h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Hochschule Darmstadt

– Fachbereich Mathematik und Naturwissenschaften–

Few-shot prompting with large language models

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

vorgelegt von

Lorenc Zhuka

Matrikelnummer: 1112405

Referent	:	Prof. Dr. Markus Doehring
Korreferent	:	Prof. Dr. Sebastian Doehler

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, 15. November 2023

Lorenc Zhuka

ABSTRACT

OBJECTIVE: Recent advancements in the few-shot prompting field have shown considerable progress in scenarios with limited labeled data. However, the effectiveness of few-shot prompting methods significantly depends on the choice of in-context demonstrations. Various techniques, such as Auto-CoT, Active-CoT, and Retrieval-CoT have demonstrated their effectiveness, emphasizing diversity (Auto-CoT), uncertainty (Active-CoT), and similarity with the test example (Retrieval-CoT). Notably, while some methods, like Random-CoT and Auto-CoT, utilize labels generated by a large language model, Active-CoT relies on human-generated labels. This thesis aims to evaluate and compare several prompting methods using closed-source and open-source models in arithmetic reasoning tasks under both labeling scenarios: labels generated by GPT-3.5-turbo and human-generated labels. It also aims to develop new prompting methods by combining diversity, uncertainty, or similarity with the test example.

RESULTS: In experiments on GSM8K using GPT-3.5-turbo, Diverse-CoT, Retrieval-CoT, and Active-CoT outperform Random-CoT by margins of 1.1%, 1.7%, and 3.5%, respectively. Similarly, in evaluations on AQUA, Diverse-CoT and Active-CoT surpass the Random approach by margins of 3.9% and 2.7%, respectively. These results are consistent when GPT-3.5-turbo labels the questions. However, when human-generated labels are used, these methods achieve performance comparable to or even lower than Random baseline.

Additionally, I propose new methods: Diverse-Active-KMeansPlusPlus-CoT, which combines diversity and uncertainty, and Diverse-Active-KMeans PlusPlus-Retrieval-CoT, integrating similarity with the test question in the Diverse-Active-KMeansPlusPlus-CoT method. These new methods outperform the Random baseline by 1.9% and 2.5%, respectively, on GSM8K when using GPT-3.5-turbo-generated labels. On AQUA, Diverse-Active-KMeans PlusPlus-Retrieval-CoT surpasses Random by 3.3%.

Moreover, Falcon-40B-Instruct achieves 37.3% accuracy on GSM8K and 18.5% on AQUA, while Falcon-7B-Instruct achieves 5.4% on GSM8K and 11.4% on AQUA. This emphasizes that larger models perform better in a few-shot setting but exhibit inferior performance compared to GPT-3.5-turbo, with margins exceeding 30%. The code is available at https://github.com/Lori10/Master-Thesis-Few-Shot-CoT-Prompting-LLM.

ZUSAMMENFASSUNG

ZIEL: Aktuelle Entwicklungen im Bereich des Few-Shot Prompting haben deutliche Fortschritte in Szenarien mit begrenzten gelabelten Daten gezeigt. Die Effektivität von Few-Shot Prompting-Methoden hängt jedoch wesentlich von der Wahl der kontextbezogenen Beispiele ab. Verschiedene Techniken wie Auto-CoT, Active-CoT und Retrieval-CoT haben ihre Wirksamkeit unter Beweis gestellt, indem sie die Diversität (Auto-CoT), die Unsicherheit (Active-CoT) und die Ähnlichkeit mit dem Testbeispiel (Retrieval-CoT) hervorheben. Während einige Methoden, wie z.B. Random-CoT und Auto-CoT, von einem großen Sprachmodell generierte Labels verwenden, basiert Active-CoT auf von Menschen erstellten Labels. Ziel dieser Arbeit ist es, verschiedene Prompting-Methoden zu evaluieren und zu vergleichen, die Closed-Source- und Open-Source-Modelle in arithmetischen Denkaufgaben verwenden, und zwar unter beiden Labeling-Szenarien: Labels, die von GPT-3.5-turbo generiert werden, und von Menschen erzeugte Labels. Außerdem sollen neue Prompting-Methoden entwickelt werden, die Diversität, Unsicherheit oder Ähnlichkeit mit dem Testbeispiel kombinieren.

ERGEBNISSE: In Experimenten auf GSM8K mit GPT-3.5-Turbo übertreffen Diverse-CoT, Retrieval-CoT und Active-CoT den Random-CoT-Ansatz mit einer Differenz von 1,1%, 1,7% bzw. 3,5%. In ähnlicher Weise übertreffen Diverse-CoT und Active-CoT bei den AQUA-Evaluierungen den Random-Ansatz mit einer Differenz von 3,9% bzw. 2,7%. Diese Ergebnisse sind konsistent, wenn GPT-3.5-turbo die Fragen labelt. Werden jedoch von Menschen erstellte Labels verwendet, erreichen diese Methoden eine Performance, die mit der des Random-Ansatzes vergleichbar oder sogar niedriger ist.

Zusätzlich schlage ich neue Methoden vor: Diverse-Active-KMeansPlusPlus-CoT, das Diversität und Unsicherheit kombiniert, und Diverse-Active-KMeans PlusPlus-Retrieval-CoT, das die Ähnlichkeit mit der Testfrage in die Diverse-Active-KMeansPlusPlus-CoT-Methode integriert. Diese neuen Methoden übertreffen die Random-Baseline auf GSM8K um 1,9% bzw. 2,5%, wenn sie mit GPT-3.5-Turbo-generierten Labels eingesetzt werden. Auf AQUA übertrifft Diverse-Active-KMeansPlusPlus-Retrieval-CoT die Random-Methode um 3,3%.

Außerdem erreicht Falcon-40B-Instruct eine Genauigkeit von 37,3% bei GSM8K und 18,5% bei AQUA, während Falcon-7B-Instruct 5,4% bei GSM8K und 11,4% bei AQUA erreicht. Dies betont, dass größere Modelle in einer Few-Shot Situation besser abschneiden, aber im Vergleich zu GPT-3.5-turbo eine schlechtere Performance zeigen, mit Abweichungen von über 30%. Der Code ist verfügbar unter `https://github.com/Lori10/Master-Thesis-Few-Shot-CoT-Prompting-LLM`.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABKÜRZUNGSVERZEICHNIS

LLMs Large Language Models

DNN Deep Neural Networks

NLP Natural Langugage Processing

LLM Large Language Model

LoRA Low-Rank Adaptation of Large Language Models

QLoRA Quantized LoRA

ICL In-context learning

PEFT Parameter-Efficient-Fine Tuning

GPT Generative Pre-trained Transformer

IQR Interquartile Range

BERT Bidirectional Encoder Representations from Transformers

RLHF Reinforcement Learning from Human Feedback

LSTM Long Short-Term Memory

RNN Recurrent Neural Networks

1 INTRODUCTION

1.1 MOTIVATION

Large Language Models (LLMs) are deep learning models designed to understand and manipulate human language. They have achieved advanced performance in Natural Langugage Processing (NLP) tasks and significantly impacted artificial intelligence. LLMs are trained through a two-step process: pre-training and fine-tuning. Pre-training allows models to learn general language understanding capabilities and then adapt to specific tasks during fine-tuning. The LLM is fine-tuned on a smaller, task-specific labeled dataset, using supervised learning techniques to update the model's weights. This process is used for tasks like sentiment analysis, question-answering, and named entity recognition [49].

