned with reinforcement learning from human feedback (RLHF) available on HuggingFace,[2] for the chatbot reader role. It has a context window of 4096 tokens, so it can take input prompts with several in-context demonstrations.

2.3.2 BioMedLM

BioMedLM[3] is a domain-specific model with 2.7B parameters based on GPT-2 and fine-tuned on all PubMed abstracts and full documents from The Pile [28]. It has a context window of 1024 tokens, enough to generate background documents but not for prompt with in-context demonstrations. Furthermore, the model struggles to follow the prompt instructions and is fine-tuned for downstream question-answering tasks.

2.3.3 PMC-LLaMA

PMC-LLaMA [15] proposes a series of open-source models based on LLaMA [5], specifically designed for medical applications. The most important models are MedLLaMA_13B,[4] pre-trained on a medical corpus, and PMC_LLaMA_13B[5] further fine-tuned on instruction following dataset. This last model, which can follow prompt instructions better than the other, achieves important results under zero-shot inference in MedMCQA.

2.3.4 DoctorGPT

DoctorGPT is an open-source project that proposes an LLM based on LLaMA 2, mdellama2_7b,[6] that is fine-tuned on a medical dialogue dataset and improved using reinforcement learning and constitutional AI [29].

[2] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf
[3] https://huggingface.co/stanford-crfm/BioMedLM
[4] https://huggingface.co/chaoyi-wu/MedLLaMA_13B
[5] https://huggingface.co/axiong/PMC_LLaMA_13B
[6] https://huggingface.co/llSourcell/medllama2_7b

Chapter 3

Method

This chapter describes and analyzes the proposed method (Figure 3.1) to solve multi-choice question-answering in the biomedical domain using open-source LLM and ensembling approaches, leveraging the GenRead pipeline.

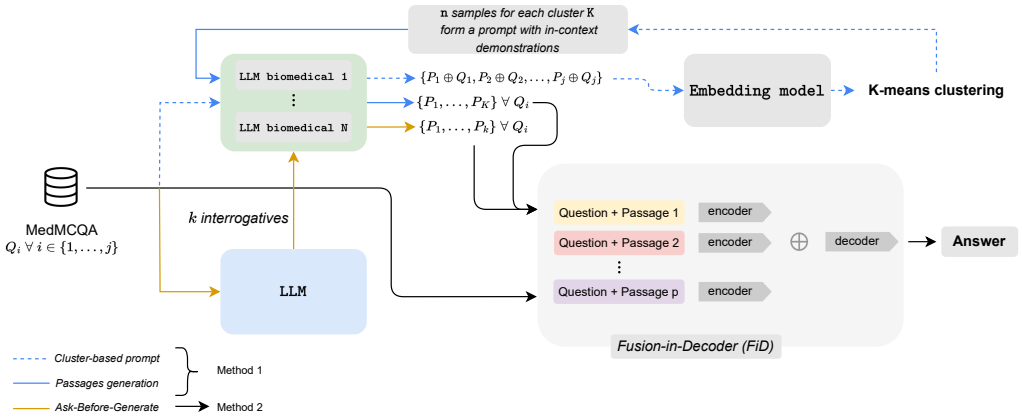


Figure 3.1: Diagram to illustrate the proposed method. Passages P generation consists of (i) performing cluster-based prompt (GenRead) to create K different prompt with n in-context demonstrations for each question Q, (ii) using Ask-Before-Generate pipeline to produce a passage for each k interrogative generated by an LLM, it also performed for each Q. These two generation methods are performed for each biomedical fine-tuned LLM N_i. All passages P are then passed to the Fusion-in-Decoder reader to generate the final answer.

3.1 HuggingFace Models

The GenRead implementation is extended to support the HuggingFace models because, originally, the inference was made by calling OpenAI APIs. Using transformer models from HuggingFace must load the model and the

tokenizer locally after logging in to the hub if required. The model can be loaded with different data types for the parameters or quantized, important configurations with limited hardware.[1] In the implementation, the use of API or local model is managed by an input flag.
The function that initializes the model is reported in Figure 3.2.

```
def init_model(model_id, load_in_8bit=False):
    # There is an error with AutoTokenizer and some LLaMA-based models
    if "llama" in model_id.lower():
        tokenizer = LlamaTokenizer.from_pretrained(model_id)
        model = LlamaForCausalLM.from_pretrained(model_id, device_map="auto",
        ↪  load_in_8bit=True)
        return model, tokenizer

    tokenizer = AutoTokenizer.from_pretrained(model_id)
    tokenizer.pad_token = tokenizer.eos_token
    model = AutoModelForCausalLM.from_pretrained(model_id, device_map="auto",
    ↪  torch_dtype=torch.float16, load_in_8bit=load_in_8bit)
    return model, tokenizer
```

Figure 3.2: Function to initialize HuggingFace model and tokenizer.

3.2 Document Generation

The preliminary step is the generation of background documents for each question and is denoted as *step1*.

3.2.1 Prompt

Before generation, it is also important to define the input prompt. GenRead proposes one like "*Provide a background document from Wikipedia to answer the given question. {query}* ", where "*{query}*" represent the placeholder for the input question. This prompt works well to produce long documents with InstructGPT and question-answer datasets. Giving only the question is not enough for a multi-choice question-answering dataset because often, without the options is hard to understand the context. So, is used a prompt like "*Provide a background document from Wikipedia to answer the given quiz. Quiz:{query} A) {opa} B) {opb} C) {opc} D) {opd} Documents:*"; it presents additional option placeholders.
For prompting LLaMA 2, spacial tokens and tags[2] have also been added, as

[1] A model with 13B parameters in full-precision (32bit) requires approximately 42GB of RAM, in half-precision (16bit) 21GB and quantized in 8bit 12GB.

[2] Special tokens are utilized in training like "<s> </s>" to denote the beginning and end of a sequence in the context, while tags are separator characters like "[INST] [/INST]"

reported in the official paper [14], to specify the system and instruct section in the prompt.
All prompts are saved in a JSON file, one for a row. In addition to the prompt, for each one, the following is indicated:

- prompt id
- the dataset type and the step for which it is used
- the number of special token characters, if necessary

An example of prompt information format is reported in Figure 3.3.

```
1 {"type": "multi choice qa", "task": "step1", "pid": 2, "prompt": "Provide a background
  ↪  document from Wikipedia to answer the given quiz.\n\nQuiz:\n{query}\nA) {opa}\nB)
  ↪ {opb}\nC) {opc}\nD) {opd}\n\nDocuments:"}
2 {"type": "multi choice qa", "task": "step1", "pid": 4, "special_tokens": 3, "prompt":
  ↪  "<s>[INST] <<SYS>>\nGenerate a background document from Wikipedia to answer the
  ↪  user's quiz.\n<</SYS>>\n\nUser's quiz:\n{query}\nOptions:\nA. {opa}\nB. {opb}\nC.
  ↪ {opc}\nD. {opd} [/INST]"}
```

Figure 3.3: Format of prompt information in JSON file.