In NLP, the standard fine-tuning approach requires a large dataset and computational resources. To overcome these drawbacks, there are several options like Parameter-Efficient-Fine Tuning (PEFT) or few-shot prompting (also referred to as ICL). PEFT, which includes Low-Rank Adaptation of Large Language Models (LoRA), Quantized LoRA (QLoRA), is a long-standing paradigm in deep learning that aims to adapt large-scale pre-trained models to various downstream tasks by modifying as few parameters as possible. However, with PEFT we must optimize the model's parameter and need computational resources [27].

However, to avoid the entire process of optimizing and updating the model's weights few-shot prompting techniques were introduced. Few-shot prompting techniques have gained popularity due to their effectiveness in scenarios where labeled data is limited and annotation is time-consuming. These methods process a small number of labeled training examples and adapt pre-trained language models for specific tasks. Some LLMs, like Generative Pre-trained Transformer (GPT)-3, have demonstrated the ability to perform few-shot learning, enabling more efficient transfer learning and reducing the need for extensive fine-tuning [6].

1.2 OBJECTIVE

Few-shot prompting is a technique that leverages the capabilities of LLMs to perform specific tasks by providing a few examples and instructions related to a specific task. This approach has demonstrated promising results in improving LLM performance on complex tasks that require reasoning and step-by-step thinking. Given a dataset, there are several techniques for selecting in-context demonstrations like Random-CoT [63], Auto-CoT [66], Active-CoT [14], Retrieval-CoT [66]. Random-CoT is straightforward approach for randomly selecting demonstrations to prompt a LLM for unseen data. On the other hand, Retrieval-CoT involves the selection of semantically similar in-context examples for a given test example based on cosine similarity. [32] demonstrated that this retrieval-based prompt selection consistently outperforms the random baseline. However, when an LLM is used to generate answers, there is a risk of obtaining incorrect demonstrations (i.e., demonstrations with incorrect answers), which, in turn, in case of Retrieval-CoT, may lead the LLM to replicate similar mistakes when reasoning for the test question. This is known as "misleading by similarity" [66]. [66] demonstrated that "misleading by similarity" is a contributing factor to the inferior performance of Retrieval-CoT, and diversity can help mitigate this issue by increasing the likelihood of selecting correct demonstrations by clustering the questions and sampling them from each cluster. Active-CoT is a recent method that samples examples about which the LLM is most uncertain. As both diversity and uncertainty are proven to be useful and complementary for selecting the most informative questions, [14] consider the combination of them as an important future direction. Additionally, it's important to note that Auto-CoT and Active-CoT were evaluated differently in previous studies, with Auto-CoT using labels generated by LLM and Active-CoT utilizing human-generated labels. Moreover, open-source models typically require fine-tuning for achieving optimal performance on a downstream task [36]. However, they can also be utilized in a few-shot setting. In the landscape of advanced LLMs, GPT-4 stands out as one of the most capable models, showcasing superior performance compared to GPT-3.5-turbo in a wide variety of tasks. With this context in mind, these key research questions arise:

1. How can we effectively combine diversity and uncertainty to identify and select the most informative examples from a dataset?

2. How do various few-shot prompting methods perform on reasoning tasks compared to Random-CoT under different labeling scenarios, and how does the choice of labeling affect their few-shot performance?

3. Can few-shot prompting with GPT-3.5-turbo outperform GPT-4 in zero-shot settings for reasoning tasks?

4. How do open-source models such as Falcon perform on reasoning tasks when employed in few-shot prompting scenarios and how does the model size influence the performance?

1.3 STRUCTURE

This thesis is structured into five chapters, each serving a specific purpose in exploring LLMs and few-shot prompting. Below is a comprehensive overview of the content and objectives covered in each chapter. The first chapter provides motivation, objectives and structure of the thesis.

In the second chapter, the methodology section provides a historical context on the evolution of LLMs in NLP, incuding the pre-transformer and transformer area. Moreover, it provides a more detailled overview of how GPT-based LLMs are evaluated and improved over time, examining Transformer architecture and various LLMs, including Base and Instruction-Tuned LLMs, and highlighting their distinctions. Additionally, this section provides a brief overview of several transformer-based task-adaption methods, the advantages and limitations of ICL. Furthermore, it explores several prompting methods, covering Random-CoT, Auto-CoT, Active-CoT, Retrieval-CoT.

In the third chapter, the proposed prompting methods are introduced and thoroughly explained, providing detailed insights into their mechanisms.

The fourth chapter presents the experimental setup, covering aspects such as data, metrics, model choices, baselines, results, and comparisons with prior work. It offers comprehensive comparisons of model performance, including detailed evaluations of prompting methods under various scenarios, comparisons among different models, and additional analyses addressing the influence of uncertainty on few-shot performance.

The fifth chapter outlines potential improvements for future research and the key findings and contributions of the thesis are summarized, providing a concise overview of the research outcomes.

2 METHODOLOGY

2.1 HISTORY AND EVOLUTION OF LANGUAGE MODELS

Language models have developed over time due to advancements in NLP, machine learning, and computing resources. This section offers an overview of key milestones and breakthroughs in the evolution of LLMs.

2.1.1 *Pre-Transformer Era*

- **Eliza:** One of the earliest NLP programs, Eliza was a simple chatbot designed by Joseph Weizenbaum to mimic a psychotherapist. It used patterns to generate responses, laying the foundation for future conversational AI systems [64].
- **Statistical Language Models:** Statistical language models, like n-grams, predicted word probability based on preceding words. While used in speech recognition and translation, they struggled with understanding long-range dependencies in text [67].
- **Neural Language Models:** Neural language models, including feed-forward and Recurrent Neural Networks (RNN), emerged as an alternative to statistical model [67]. [5] introduced a feedforward neural network for language modeling, while [34] popularized RNN-based models.
- **LSTM Models:** Long Short-Term Memory (LSTM), introduced by [58], solved RNN's vanishing gradient problem. They were used in tasks like machine translation and served as the foundation for several LLMs.

2.1.2 *Transformer Era*

The transformer model is a revolutionary architecture in the field of deep learning. It differs from traditional models by completely forgoing recurrent and convolutional layers. Instead, it harnesses the power of attention mechanisms to compute contextualized embeddings, effectively addressing issues associated with RNN. These issues include sequential computation, which hinders parallelization and slows down training. In contrast, the transformer is highly parallelizable, making it significantly faster to train [61].

- **Transformer:** [61] introduced the transformer architecture, replacing recurrent layers with self-attention mechanisms. This innovation enabled more efficient and powerful LLMs, laying the foundation for models like GPT, Bidirectional Encoder Representations from Transformers (BERT), and T5.