3.2.2 LLMs

After loading the model and selecting the prompt, it is time to proceed with document generation.
The dataset is loaded in an array of Python `dict`, each representing information about a question such as options, correct option, etc. The prompt for each question is built by the function in Figure 3.4. The hyperparameters to control the inference are specified in the GenerationConfig,[3] example in Figure 3.5.
For this step, "max_new_tokens" is set to 300, approximately 230 words, allowing the LLM to generate a long document.
Finally, the prompt is tokenized and, along with the generation configuration, is passed to the inference method of the model (Figure 3.6).
The output also contains the input prompt, and to obtain only the document generated, it is necessary to omit it by using a slicing. It is important to know the number of special token characters in the prompt because they are not present in the output, so it is necessary to subtract this number from prompt length when performing slicing.
A background document is generated for each question and the output is saved along with other information in a JSON file (Figure 3.7).

[3]GenerationConfig is a new class recommended by HuggingFace to control the behavior of the model generation method instead of passing every single hyperparameter directly as method parameters.

```python
def add_prompt(item, prompt):

    def rmreturn(s):
        s = s.replace('\n\n', ' ')
        s = s.replace('\n', ' ')
        return s.strip()

    query = item['question']
    prompt = prompt.replace('{query}', query)

    if (prompt.find("{opa}") != -1) and (prompt.find("{opd}") != -1):
        prompt = prompt.replace('{opa}', item["opa"]).replace('{opb}',
        ↪  item["opb"]).replace('{opc}', item["opc"]).replace('{opd}', item["opd"])

    if item.get('output'): # background info
        backinfo = rmreturn(item['output'])
        prompt = prompt.replace('{background}', backinfo)

    if (prompt.find("{subject}") != -1):
        prompt = prompt.replace("{subject}", item.get("subject_name", "medicine"))
        topic = f", {item['topic_name']}" if  item.get("topic_name") else ""
        prompt = prompt.replace("{topic}", topic)

    return prompt
```

Figure 3.4: Function to create the prompt, "item" contains the question information, and the "prompt" is a template. The function substitutes each placeholder with the relative data if it is present.

```python
gen_config = GenerationConfig(
    max_new_tokens=max_tokens,
    temperature=temp,
    pad_token_id=pad_token_id
)

if beam_search == 1: # Normal beam search
    gen_config.num_beams = num_beams
elif beam_search == 2: # Diverse beam search
    gen_config.num_beams = num_beams
    gen_config.num_beam_groups = num_beam_groups
```

Figure 3.5: GenerationConfig initialization used in the project.

3.2.3 Clustering-based prompt

The clustering-based prompt method proposed by GenRead is useful for producing a document that covers different perspectives, and using smaller LLMs for document generation is also useful for teaching the task to the model through in-context demonstration.
Same as document generation, GenRead used a GPT-3 [6] model to perform this method, resulting in a 12,288-dimensional vector. Embedding models on HuggingFace reach a maximum number of dimensions that hovers around

```python
def run_inference(input_with_prompt, model, tokenizer, generation_config):
    # Encoding input prompt
    input_ids = tokenizer(input_with_prompt,
    ↪  return_tensors="pt").input_ids.to(model.device)
    # Generate
    outputs = model.generate(input_ids, generation_config=generation_config)
    # Decode the output
    output_decoded = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return output_decoded
```

Figure 3.6: Function to tokenize the input prompt, infer the model, and detokenize the output.

```python
...
output_file.write(json.dumps({
    'question': ...,
    'answer': ...,
    'opa': ...,
    'opb': ...,
    'opc': ...,
    'opd': ...,
    'cop': ...,
    'output': output_decoded[len(input_prompt) - prompt_special_tokens : ]})
    +'\n')
```

Figure 3.7: Saving the model output and additional details within a JSON file.

1024 and can be loaded using the SentenceTransformers[4] library, example in Figure 3.8.

```python
from sentence_transformers import SentenceTransformer
embed_model = SentenceTransformer(model_id)
```

Figure 3.8: Example of an embedding model from HuggingFace initialization using SentenceTransforme library.

After loading the embedding model and generating the background documents with the previous method (3.2.2), the creation of the prompt consists of 3 steps.

1. Generate an embedding for each question-document pair. The answers for each question must have already been generated using the chatbot reader (3.3.1). Only documents that produce a correct response are selected. Then, the question and document concatenation is passed to the model and the embeddings are created (Figure 3.9).

2. Create the cluster. Using the K-means algorithm and the embeddings previously generated, the k clusters are created (Figure 3.10).

[4] `https://www.sbert.net/`

The embeddings are merged in a matrix, where each row is a question-document embedding and the columns are the vector dimensions. The "numpy" library is optimized for matrix calculation, so the K-means algorithm performs more efficiently. After the cluster creation, the labels for each pair are saved to indicate which cluster it belongs to.

3. Prompt creation. Each cluster is selected n random samples as in-context demonstrations pre-appended to the prompt template, and for each is created a new prompt. An example of a created prompt for a cluster is reported in Table 3.1

```
for opt in chatbot_answers:
    if opt == correct_option:
        # document that is used to produce this answer is selected
...
input = ' '.join([question, document])
embedding = embed_model.encode(input)

# Save to file
```

Figure 3.9: Example of encoding the question-document concatenation for step 1 of the clustering-based prompt.

```
matrix = np.vstack(embeddings_list)
kmeans = KMeans(n_clusters=10,          # Number of cluster to generate
                init='k-means++',       # Method for centroids initialization
                random_state=42         # For reproducibility
        )
kmeans.fit(matrix)
labels = kmeans.labels_

# Save to file the label for each question
```

Figure 3.10: Example of clustering the embeddings using the K-Means algorithm for step 2 of the clustering-based prompt.

3.2.4 Ask-Before-Generate

In this thesis, the new pipeline to address the short document generation of LLMs with limited parameters is named Ask-Before-Generate. Through multiple questions asked to an LLM, it is possible to generate a longer document than to ask directly to generate a background document.

1. Quality of the interrogatives is important to generate a document that can be useful to answer the question. This step is also performed by

Prompt

```
Provide a background document from Wikipedia to answer the given quiz.
Quiz:
Coenzyme A contains which of the following vitamins:
A) Biotin
B) Pyridoxine
C) Pantothenic acid
D) Niacin
Documents:  Coenzyme A (CoA) is a coenzyme found in all living cells.  It plays a
crucial role in the metabolism of fatty acids, amino acids, and other compounds.
CoA is synthesized from pantothenic acid, a B vitamin.


Provide a background document from Wikipedia to answer the given quiz.
Quiz:
Which of the following is an essential fatty acid?
A) Linoleic acid
B) Alpha linolenic acid
C) Both of the above
D) None of the above
Documents:  Essential fatty acids (EFA) are a class of fatty acids that are
required by the body for proper functioning.  They cannot be produced by the
body and must be obtained through diet.  The two main types of EFAs are omega-3
and omega-6 fatty acids.  Omega-3 fatty acids are found primarily in fish and
other seafood, while omega-6 fatty acids are found in vegetable oils and nuts.
Linoleic acid is an omega-6 fatty acid, while alpha linolenic acid is an omega-3
fatty acid.  Both are essential fatty acids.


Provide a background document from Wikipedia to answer the given quiz.
Quiz:
{query}
A) {opa}
B) {opb}
C) {opc}
D) {opd}
```

Table 3.1: Example of a prompt created for a specific cluster using a cluster-based prompt method and two examples.

an LLM with prompts like "*Generate {n} questions to be asked of an expert to know more about the following question ...*". Every interrogative generated is extrapolated from the output and saved in a file.

2. Next, each question is asked to another LLM, and the answers are concatenated to form a single document (Figure 3.11).

This is also useful because each question can refer to a different aspect of the question.

3.3 Reader

This represents the final step (*step2*) in producing an answer based on the previously generated contextual document. A reader, obtainable through

```
...
template_prompt = ...
for question in dataset:
     ...
    output_LLM = ""
    for interrogative in interrogatives_for_question:
        prompt = template_promt.replace("{interrogative}", interrogative)
        # LLM generation
        output_LLM += "\n" # Answers concatenation

    # Save to file the document generated
```

Figure 3.11: Ask-Before-Generate prompt creation.

various methods, takes the question, options, and document as input and then generates the final answer.

3.3.1 Chatbot (decoder-only)

A chatbot is an LLM fine-tuned for dialog systems and following user instructions, commonly based on decoder-only architecture. Using this model as the reader is useful when the documents are not too long because it has a limited context window. It is mainly used with one document that augments the prompt context.
Various methods have been employed to generate and evaluate the final answer:

1. **ICL**: some examples of the MCQA task are inserted in the prompt to teach the LLM to generate only the letter of the correct option (Table 3.2). In this case, setting the "max_new_tokens" hyperparameter of generation to 1 is also useful.

   This approach also simplified response evaluation, which consists of comparing only the output answer and the letter of the correct option and then calculating the accuracy, an example in Figure 3.12.

   However, using ICL in the prompt reduces the available context window and introduces noise that can affect the response.

2. **Structured prompt**: in the prompt, more instructions allow the model to understand the task well. Moreover, it can be specified to generate a JSON string, where the information can be easily extrapolated and evaluated (Table 3.3).

   LLMs like GPT-4 with this type of prompt experience improvement,[5] but smaller LLMs struggle to follow all instructions.

[5] `https://www.kaggle.com/code/max2020/using-gpt4-the-art-of-prompt-engineering`

3. **Fine-tuning**: is an alternative to train an LLM for the specific task, but is more memory intensive and requires time. Initially, it is necessary to create a dataset of input-output and then set the hyperparameters. LLaMA 2, fine-tuned with MCQA examples, sometimes doesn't generate a token representing one of the possible letters, which is impossible to evaluate.

```python
def eval_multi_choice_qa(data_questions):
    correct_answers = 0

    for line in data_questions:
        correct_option = str(line["cop"]).replace("1", "A") \
                                         .replace("2", "B") \
                                         .replace("3", "C") \
                                         .replace("4", "D")
        llm_option = line["output"]

        if correct_option.strip() == llm_option.strip():
            correct_answers += 1

    accuracy = round(correct_answers / len(data_questions), 4)

    return accuracy, correct_answers
```

Figure 3.12: Function to evaluate the accuracy of the generated answers.

Prompt
Question: Respiratory rhythm generation center is located at: A: Dorsal respiratory group B: Pre-Botzinger complex C: Ventral respiratory neurons D: Pneumotaxic center Answer:B Question: The zygomatic bone does not articulate with: A: Frontal bone B: Maxillary bone C: Nasal bone D: Temporal bone Answer:C Refer to the passage below and select the correct answer. Passage: background Question: query A: opa B: opb C: opc D: opd Answer:

Table 3.2: Example of using in-context learning with two task examples in the prompt.

Prompt

```
Refer to the passage below and select the first char (A, B, C, D) of the best
options for the user's question.
Don't write anything but provide the result in a single JSON with the following
format
{
"best_answer":  ["."],
"best_answer_explanation":["..."]
}

Passage:
{background}

User's question:
{query}
Options:
A. {opa}
B. {opb}
C. {opc}
D. {opd}
```

Table 3.3: Example of prompt with several instructions for MCQA.

3.3.2 FiD (encoder-decoder)

The FiD architecture is created specifically to answer a question with several contextual documents. It is observed that increasing the number of steps from 10 to 100 leads to an improvement in different QA datasets [16]. The training data consists of input questions, correct answers, and k documents for each question. The data format proposed by FiD is reported in Figure 3.13. The "target" is the correct answer for training, while the other answers are considered for evaluation.
For the MCQA task, it is necessary to add the options along with the input question, and including instructions can also prove beneficial. The "answers" are three common model responses:

1. the letter of the correct option
2. the correct answer
3. the letter of the correct answer followed by the answer

An example of this format is reported in Figure 3.14.
So, the Fid reader can answer a question considering documents generated from different LLMs and with different methods.

3.3.3 Sentence-similarity (encoder-only)

An alternative approach to solve the MCQA task involves utilizing embeddings and sentence similarity, employing encoder-only models like BERT [19].

```
{
  "id": "0",
  "question": "What element did Marie Curie name after her native land?",
  "target": "Polonium",
  "answers": ["Polonium", "Po (chemical element)", "Po"],
  "ctxs": [
            {
                "title": "Marie Curie",
                "text": "them on visits to Poland. She named the first chemical element
↪  that she discovered in 1898 \"polonium\", after her native country. Marie Curie ..."
            },
            {
                "title": "Marie Curie",
                "text": "was present in such minute quantities that they would eventually
↪  have to process tons of the ore. In July 1898, Curie and her husband ..."
            }
          ]
}
```

Figure 3.13: Format example for FiD input data.