- **GPT:** OpenAI released GPT, a unidirectional transformer model pre-trained on a vast text corpus. GPT displayed impressive language generation capabilities, marking the start of a new LLM era [49].
- **BERT:** Google introduced BERT, using a masked language model to enable bidirectional context representation. BERT achieved state-of-the-art performance, revolutionizing NLP [13].
- **BERT Variants:** NLP saw significant advancements with the introduction of ALBERT, RoBERTa, DistilBERT, XLNet, XLM-RoBERTa, and BART. These models introduced innovative techniques, such as lite versions of BERT, optimizing hyperparameters and data size, and enhancing bidirectional dependencies. ALBERT introduced a lite version of BERT with contextualized language representations, while RoBERTa improved BERT by optimizing hyperparameters and data size. DistilBERT made large models more efficient for edge devices, while XLNet achieved bidirectional dependencies by maximizing anticipated likelihood over permutations. BART is a generalization of BERT and GPT, using bidirectional encoding and left-to-right decoding [61].
- **GPT-2:** OpenAI released GPT-2, a more powerful version of GPT. GPT-2 excelled in text generation, producing coherent content with minimal prompting [50].
- **GPT-3:** GPT-3, a LLM with 175 billion parameters, was trained and tested for few-shot prompting without gradient updates or fine-tuning. It displayed strong performance in various NLP tasks but faced challenges in specific datasets [6].
- **Instruction-Tuned LLMs:** Instruction-Tuned LLMs are versatile and designed to execute various tasks based on instructions [46]. Some prominent Instruction-Tuned LLMs introduced in recent years include GPT-4, ChatGPT, Claude, Anthropic's next-generation AI assistant, ChatGLM etc. These models are designed for different use cases and constraints [67]. For instance, GPT-4, developed by OpenAI, delivers exceptional performance across a wide array of tasks [44], while ChatGPT focuses on text-based natural language conversations [3].

2.2 TRANSFORMER ARCHITECTURE

2.2.1 *Introduction*

RNN, particularly LSTM and gated RNN, are widely used in sequence modeling and translation problems like language modeling and machine translation. However, their sequential nature prevents parallelization within training examples, which is critical at longer sequence lengths. Recent work has improved computational efficiency through factorization tricks and conditional computation, but the fundamental constraint of sequential computation remains [61].

In contrast to RNNs, the Transformer is a model architecture that relies entirely on an attention mechanism to draw global dependencies between input and output, enabling parallelization. [61] reached state-of-the-art performance in translation task after training a Transformer for as little as twelve hours on eight P100 GPUs. Transformers are designed to process sequential input data like natural language, perform tasks like text summarization and translation, but process the entire input at once. The attention mechanism allows the model to focus on the most relevant parts of the input for each output, reducing training time and leading to the development of large pre-trained systems such as BERT and GPT. Introduced by the Google Brain team in 2017, Transformer architecture is increasingly becoming a model of choice for NLP problems, replacing RNN models such as LSTM [61].

2.2.2 *Model Architecture*

Transformer has an encoder-decoder structure, with Nx identical encoder layers and Nx identical decoder layers. The encoder's role, on the left side of the Transformer architecture, is to map an input sequence of symbol representations to a sequence of continuous representations. These representations are subsequently passed on to the decoder, which generates an output sequence of symbols one element at a time. The model is auto-regressive, consuming previously generated symbols as additional input. The Transformer architecture uses stacked self-attention and point-wise, fully connected layers for both the encoder and decoder components, avoiding recurrence and convolutions for output generation. It has an extra layer of positional encoding between the encoder and decoder stacks to take advantage of the order of the sequence. In Figure 2.1, the architecture of this model is depicted [61].

Encoder The Transformer encoder is a deep learning architecture consisting of Nx identical layers where each word in the input sequence is translated one after the other by the Nx layers of the encoder, each using its own weight and bias parameters. Each layer c with two sub-layers. The first sublayer generates self-attention through a multi-head mechanism, while the second sublayer has a position-wise fully connected feed-forward network. There is also a residual connection around each sublayer and a normalization layer, which normalizes the sum calculated between the sublayer input and output [61].

Decoder The decoder is also constructed with a set of Nx identical layers. While the encoder layers have two sublayers, each decoder layer incorporates one more sublayer, which takes the previous encoder output and conducts multi-head attention. Like the encoder, the three subcomponents on the decoder side are equipped with residual connec-

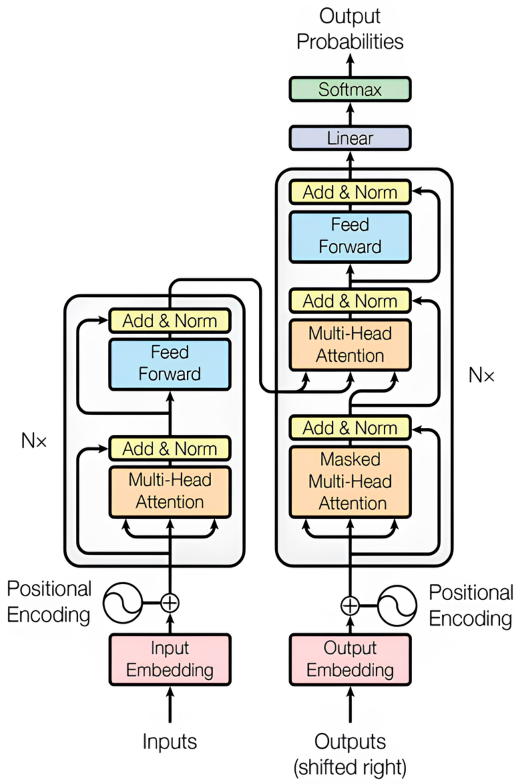

Figure 2.1: Transformer Architecture. Reprinted from [61]

tions that encircle them, and these are followed by a normalization layer. Multi-head attention is facilitated by introducing a masking technique which ensures that the predictions for a word at a given position depend only on known outputs from earlier words in the sequence [61].

Attention Mechanism The attention mechanism aimed to solve the issue of a bottleneck caused by using a fixed-length encoding vector. This bottleneck problem limits the decoder's access to the information in the input, which becomes especially problematic when working with long or complex sequences. In such cases, the dimensionality of the representation is constrained to be the same for short as well as for long sequences. Transformers use an attention mechanism to focus on relevant parts of the input sequence when processing each word. This mechanism assigns higher weight or importance to a certain element of the input for each element of the output. It is part of a network's architecture and manages and quantifies the interdependence between input and output elements. The self-attention mechanism helps identify relevant parts of the sequence by comparing positions using query, key, and value matrices where keys and values represent the input sequence and queries are vectors that represent the current element of the output sequence. It computes the similarity between the query and each key using a dot product or other similarity function. The resulting scores are normalized using a softmax function to produce the attention weights. The attention weights are then used to weight the values, which are summed to produce the context vector. The context vector is then used to generate the output. An example of scaled dot-product attention is shown in 2.2. This enables transformer models to consider context and generate more accurate outputs than traditional neural networks [61].

Embeddings The embedding layer in the architecture of a transformer is responsible for converting the input tokens into a continuous vector space representation. This layer is the first layer in the transformer model and is used to create word embeddings that capture the semantic meaning of words in a sentence. The embeddings layer maps input tokens to a sequence of vectors (word embeddings) via an embedding matrix which is learned during training [61].

Positional Encoding Transformer architecture doesn't capture information about the positions of words in a sequence. To address this, it makes use of positional encodings, which are vectors that match the dimension of the input embeddings and are created using sine and cosine functions with different frequencies. Afterward, they are simply combined with the input embeddings to generate information about the positions of words in the sequence [61].