Questions often lack meaningful context without the options, making it ineffective to encode only the questions and select the most similar option.
To use an embedding model as the reader, the generated background document is considered instead of the question. After the document encoding, each option is also encoded, and sentence similarity is measured using the cosine similarity metric between the two embedding vectors. The option with the highest similarity is then determined as the correct answer.
It is also possible to consider multiple documents for a question, considering the sum of the similarities between each document and the options.

```
{
  "question": "Indicates the letter of the correct answer.
                Question: Cofactor involved in metabolism of sulfur containing amino
↪  acids:
                Option A: Biotin
                Option B: Vitamin B2
                Option C: Vitamin B1
                Option D: Vitamin B12
                Answer:",
  "target": "D",
  "answers": ["D", "Vitamin B12", "D. Vitamin B12"],
  "..."
}
```

Figure 3.14: Format example for FiD input data in MCQA task.

Chapter 4

Experiments

This chapter provides a comprehensive explanation of several experiments conducted. It also describes the experimental environment and the various LLMs employed.

4.1 Environment

Loading and executing the LLMs requires sophisticated hardware. Preliminary experiments are conducted on Google Colab but it has execution time limits. The university has a cluster of four servers, and the main one is composed of four NVIDIA GeForce RTX3090 GPUs with 24GB of dedicated memory, 124 GB of RAM, and an AMD EPYC 7443 24-core processor. SLURM [30] technology manages the cluster and allows the scheduling and allocation of tasks on GPUs dynamically. Furthermore, a Docker container executed with bash script was useful to manage all library dependencies.

4.2 Dataset

MedMCQA [13] is an MCQA medical dataset that contains more than 194k entrance exams of AIIMS and NEET PG. The features are:

- **id**: question identifier
- **question**: the question
- **opa**: answer option A
- **opb**: answer option B
- **opc**: answer option C

- **opd**: answer option D
- **cop**: the number of the correct option, not the letter
- **choice_type**: defines whether the answer is single or multiple choice
- **exp**: expert explanation of the answer
- **subject_name**: medical domain of the question
- **topic_name**: specific topic

The dataset is provided split into a training set (182,822), a development set (4,183), and a test set (6,150). The test set doesn't contain the correct option, so the development set is used instead for the experiments.

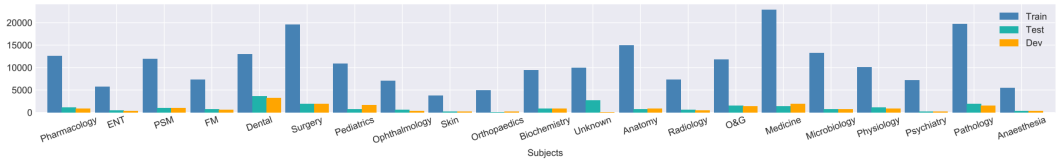


Figure 4.1: Distribution of unique tokens in the union of train, test, and development set in MedMCQA dataset.

From [13]

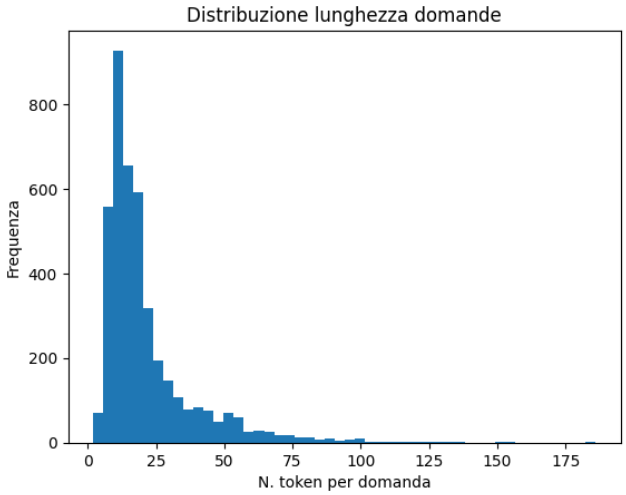


Figure 4.2: Distribution of question length in tokens in the development set.

4.3 Zero-shot inference

Various pre-trained or medical fine-tuned LLMs are tested to show their ability and knowledge to answer without augmented context (Table 4.3). The first of several prompts is a few-shot with two task examples (PID-1 in Table 4.1).

Prompt

```
Question:  Respiratory rhythm generation center is located at:
A: Dorsal respiratory group
B: Pre-Botzinger complex
C: Ventral respiratory neurons
D: Pneumotaxic center
Answer:B

Question:  The zygomatic bone does not articulate with:
A: Frontal bone
B: Maxillary bone
C: Nasal bone
D: Temporal bone
Answer:C

Question:  {query}
A: {opa}
B: {opb}
C: {opc}
D: {opd}
Answer:
```

Table 4.1: Few-shot prompt for zero-shot inference (PID-1).

Hyperparameter	Value
max_new_tokens	1
temperature	1

Table 4.2: Generation hyperparameter for zero-shot inference. The ones that were not included have been retained with their default values.

The previous prompt constricted the model to generate only the letter. While the accuracy of models like *LLaMA2-7b-chat-hf* are in line with those reported in other studies [15], the results of *BioMedLM* seem suboptimal (Table 4.7), although there are no other studies on this dataset to compare with. So, for this model, it also tested a prompt that only asked to answer the question (PID-4 in Table 4.4).

To extrapolate the correct option in the output, regular expressions (RegEx) like `Answer:\s*([A-Z])` are used, determined by analyzing the output. However, the output is not always formatted in the same way and it's hard to find the answer, leading to a further drop in accuracy (Table 4.7).
Examples of output where it is possible to extrapolate the answer and another where it is hard are reported in Table 4.5.

LLM	Split	Parameter	Mem.	Time	Accuracy
Human expert	dev	-	-	-	90.0
Random model	dev	-	-	-	25.0
LLaMA2-7b-chat-hf	dev	16bit	15GB	10min	35.2
medllama2_7b	dev	16bit	15GB	10min	36.8
PMC-LLaMA_13B	dev	8bit	15GB	15min	48.5
medalpaca-13b	dev-1k	8bit	15GB	15min	29.0

Table 4.3: Comparing human expert, random model, LLaMA2-7b-chat-hf, medllama2_7b, PMC-LLaMA_13B and medalpaca-13b in zero-shot inference without augmented context. It also reported the dataset split and the size (dev-1k mean 1k questions from the dev split, while only dev means the whole set), the data type of the parameters (8bit mean quantization), the memory required on the GPU, the inference time and the final accuracy. Using PID-1 as the input prompt and hyperparameters in Table 4.2.