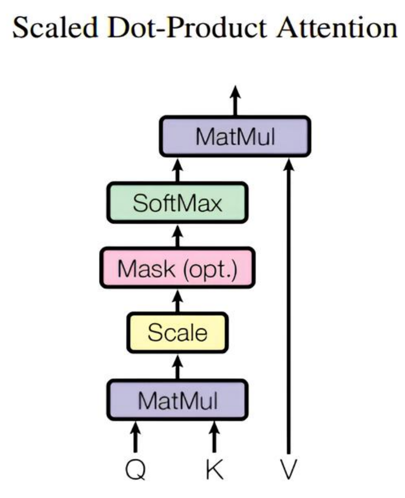

Figure 2.2: Scaled Dot-Product Attention. Reprinted from [61]

Let's explore a simple example of a forward pass in the Transformer architecture to better understand how information flows and how its components are connected. In this example, as depicted in Figure 2.3, I focus on a translation task, which was one of the main goals of designing the Transformer.

A French phrase will be translated into English. Initially, the input words will be tokenized using the same tokenizer utilized during training. The tokens will be fed into the encoder, pass through the embedding layer before it enters the encoder part. Then, after adding a positional encoding vector to embeddings, the resulting output goes through a multi-head self-attention layer. The outputs of the multi-headed attention layers are fed through a feed-forward network to the output of the encoder, which results in a deep representation of the input sequence. This representation is then integrated into the decoder to influence its self-attention mechanisms. Next, an "start-of-sequence" token is added to the decoder's input, initiating the prediction of the next token. This prediction relies on the contextual understanding provided by the encoder. The output from the decoder's self-attention layers is conveyed through the decoder's feed-forward network and concludes with a final softmax output layer, yielding the first token. This process repeats, as the output token is looped back into the input to prompt the generation of the next token, until the model produces the "end-of-sequence" token. Consequently, the final sequence of tokens can be transformed into words, resulting in the output. In this example, the output is, "I love machine learning." It's important to know that there are various methods for utilizing the output from the softmax layer to predict the next token, each affecting the creativity of the generated text [19].

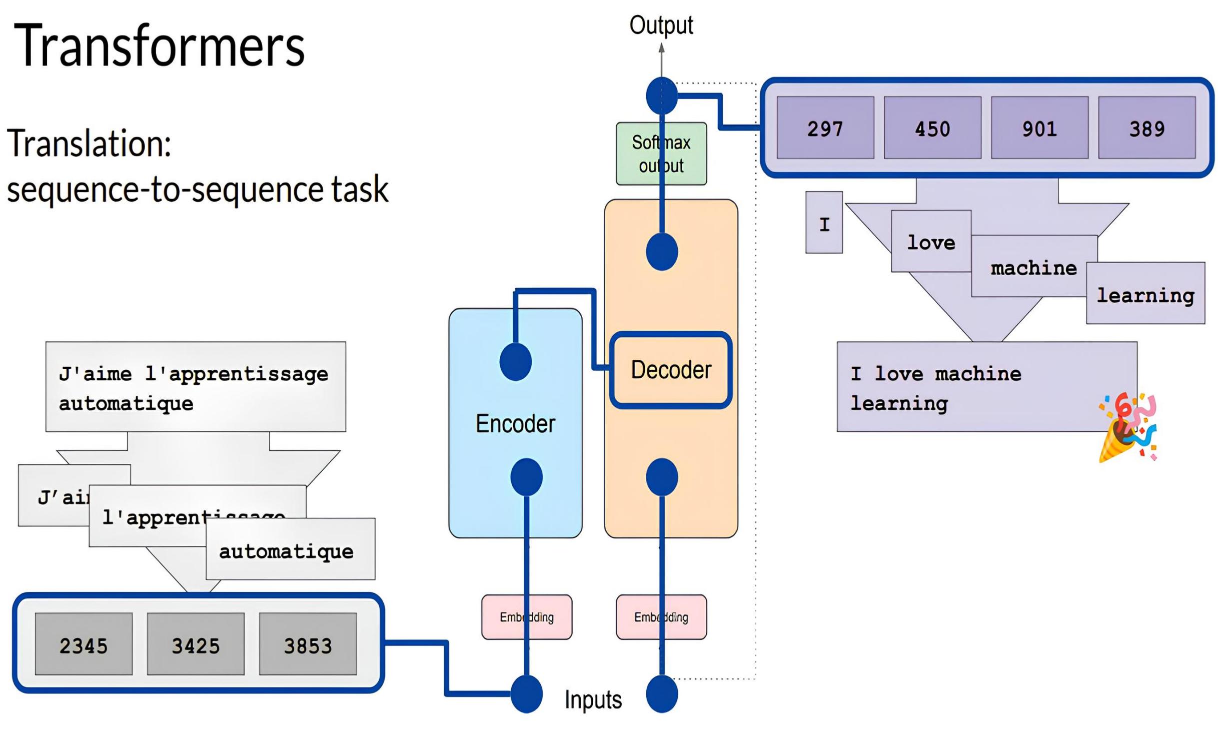

Figure 2.3: An illustration of the forward pass during a translation task within the Transformer architecture. Retrieved from [10]

2.3 EVOLUTION OF GPT-BASED LLMS

Before investigating the evolution GPT-based LLMs, it's imporant to gain an understanding of the different types of LLMs built upon the Transformer architecture. These LLMs can be broadly classified into three categories: encoder-only models, encoder-decoder models, and decoder-only models, as illustrated in Figure 2.4 [10, 29]. GPT, which will be explored in detail since models used on these work are gpt-based, belongs to the category of decoder-only models. Now, let's highlight some key differences between these LLM types to gain a better understanding of their distinct characteristics.

- **Encoder-decoder** Encoder-decoder models leverage both components of the Transformer architecture, the encoder and decoder. These models frequently integrate pre-training objectives or structural adjustments to improve their effectiveness across a range of NLP tasks. They perform well on sequence-to-sequence tasks such as translation, where the input sequence and the output sequence can be different lengths. They can also be scaled and trained to perform general text generation tasks. Examples of encoder-decoder models include BART and T5 [10, 29].

- **Encoder-only** Encoder-only models can operate as sequence-to-sequence models, but without further modifications, the input and output sequences are of equal length. Their usage is less common nowadays, but by introducing additional layers to the architecture, we can train encoder-only models to perform classification tasks, such as sentiment analysis. BERT stands as an example of an encoder-only model, which utilizes masked language and next-sentence pretraining objectives, to learn bidirectional contextual representations of input texts. These representations can then be used for fine-tuning for diverse downstream tasks such as sentiment analysis, question-answering, and named entity recognition [10, 29].

- **Decoder-only** Finally, decoder-only models utilize the decoder part of the transformer architecture, often with certain architectural adjustments. These models are pre-trained using language modeling objective and are referred to as autoregressive models. This autoregressive characteristic ensures that the model produces output tokens sequentially, utilizing previously generated tokens as context for generating the next token. In contrast to BERT, this leads to unidirectional representations. However, decoder-only models are some of the most commonly used today. Again, as they have scaled, their capabilities have grown. These models can now generalize to most tasks. Popular decoder-only models include the GPT family of models, BLOOM, Jurassic, LLaMA, and many more [10, 29].