Prompt

`Question: {query}`
`A. {opa}`
`B. {opb}`
`C. {opc}`
`D. {opd}`
`What is the correct option?`

Table 4.4: Simple prompt without in-context learning (PID-4).

Answer can be extrapolated
`A. B. C. D. E. Answer: C. Saltatory conduction of impulses is seen. The nerve is covered by myelin sheath. The myelin sheath is a multilayered structure that surrounds the axon. The sheath is composed of a central layer of compact myelin and an outer layer of loosely packed myelin. The compact myelin is formed by the wrapping of the axon with the plasma membrane of the Schwann cell. The loosely packed myelin is formed by the wrapping of the axon with the plasma membrane of the oligodendrocyte. The compact myelin is electrically insulating, whereas`
Answer cannot be extrapolated
`A. The oncotic pressure of the fluid leaving the capillaries is greater than that of fluid entering it B. Glucose concentration in the capillaries is greater than that in glomerular filtrate C. Constriction of afferent aeriole increases the blood flow to the glomerulas D. Hematocrit of the fluid leaving the capillaries is less than that of the fluid entering it Answers #sec1-1 ======= A. The oncotic pressure of the fluid leaving the capillaries is greater than that of fluid entering it B. Glucose concentration`

Table 4.5: Examples of BioMedLM output, one where the answer is easy to extract and another where it is hard.

For *medllama2_7b* and *LLaMA2-7b-chat-hf* are conducted other experiments with LLaMA special tags and more instruction in the prompt (PID-3) in Table 4.8 and the results setting "max_new_tokens" to 512 are reported in Table 4.9.
Both models generate output with JSON strings not always formatted correctly. So, the convenience of a JSON format cannot be leveraged, and reliance on regex is still necessary, which decreases accuracy.

PID	Hyperparameter	Value
PID-1	max_new_tokens	1
	temperature	1
	repetition_penalty	1
PID-4	max_new_tokens	150
	temperature	0.7
	repetition_penalty	1.1

Table 4.6: Generation hyperparameter for BioMedLM inference using different prompts.

LLM	PID	Parameter	Mem.	Time	Accuracy
BioMedLM	PID-1	16bit	6GB	5min	24.4
	PID-4			3:30H	16.8

Table 4.7: BioMedLM results using different prompts in the development set and hyperparameters in Table 4.6.

Prompt

```
<s>[INST] «SYS»
You are an expert in the bio-medical and medical domain.
«/SYS»

You need to provide the best possible answer to the user's question.

As an expert, think step by step to answer the question with an in-depth
explanation.
Then, summarize the answer in fewer than 100 words.
Finally, let the expert select the first char (A, B, C, D) of the best option for
the question.

Don't write anything but provide the result in a single JSON with the following
format:
{
"expert":"...",
"expert_answer_summarized":"...",
"best_answer":["."],
"best_answer_explanation":["..."]
}

User's question:
{query}
Options:
A. {opa}
B. {opb}
C. {opc}
D. {opd} [/INST]
```

Table 4.8: Prompt with several instructions and LLaMA special tags (PID-3).

4.4 Generate-then-read

Chatbot as reader For this experiment, the pipeline name comprises the first model that generates the background document and the second model

LLM	PID	Parameter	Mem.	Time	Accuracy
medllama2_7b	PID-3	16bit	15GB	5H	12.26
LLaMA2-7b-chat-hf	PID-3	16bit	15GB	5H	34.4

Table 4.9: medllama2_7b and LLaMA2-7b-chat-hf results using the specific prompt PID-3.

that is the reader and produces the final answer (*first_model-second_model*). The results using PID-2 for document generation and PID-1 for answering are reported in Table 4.10. Furthermore, several prompts are analyzed for a specific pipeline in Table 4.11.
To run these pipelines, we call a bash script passing the dataset split, the two models, and the prompt ID for each step (Figure 4.3).

```
#!/bin/bash

split="$1"
model_step1="$2"
model_step2="$3"
pids1="$4"
pids2="$5"
pipeline="${model_step1##*/}-${model_step2##*/}"

python3.8 mainfunc.py --dataset "medmcqa" --task "step1" --split "$split" --modeltype
↪  "huggingface" --idmodel "$model_step1" --pipeline "$pipeline" --pids1 "$pids1"
python3.8 mainfunc.py --dataset "medmcqa" --task "step2" --split "$split" --modeltype
↪  "huggingface" --idmodel "$model_step2" --pipeline "$pipeline" --pids1 "$pids1"
↪  --pids2 "$pids2"
```

Figure 4.3: Bash script to run the generate-then-read pipeline.

LLMs	Split	Prm	Mem	Time	Accuracy
BioMedLM-LLaMA2-7b	dev	16bit 16bit	6GB 15GB	6H	36.9
medllama2_7b-LLaMA2-7b	dev	16bit 16bit	15GB 15GB	4H	39.0
medllama2_7b-LLaMA2_13b	dev	16bit 8bit	15GB 15GB	4H	40.0

Table 4.10: Comparing different models in generate-then-read pipeline. It also reported the dataset split and the size (dev-1k mean 1k questions from split dev, while only dev means the entire set), the data type of the parameters (8bit mean quantization), the memory required on the GPU, the inference time and the final accuracy.

LLM	Split	PromptsID	Accuracy
medllama2_7b-LLaMA2-7b	dev	PID-2 / PID-1	39.0
		PID-4 / PID-3	36.8
		PID-5 / PID-1	38.0
		PID-6 / PID-1	39.1

Table 4.11: medllama2_ 7b-LLaMA2-7b results using different prompt for each step.

FiD as reader To use FiD reader, the background document is generated with *medllama2_ 7b* and with *PMC-LLaMA_ 13B*. These are the models that, in previous tests, have shown the best performance.

The whole training set is extensive to generate documents for each question, considering the inference time in Table 4.10. So, a stratified subset composed of 4000 questions is selected from the training set, and seven documents for each question are generated using a cluster-based prompts method employing *medllama2_ 7b*. The stratification is obtained using the "splitstackshape"[1] package and considering the "subject_name" and "topic_name" features to have a distribution of questions that reflects the whole set.

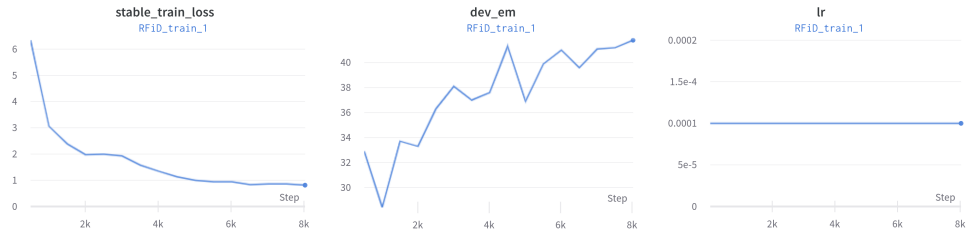
Next, we conduct two FiD training, particularly using the RFiD implementation and "wandb"[2] logging:

- first training with 4000 step (Figure 4.4) and the hyperparameters in Table 4.12
- second training with 8000 step (Figure 4.5) and the hyperparameters in Table 4.13

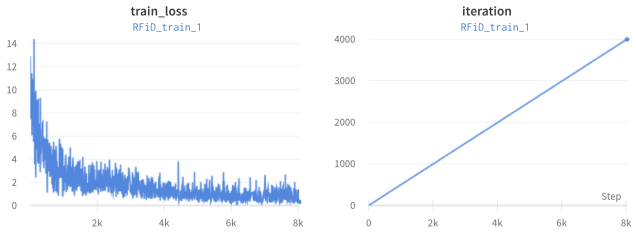
In both training, the model is evaluated on a subset of the validation set every k step, and the best-performing model is selected at the end of the training. For the evaluation in the whole validation set, several configurations were tested; one of them includes seven documents generated with *medllama2_ 7b* and two documents with *PMC-LLaMA_ 13B*. The number of documents was selected considering the limitation of inference times.
As shown in Table 4.14, the model trained on 8000 steps improves accuracy by approximately 2 points.

Embedding model as reader To select the most relevant option to the document, the BertScore [31] automatic evaluation metric is used. Several embedding models were tested, and the results are reported in Table 4.15.

[1] `https://github.com/mrdwab/splitstackshape`
[2] `https://wandb.ai/site`

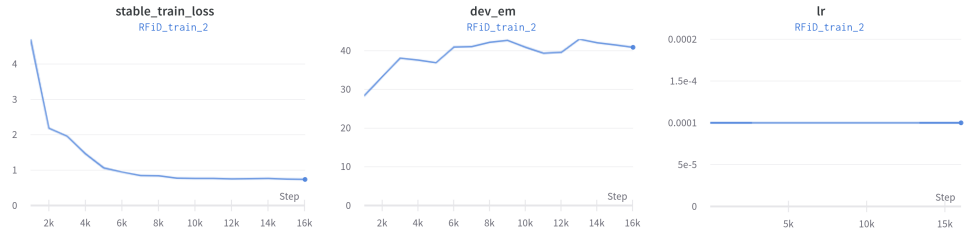


(a) Stable train loss *(b) Accuracy dev set* *(c) Learning rate*

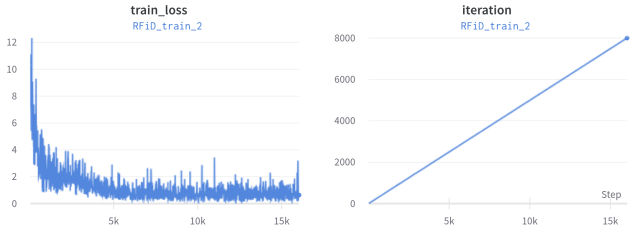


(d) Train loss *(e) Iteration*

Figure 4.4: RFiD training (1) progress using the hyperparameters in Table 4.12 ("wanb" logging).



(a) Stable train loss *(b) Accuracy dev set* *(c) Learning rate*



(d) Train loss *(e) Iteration*

Figure 4.5: RFiD training (2) progress using the hyperparameters in Table 4.13 ("wanb" logging).

4.4.1 Ask-Before-Generate

Using this pipeline, the name of each experiment comprises the first model that produces k interrogative, the second model which provides the answers,

Hyperparameter	Value
per_gpu_train_batch_size	1
per_gpu_eval_batch_size	1
accumulation_steps	64
total_steps	4000
eval_freq	250
add_loss	binary
cat_emb	True
model_size	large

Table 4.12: Hyperparameters value for RFiD training (1).

Hyperparameter	Value
per_gpu_train_batch_size	1
per_gpu_eval_batch_size	1
accumulation_steps	64
total_steps	8000
eval_freq	500
add_loss	binary
cat_emb	True
model_size	large

Table 4.13: Hyperparameters value for RFiD training (2).

and the third model that generates the final answer with augmented context (*first_model-second_model-third_model*). This pipeline is used with the chatbot reader (Table 4.17) and to increase the number of documents with the FiD reader (Table 4.14).

4.4.2 Expert explanation as document

An expert explanation from the dataset is used as the background document to verify whether the model can extract information from the augmented context accurately. By doing this, we ensure that the document contains the correct answer. Considering only the question where this explanation is present, 2206 out of 4183 in the development set, the accuracy reaches 70% with LLaMA2-7b-chat-hf, demonstrating good but not excellent reading skills from the context.

Context	N. Context	RFiD	Accuracy
medllama2_7b	7	1	41.09
medllama2_7b	7	2	42.46
PMC-LLaMA_13B	2	1	47.83
PMC-LLaMA_13B	2	2	49.94
medllama2_7b PMC-LLaMA_13B	7 2	1	43.17
medllama2_7b PMC-LLaMA_13B	7 2	2	45.11
medllama2_7b PMC-LLaMA_13B medllama2_7b (Ask-Before-Generate)	7 2 1	1	43.03
medllama2_7b PMC-LLaMA_13B medllama2_7b (Ask-Before-Generate)	7 2 1	2	45.14

Table 4.14: Results of RFiD in the validation set under different configurations. "Context" indicates which LLMs have generated the contextual documents, "N. context" the total number of documents, and "RFiD" indicates which trained model is used. For all tests, the hyperparameter "answer_max_length" is set to 30.

Embedding model	Time	Accuracy
BiomedNLP-PubMedBERT-base-uncased-abstract-fulltext	5H	32.22

Table 4.15: Results obtained using embedding models and BertScore. The documents are generated by medllama2_7b and the hyperparameters are reported in Table 4.16.

Hyperparameter	Value
lang	"en"
idf	True
rescale_with_baseline	True

Table 4.16: Hyperparameters configuration for BertScore.