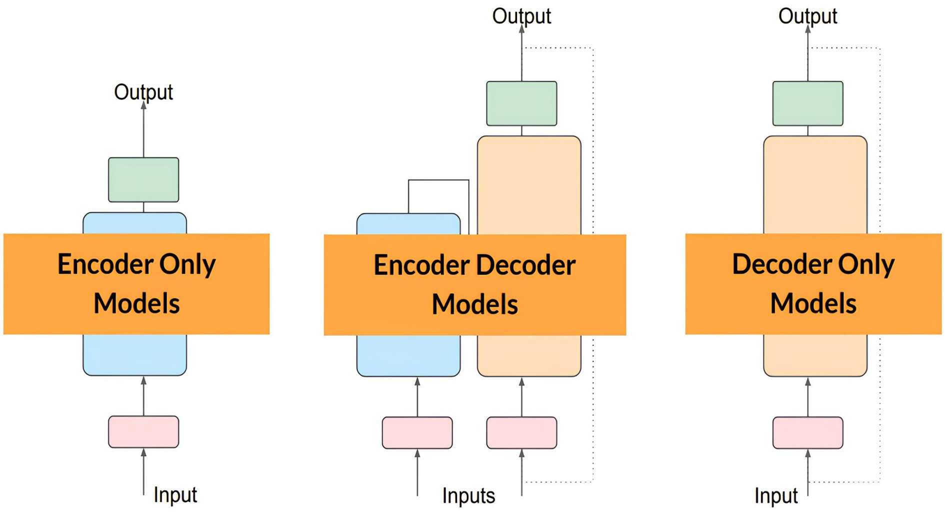

Figure 2.4: Illustration displaying three LLM architectures: Encoder-only, Decoder-only, and Encoder-Decoder. Retrieved from [10]

2.3.1 *Base LLMs*

In this section, I explore the evolution of decoder-only models, with a specific focus on GPT-based models. It begins with the introduction of GPT in 2018, marking the start of a new era in LLMs [49]. The path then leads us through following iterations, including GPT-2 [50] and GPT-3 [6], where advancements were made by training larger models on extensive datasets. GPT, GPT-2, and GPT-3 fall under the category of Base LLMs, as they are designed to predict the next word in a sequence. Additionally, I examine various other LLMs that have emerged, such as Falcon-40B-Instruct, Falcon-7B-Instruct, LLMs within the GPT-3.5 family, including models like code-davinci-002, InstructGPT, text-davinci-002, text-davinci-003, GPT-3.5-turbo and concluding with the release of GPT-4, the latest model from OpenAI. These mentioned models belong to the category of Instruction-Tuned LLMs. The primary focus lies on these types of models as some of them are used in my thesis.

Let's use a simple example to better understand the difference between of Base LLMs and Instruction-Tuned LLMs when generating text.

- **Base LLMs:** Base LLMs are trained to predict the next word based on text training data, often using a large amount of data from the internet and other sources to determine the most likely word to follow. For instance, if prompted with "Once upon a time there was a unicorn," it might complete the sentence with, "that lives in a magical forest with all unicorn friends." However, if asked, "What is the capital of France?" a base LLM might respond with various related questions, such as, "What is France's largest city?" or "What is France's population?" as internet articles often include lists of quiz questions about countries [38].

- **Instruction-Tuned LLMs:** On the other hand, Instruction-Tuned LLMs, where much of the recent momentum in LLMs research lies, are trained to follow instructions. So, if asked about the capital of France, an Instruction-Tuned LLM is more likely to provide a straightforward answer, like "The capital of France is Paris." These models are typically trained by starting with a base LLM trained on a large amount of text data, fine-tuning it using inputs and outputs that consist of instructions and corresponding attempts to follow those instructions. Further refinement is done through Reinforcement Learning from Human Feedback (RLHF) to enhance the system's ability to be helpful and follow instructions. Instruction-Tuned LLMs are designed to be helpful, truthful, and non-harmful. They are less likely to generate problematic or toxic text compared to base LLMs, which is why many practical usage scenarios are shifting towards Instruction-Tuned LLMs [38].

Now, let's explore deeper into Base LLMs.

2.3.1.1 *GPT*

NLP involves various tasks like textual entailment, question answering, semantic similarity assessment, and document classification. However, in many use-cases or domains, there is a lack of labeled data for training models for these tasks. [49] propose that by pre-training a language model on unlabeled text and fine-tuning it for each task, significant improvements can be achieved. They named this model GPT, built on the transformer architecture and leveraging the decoder component, as illustrated in Figure 2.5. Furthermore, task-aware input modifications are introduced during fine-tuning for effective knowledge transfer with minimal changes to the model's architecture [49].

Training Procedure

Training procedure of GPT is divided into two stages. In the first stage, a high-capacity language model is learned on a large corpus of text. This is followed by a fine-tuning stage, where the model is adapted to a discriminative task using labeled data [49].

Unsupervised pre-training: The goal is, with an unsupervised corpus represented as $T = \{t_1, \ldots, t_n\}$, to utilize a standard language modeling objective in order to maximize the following likelihood:

$$L_1(U) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{2.1}$$

where k represents the size of the context window, and the conditional probability P is estimated by a neural network with parameters denoted as Θ. These parameters are iteratively trained through the stochastic gradient descent method [52].

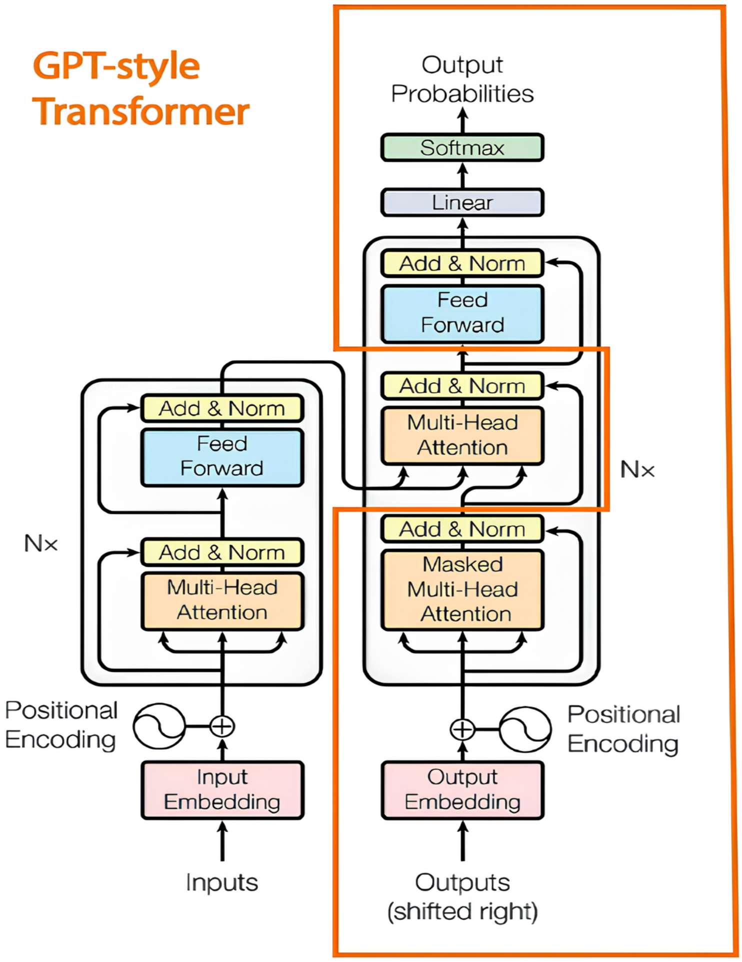

Figure 2.5: GPT-style Transformer. Reprinted from [59]

[49] used a multi-layer Transformer decoder for the language model, which applies multi-headed self-attention across the input context tokens, followed by position-wise feedforward layers, generating an output distribution over target tokens:

$$h_0 = U \cdot W_e + W_p \tag{2.2}$$

$$h_l = transformer\_block(h_{l-1}) \quad \forall\, i \in [1, n] \tag{2.3}$$

$$P(u) = softmax(h_n, W_e^T) \tag{2.4}$$

where U represents the context vector of tokens, with $U = (u_{i-k}, \ldots, u_{i-1})$. Here, n stands for the number of layers, W_e denotes the token embedding matrix, and W_p is the position embedding matrix [49].

The BooksCorpus dataset [69] is used for training the language model, including more than 7,000 unique, unpublished books spanning diverse genres, such as Adventure, Fantasy, and Romance. An essential feature of this dataset is the presence of extended, long text segments. This characteristic allows the generative model to effectively capture and utilize long-range contextual information. Exceptional performance is demonstrated by the language model, with a remarkably low token-level perplexity of 18.4 on this corpus [49].

Supervised fine-tuning: Following pre-training of the model, the model's parameters are adapted for the supervised target task. A labeled dataset, denoted as C, is assumed, where each instance consists of a sequence of input tokens, $x_1, \ldots, x_m$, along with a label y. These input sequences are then passed through the pre-trained model to obtain the final activation of the transformer block, denoted as h_l^m. Subsequently, h_l^m is fed into an additional linear output layer with parameters W_y to make predictions for y [49].

$$P(y|x_1, \ldots, x_m) = softmax(h_l^m \cdot W_y) \tag{2.5}$$

This leads to the following objective for maximization:

$$L_2(U) = \sum_i \log P(u_i|x_1, \ldots, x_m) \tag{2.6}$$

[49].

Task-specific input transformation and output layer adaptation

For some tasks, such as text classification, adding a linear and a softmax layer on top of the architecture is sufficient for fine-tuning. However, for certain others like question answering or textual entailment, which involve structured inputs, the pre-trained model requires adaptations. Rather than introducing task-specific architectures, [49] opt for a traversal-style approach. This approach involves converting structured inputs into ordered sequences that the pre-trained model can handle effectively [49]. In the case of textual entailment, the token sequences of the premise and hypothesis are concatenated with a delimiter token in between. For similarity tasks, where there is no inherent sentence ordering, the input sequence is modified to encompass both possible sentence orderings, and the results are processed independently, combined, and then fed into the output layer. In question answering and commonsense reasoning tasks are provided a context document, a question and a set of potential answers. To address these tasks, the context and question are concatenated with each possible answer, separated by a delimiter token, creating distinct sequences. Each sequence is independently processed by the model, and the outputs are normalized using a softmax layer to generate a distribution of possible answers. Based on the task, we add one or more Linear + Softmax layers, often referred to as output head, on top the architecture. These input transformations ensure that the model can adapt to structured inputs while minimizing significant architectural modifications across various tasks and output heads allow a transformer-based model to be used for different tasks [49].

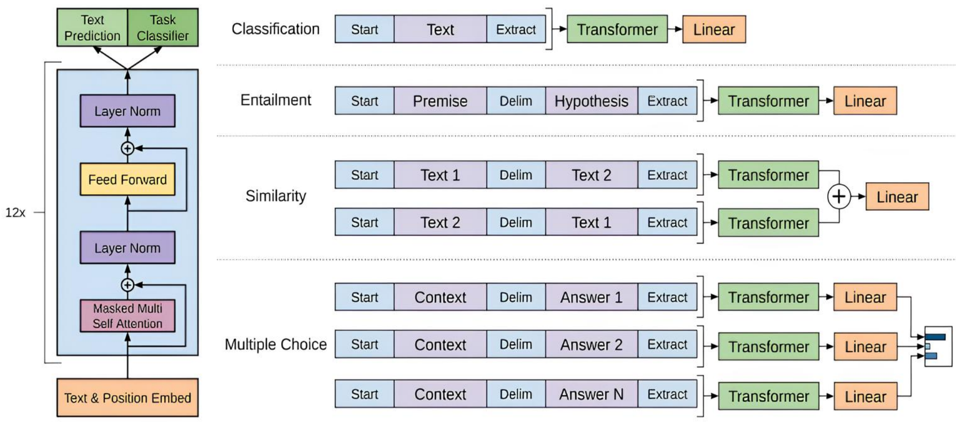

Figure 2.6: Task-specific input transformations. Reprinted from [49]

Performance

GPT consistently outperforms models that are individually crafted for specific tasks, resulting in significant improvements in 9 out of the 12 tasks. For instance, [49] achieved improvements, such as 8.9% in commonsense reasoning (Stories Cloze Test), 5.7% in question answering (RACE), and 1.5% in textual entailment (MultiNLI).

2.3.1.2 *GPT-2*

GPT-2 is a transformer-based LLM with 1.5 billion parameters. It was trained on a diverse dataset of 8 million web pages, which was carefully curated to ensure high document quality. The content for this dataset was sourced from the Internet, specifically from outbound links on Reddit with at least 3 karma. This is an indicator of whether other users found the link to be of interest, whether for educational or entertainment purposes, ensuring a dataset of superior quality compared to others, such as Common Crawl. The core objective of GPT-2's training, like GPT, is to predict the next word in a given text, utilizing all preceding words. The diverse nature of the dataset naturally exposes the model to a wide array of tasks across various domains [50].

GPT-2 represents a significant evolution from its predecessor, GPT, with over ten times the number of parameters and training data volume. It is based on the Transformer architecture as the foundation for many language models. The model is closely aligned with the specifications of the GPT model, introduced by [49], with a few adjustments. In GPT-2, layer normalization, following the approach of [2], has been relocated to the input of each sub-block. Additionally, an extra layer of normalization follows the final self-attention block. The vocabulary size is increased to 50,257 tokens because of the large dataset. Moreover, the context size is expanded from 512 to 1024 tokens, which improves GPT-2's ability to grasp how words relate in longer sentences, surpassing the capabilities of GPT [50].

GPT-2 showcases a broad spectrum of capabilities, excelling in generating high-quality synthetic text samples. Furthermore, it demonstrates superior performance compared to other language models designed for specific domains like Wikipedia, news, or books. The proficiency of this model extends to language tasks such as question answering, reading comprehension, summarization, and translation, and it can learn these tasks directly from raw text, without making any adjustments to the models and the need for task-specific training data. While the performance in these downstream tasks may not be at the state-of-the-art level, it emphasizes the potential of unsupervised techniques when provided with abundant unlabeled data and computational resources. GPT-2 is not trained using data specific to the downstream tasks; it is only tested on them in a final evaluation, a scenario called "zero-shot." In this "zero-shot" setup, GPT-2 outperforms models trained on domain-specific datasets like Wikipedia, news, or books when assessed on those same datasets [50].