LLMs	Prm	Mem	Time	Accuracy
Llama-2-7b-chat-hf	16bit	15GB		
medllama2_7b	16bit	15GB	16H	35.67
Llama-2-7b-chat-hf	16bit	15GB		

Table 4.17: Results using Ask-Before-Generate pipeline. It also reported the data type of the parameters, the memory required on the GPU, the inference time, and the final accuracy.

4.5 Data analysis

An analysis is provided to gain deeper insight into previous results.

4.5.1 Mixed options

Several tests with mixed options are performed to ensure that the chatbot deeply understands the question and generates an answer based on the knowledge.
Three mixes are evaluated:

1. **Default**: [“A”, “B”, “C”, “D”]
2. **Mix-1**: [“C”, “D”, “A”, “B”]
3. **Mix-2**: [“D”, “C”, “B”, “A”]

In zero-shot inference employing *LLaMA2-7b-chat-hf* without augmented context, despite the similar accuracy reported in Table 4.18, comparing the answers of the various mixes, we observe:

- **Default - Mix-1**: the model generate the same answer 31.8% of the time, of which 14.2% are correct
- **Default - Mix-2**: the same answers are 37.4%, of which 15.9% are correct
- **Mix-1 - Mix-2**: the same answers are 36.6%, of which 14.3% are correct
- **Default - Mix-1 - Mix-2**: the same answers 19.2%, of which 9.5% are correct

A similar result is obtained using the GenRead pipeline.

LLM	**Mix type**	**Accuracy**
LLaMA-2-7b-chat-hf	Default	35.2
	Mix-1	31.8
	Mix-2	31.3

Table 4.18: Results achieved through zero-shot inference without augmented context employing different options shuffles in the validation set.

4.5.2 Different prompts

Comparing the answers obtained using different prompts employing *LLaMA2-7b-chat-hf* in zero-shot inference without augmented context, we observe:

- **PID-1** and **PID-4**: the same answers are 50%, of which 25% are correct

Instead, using *medllama2_ 7b-LLaMA2-7b-chat-hf* pipeline with PID-1 and PID-4, but adding the background documents, we observe:

- the same answers are 75%, of which 31% are correct

4.5.3 Phoenix

Phoenix[3] is an Open Source ML Observability library. This tool can visualize vector embedding in a three-dimensional space through a dimensionality reduction algorithm and create clusters using "hdbscan" algorithm.
We use this tool to show different sets of embeddings, such as *input_prompt*, *documents*, and the pair *input_prompt-document*. The objective was to investigate the distinct clusters and attempt to identify any connections among the incorrect answers.
Considering the setting reported in Table 4.19 and the answer generate by *medllama2_ 7b-LLaMA2-7b-chat-hf*, the cluster with the lowest accuracy (19%) consists of:

- incorrect answers even when the documents clearly contain the correct answer
- incorrect or correct answers but which did not reflect the response contained in the document

Parameter	Value
min_cluster_size	10
cluster_minimum_samples	1
cluster_selection_epsilon	0
min_distance	0
n_neighbors	30
n_samples	1000

Table 4.19: Results achieved through zero-shot inference without augmented context employing different options shuffles in the validation set.

[3] `https://github.com/Arize-ai/phoenix`

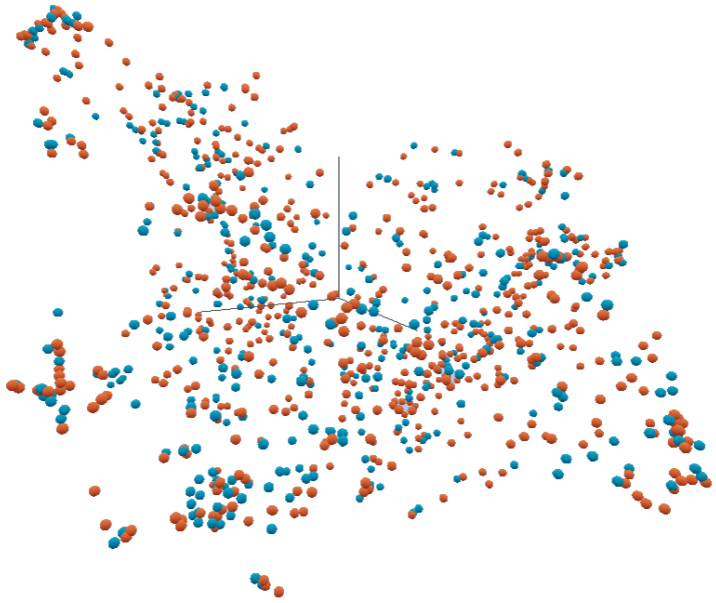


Figure 4.6: Prompt embeddings representation in Phoenix and results generated by medllama2_7b-LLaMA2-7b-chat-hf in the validation set. The color of the points indicates whether the generated answers are correct (blue) or not (red).

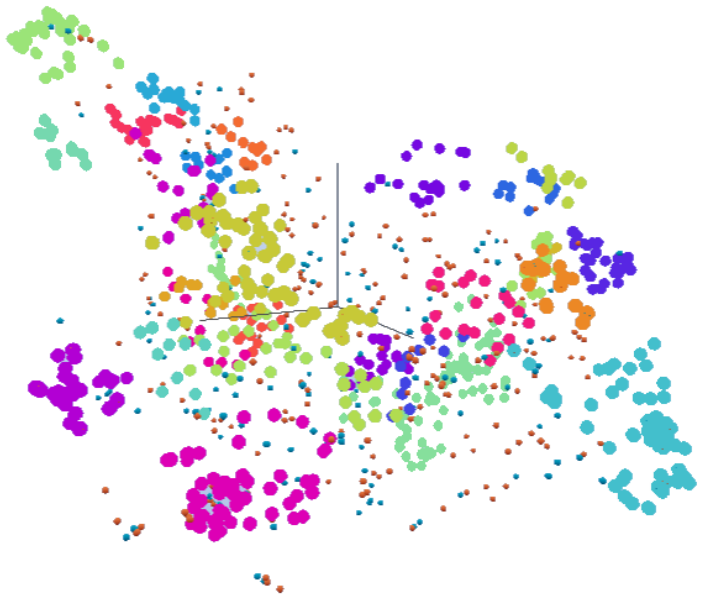


Figure 4.7: Highlighting clusters created by algorithm hdbscan.