2.3.1.3 *GPT-3*

GPT-3 is an autoregressive language model with 175 billion parameters. This model has ten times more parameters than any previously existing non-sparse language model. The performance of this model is assessed in the few-shot learning setting as it relies on tasks and few-shot demonstrations conveyed through text interactions with the model without any gradient updates or fine-tuning. GPT-3 excels in a wide range of NLP tasks, including translation, question-answering, and cloze exercises. Furthermore, it demonstrates exceptional performance in tasks that require dynamic reasoning and domain adaptation, such as word unscrambling, incorporating new words into sentences, and performing complex arithmetic operations. Although it has exhibited some initial potential, GPT-3 in few-shot setting still produces outcomes significantly inferior to those achieved through fine-tuning [6].

Increasing model size enhances the utilization of in-context information, resulting in better performance in few-shot settings, as demonstrated in [7]. Essentially, the larger the model, the better the LLM performs in these scenarios [6].

For GPT-3, the same model and architectural framework as GPT-2 [50] are used, incorporating the modified initialization, pre-normalization, and reversible tokenization techniques. The distinction lies in the utilization of alternating dense and locally banded sparse attention patterns within the transformer layers, which is similar to the Sparse Transformer [7] [6].

2.3.2 *Instruction-Tuned LLMs*

Instruction-Tuned LLMs are a category of LLMs fine-tuned using input-output pairs with clear instructions. They are designed to precisely follow instructions, making them suitable for practical purposes. These models start with a base LLM and then receive further fine-tuning with input-output pairs containing instructions and attempts to follow those instructions. To enhance the model's performance, they are often refined using RLHF, improving their ability to provide helpful, honest, and safe responses. Instruction-Tuned LLMs are less likely to generate problematic text and offer accurate and relevant answers [46].

2.3.2.1 *Falcon-40B-Instruct and Falcon-7B-Instruct*

Falcon is an open-source pre-trained model based on the transformer architecture. It was developed by the Technology Innovation Institute and is available in four versions: Falcon-40B for general purposes, Falcon-40B-Instruct tailored for chat applications, the smaller models including Falcon-7B, and Falcon-7B-Instruct. Falcon was trained to predict the next token using 1 trillion tokens of text. Out of these, 820 billion tokens were sourced from RefinedWeb [48], a carefully curated subset of CommonCrawl. The rest were gathered from books, code, academic papers, technical documents, and conversations on sites like Reddit and StackOverflow. Before Falcon, base models were mainly trained on small, carefully selected datasets and internet data that had been filtered for quality (either by considering link popularity or using a classifier trained on selected datasets). However, the lack of good training data often caused these models to generate incorrect responses, which is known as hallucination [1].

The architecture closely resembles GPT-3 [6] but incorporates distinctive features such as the FlashAttention algorithm and multiquery attention, which reduce memory requirements during inference. Falcon-40B-Instruct has surpassed Meta's LLaMA [60] to claim one of the top spots on the HuggingFace Open LLM Leaderboard [4], which is a leaderboard that aims to track, rank and evaluate open-source LLMs and chatbots [1].

2.3.2.2 *GPT-3.5*

The analysis conducted on GPT-3.5 models and GPT-4 is based on the information provided in the official website of OpenAI.

The OpenAI API is powered by a diverse set of models with different capabilities including gpt-3.5 series as one of the latest and most capable models, which consists of models trained on a mix of text and code data from before Q4 2021. These models can understand and generate natural language or code. GPT-3.5 series includes the following models [40]:

1. code-davinci-002: This serves as a base model and is well-suited for code completion tasks.
2. text-davinci-002: An InstructGPT model based on code-davinci-002.
3. text-davinci-003: An improved version of text-davinci-002.
4. gpt-3.5-turbo-0301: A further enhancement over text-davinci-003, specially optimized for chat applications.

To better understand the differences between GPT-3.5 models, I will examine and compare them, focusing on their performance, model size, pretrained data size, cost-effectiveness, and processing speed.

GPT-3.5 VS TEXT-DAVINCI-003/TEXT-DAVINCI-002

- The text-davinci-003 model is a powerful language model that excels in content creation, translation, and other complex tasks due to its ability to generate longer, coherent text. It has been trained on a large dataset, making it better suited for a wide range of tasks and has been further optimized with Reinforcement Learning from Human Feedback to follow instructions. This model is known for its zero-shot learning capability, enabling it to perform tasks without direct training. This is due to its large model size and extensive pre-training method, allowing the model to acquire knowledge and make extrapolations from a vast corpus of linguistic data [40].
- GPT-3.5, on the other hand, is designed to follow instructions and optimized for chat. It has a smaller model size which makes it faster than text-davinci-003 and it is more cost-effective [40].
- The text-davinci-002 model has similar capabilities to text-davinci-003 but it was trained with supervised fine-tuning instead of reinforcement learning [40].

In the next section I provide a detailed examination of InstructGPT, as it is the foundation model for Instruction-Tuned LLMs within the GPT-3.5 family.

2.3.2.3 *Instruct-GPT*

The OpenAI API relies on GPT-3 language models that can be guided to perform language tasks with well-crafted text prompts. However, these models can also produce outputs that contain false information, harmful content, or offensive language. One of the reasons for this is that GPT-3 is trained to predict the next word based on a large dataset of Internet text, rather than specifically focusing on executing language tasks in a safe and user-aligned manner. In simpler terms, these models may not always generate content that aligns with the user's intentions. To enhance the safety, usefulness, and alignment of the models, RLHF is utilized. For prompts submitted by customers to the OpenAI API, demonstrations of the desired model behavior are provided by their labelers, who also rank various outputs generated by the models. Then, this data is utilized for the fine-tuning of GPT-3. The resulting InstructGPT models demonstrates clear improvements in following instructions compared to GPT-3. Furthermore, they generate less false information and harmful content [46].

Training Procedure

In the training of InstructGPT models, RLHF is primarily used. This technique relies on human preferences as a reward signal to refine the model. Starting with a pre-trained GPT-3 [6], the model is guided by a set of prompts to generate the desired outputs, and a team of well-trained human labelers. Following this, a sequence of three steps is proceeded, as showcased in Figure 2.7 [46].

- **Step 1:** Demonstrations that illustrate the desired model behavior within the input prompt distribution are provided by the labelers. Then, a pre-trained GPT-3 model is fine-tuned using supervised learning with this data [46].
- **Step 2:** A dataset is created containing comparisons between different model-generated outputs, with labelers indicating their preference for a given prompt, which is sampled from the prompt distribution. A reward model is then trained to predict the output preferred by humans [46].
- **Step 3:** Given a new sampled prompt from the dataset, the policy generates an output, based on which the reward model computes a scalar reward. The supervised policy is further fine-tuned to optimize this reward using the PPO algorithm [56]. Steps 2 and 3 can be repeated continuously. Additional comparison data is gathered for the current best policy, which is then employed to train a new reward model and consequently, develop a new policy [46].