Conclusions and Future Challenges

This thesis proposes systems that are based on language models with a limited number of parameters, in the order of tens of billions, and leverage the generated-then-read pipeline and ensembling approaches to answer multiple-choice questions in the biomedical domain. It demonstrates the effectiveness of these models as context generators, a crucial aspect of providing evidence to support responses generated in the medical field despite not having hundreds of billions of parameters. The solution using LLaMA 2 as a chatbot reader and medllama2 as a context generator improves the basic accuracy of both models by 2 points. Using instead FiD as a reader and multiple documents generated with the cluster-based prompt method, it is obtained an improvement of up to 5 points, proving that, even in this domain, text generation and the use of specialized models lead to better results than zero-shot inference. Through ensembling solutions, it is possible to leverage the knowledge and capabilities of each model. For instance, the *BioMedLM* model excels at generating high-quality contextual documents but has limitations in generating assessable answers, achieving an accuracy of 36.9% using the *BioMedLM-LLaMA2-7b-chat-hf* pipeline compared to the 24.4% in zero-shot inference. Furthermore, the proposed solutions address the challenge of domain adaptation, thus overcoming one of the limitations previously highlighted by the generate-then-read pipeline, as indicated by the authors of GenRead. Finally, while it is observed that this system is resilient to the use of different prompts, producing 70% of equal responses against 50% of a zero-shot inference, it shows poor robustness when changing the order of the prompt response options. During this project, we have witnessed and observed the development of innovative solutions and models, confirming the constant evolution and research in this field. Analyzing and testing the recent updates to the models used, conducting experiments with a larger number of generated documents, and delving into the analysis of responses generated by different methodologies and LLMs to identify which ones perform better in specific categories of questions will be the next steps towards an open-source system with low hardware requirements, capable of handling biomedical questions.

Conclusioni e Sviluppi Futuri

Questa tesi propone sistemi che si basano su language model con un numero ridotto di parametri, nell'ordine delle decine di miliardi, e che sfruttano la pipeline generate-then-read e approcci ensembling per rispondere a domande a scelta multipla in dominio biomedico. Si dimostra l'efficacia di questi modelli come generatori di contesto un aspetto cruciale per fornire evidenze a supporto delle risposte generate in ambito medico, nonostante non abbiano centinaia di miliardi di parametri. La soluzione che utilizza LLaMA 2 come chatbot reader e medllama2 come generatore di contesto migliora l'accuratezza di base di entrambi i modelli di 2 punti. Utilizzando invece FiD come reader e molteplici documenti generati con il metodo cluster-based prompt, si ottiene un miglioramento fino a 5 punti, dimostrando che, anche in questo dominio, la generazione di testo e l'utilizzo di modelli specializzati portano a risultati migliori rispetto a un'inferenza zero-shot. Mediante soluzioni ensembling si possono sfruttare le conoscenze e le capacità migliori di ciascun modello. Ad esempio, il modello *BioMedLM* è in grado di generare documenti contestuali di qualità ma presenta limitazioni nella generazione di risposte valutabili, ottenendo un'accuratezza di 36.9% attraverso la pipeline *BioMedLM-LLaMA2-7b-chat-hf* in confronto al 24.4% di un'inferenza zero-shot. Inoltre, le soluzioni proposte affrontano la sfida dell'adattamento a nuovi domini, superando così una delle limitazioni precedentemente evidenziate dalla pipeline generate-then-read, come indicato dagli autori di GenRead. In fine, mentre si osserva come questo sistema sia resiliente all'utilizzo di prompt diversi, producendo il 70% di risposte uguali contro il 50% di un inferenza zero-shot, si dimostra una scarsa robustezza al cambiamento dell'ordine delle opzioni di risposta nel prompt. Durante questo progetto, abbiamo assistito e osservato allo sviluppo di soluzioni e modelli innovativi, confermando la costante evoluzione e ricerca in questo settore. Analizzare e testare i recenti aggiornamenti ai modelli utilizzati, condurre esperimenti utilizzando un numero maggiore di documenti generati, e approfondire l'analisi delle risposte generate da diverse metodologie e LLM al fine di identificare quali producono risultati migliori in categorie specifiche di domande saranno i prossimi passi in direzione di un sistema open-source e a basse esigenze hardware, abile nell'affrontare domande biomediche.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[2] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[3] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models. *CoRR*, abs/2203.15556, 2022.

[4] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *CoRR*, abs/2303.18223, 2023.

[5] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler,

Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[7] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *CoRR*, abs/2203.02155, 2022.

[8] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311, 2022.

[9] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. *CoRR*, abs/2209.10063, 2022.

[10] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *CoRR*, abs/1705.03551, 2017.

[11] Yinghsan Chang, Mridu Narang, Hisami Suzuki, Guihong Cao, Jianfeng Gao, and Yonatan Bisk. WebQA: Multihop and Multimodal QA. 2021.

[12] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: a benchmark for question answering research. *Trans. Assoc. Comput. Linguistics*, 7:452–466, 2019.

[13] Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In Gerardo Flores, George H Chen, Tom Pollard, Joyce C Ho, and Tristan Naumann, editors, *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 248–260. PMLR, 07–08 Apr 2022.

[14] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

[15] Chaoyi Wu, Xiaoman Zhang, Ya Zhang, Yanfeng Wang, and Weidi Xie. Pmc-llama: Further finetuning llama on medical papers. *CoRR*, abs/2304.14454, 2023.

[16] Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. *CoRR*, abs/2007.01282, 2020.

[17] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *CoRR*, abs/2206.07682, 2022.

[18] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. Are emergent abilities of large language models a mirage? *CoRR*, abs/2304.15004, 2023.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[20] Stephen E. Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In W. Bruce Croft and C. J. van Rijsbergen, editors, *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum)*, pages 232–241. ACM/Springer, 1994.

[21] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics, 2020.

[22] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401, 2020.

[23] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[24] Michiel de Jong, Yury Zemlyanskiy, Joshua Ainslie, Nicholas FitzGerald, Sumit Sanghai, Fei Sha, and William W. Cohen. Fido: Fusion-in-decoder optimized for stronger performance and faster inference. *CoRR*, abs/2212.08153, 2022.

[25] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *CoRR*, abs/1911.02150, 2019.

[26] Cunxiang Wang, Haofei Yu, and Yue Zhang. Rfid: Towards rational fusion-in-decoder for open-domain question answering, 2023.

[27] Weijia Zhang, Svitlana Vakulenko, Thilina Rajapakse, and Evangelos Kanoulas. Scaling up query-focused summarization to meet open-domain question answering. *CoRR*, abs/2112.07536, 2021.

[28] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[29] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosiute, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemí Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. Constitutional AI: harmlessness from AI feedback. *CoRR*, abs/2212.08073, 2022.

[30] Morris A Jette and Tim Wickberg. Architecture of the slurm workload manager. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 3–23. Springer, 2023.

[31] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with BERT. *CoRR*, abs/1904.09675, 2019.