Performance

[46] assessed the ability of InstructGPT to follow user instructions by comparing its outputs to those of GPT-3. The results demonstrate a clear preference for InstructGPT models when it comes to prompts submitted to both

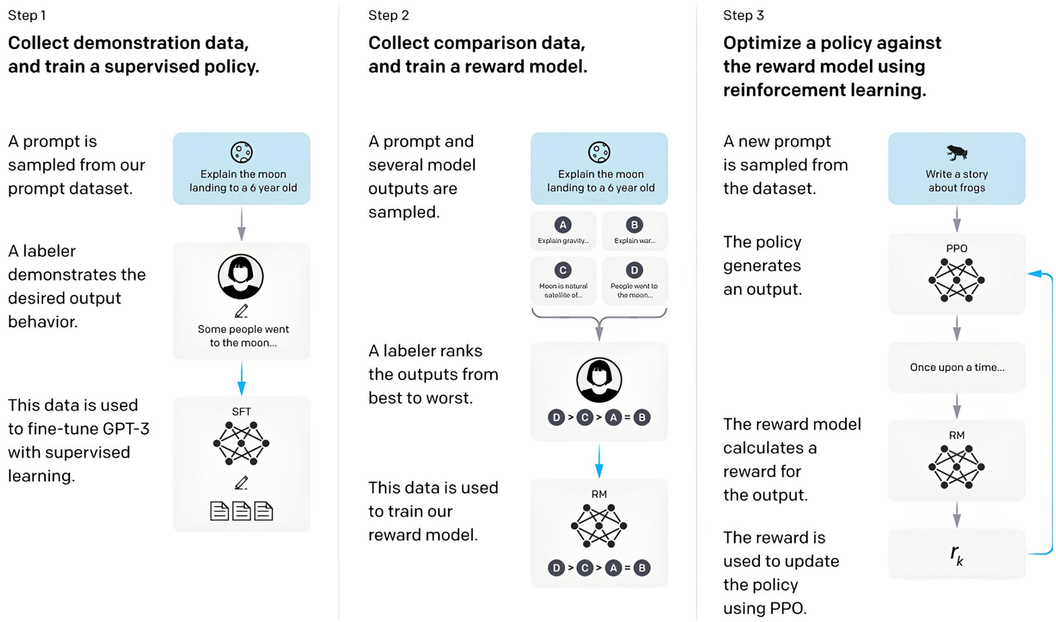

Figure 2.7: Reinforcement learning from Human Feedback. Reprinted from [46]

the InstructGPT and GPT-3 models via the API. This preference remains unchanged, even when they added a few-shot prompt to the GPT-3 to make it better at following instructions, and when compared to supervised fine-tuning of GPT-3. To assess the model's safety, [46] mainly relied on established metrics using publicly accessible datasets. In comparison to GPT-3, InstructGPT generates fewer false information (based on TruthfulQA17 [28]) and exhibits lower toxicity (based on RealToxicityPrompts18 [17]), when prompted to be respectful. InstructGPT, however, doesn't show a significant improvement over GPT-3 in generating biased content, as observed in the Winogender [54] and CrowSPairs [37] datasets.

The approach is designed to align models with the preferences of labelers, researchers, and customers, but alignment with broader preferences is not guaranteed. In experiments involving different labelers, it was found that InstructGPT performed similarly. Additionally, it was observed that reward models trained on one subset of labelers generalized well to another, suggesting that the models are not overfitted. Further research is needed to assess performance with a wider range of users and differing preferences [42].

Limitations

While InstructGPT models have made significant progress, they are not entirely aligned or safe. Efforts to improve safety include application reviews, content filters, and monitoring. Aligning models with user values, particularly those of specific groups, requires ongoing work to address potential biases and societal implications. Furthermore, InstructGPT can still make mistakes such as failing to follow instructions, making up facts, giving long answers to simple questions, or failing to detect instructions with false premises [42].

2.3.2.4 *ChatGPT*

OpenAI's most capable and cost effective model in the GPT-3.5 family is GPT-3.5-turbo, also known as ChatGPT. It has been optimized for chat using the Chat Completions API but works well for traditional completions tasks as well. This model was developed through RLHF. This method shares similarities with InstructGPT's approach but involves some variations in data collection. Initially, supervised fine-tuning were conducted where human AI trainers engaged in conversations while taking on both user and AI assistant roles. OpenAI provided these trainers with model-generated suggestions to help them compose responses. Afterward, this new dialogue dataset is combined with the InstructGPT dataset after adapting it into a dialogue format. Another distinction lies in the foundation models used in step 1: ChatGPT employs text-davinci-003, whereas InstructGPT starts with GPT-3 [41].

Limitations

- ChatGPT has the occasional issue of generating responses that sound plausible but are incorrect or nonsensical. Addressing this challenge proves difficult for several reasons. Firstly, during reinforcement learning training, there is no definitive source of truth to guide the model. Secondly, training the model to be excessively cautious leads to it declining questions it could answer correctly. Lastly, supervised training is misleading because the ideal answer depends on the model's knowledge rather than the human demonstrator's [41].
- The model's sensitivity to input phrasing is evident. For example, it might claim not to know the answer to a question presented one way but respond correctly to a slight rephrase of the prompt [41].
- ChatGPT often tends to be verbose and frequently employs specific phrases, like repeatedly stating that it's a language model created by OpenAI. These behaviors arise due to biases in the training data, where trainers prefer longer answers that appear more comprehensive, and are linked to known issues with over-optimization [41].
- Ideally, the model should seek clarifications when faced with ambiguous user queries. However, current models usually make guesses about the user's intent [41].

2.3.2.5 *GPT-4*

GPT-4 is a multimodal model capable of processing text inputs and generating outputs, with the potential to handle image inputs in the future. Its exceptional problem-solving capabilities surpass previous models due to expanded general knowledge and enhanced reasoning abilities. GPT-4 is optimized for chat-based interactions but also performs well in conventional completion tasks using the Chat completions API. GPT-4 includes several versions, like gpt-4, which has a context window length of 8192 tokens; gpt-4-0613, which includes function calling data; and gpt-4-32k, which has a context length of 32768 tokens.

Limitations

Although GPT-4 is among the most capable language models, it shares similar limitations with earlier versions like GPT-3.5-turbo. It's not entirely reliable, occasionally getting facts wrong and making errors in reasoning. However, GPT-4 has shown significant progress in reducing these hallucinations, achieving a 40% improvement in accuracy over GPT-3.5-turbo based on factuality tests conducted by OpenAI [45].

chatgpt vs gpt-4

- ChatGPT and GPT-4 are AI language models with varying model sizes and training data. GPT-4 has a larger model size, allowing it to handle complex tasks and generate more accurate responses. Its extensive training dataset provides a broader knowledge base and improved contextual understanding, making it a powerful tool for natural language understanding applications [45].
- However, GPT-4's advancements come at the cost of increased computational power requirements, making it less accessible to smaller organizations or individual developers. Additionally, higher resource demand leads to greater energy consumption during training, raising environmental concerns [45].
- On the other hand, GPT-4 is smarter than ChatGPT, capable of writing complex code, solving complex problems, and learning quickly. Both ChatGPT and GPT-4 address the challenge of bias, but GPT-4 seems less likely to give biased answers or provide factually inaccurate responses. GPT-4 is slower to respond and generate text at this early stage due to its larger size and higher processing requirements and costs [45].

2.4 in-context learning

LLMs based on transformer architecture, such as GPT, T5, and BERT, have achieved advanced results in NLP tasks and are now being applied to other domains like Computer Vision and Audio. The conventional approach involves large-scale pretraining on generic web-scale data followed by fine-tuning to downstream tasks. However, as models grow larger, regular fine-tuning becomes infeasible to train on consumer hardware and storing and deploying fine-tuned models independently becomes expensive. Several approaches adress these issues including LoRA, QLoRA and ICL. In the next section I am going to mention some key aspects of several transformer-based task-adaptation methods in NLP like ICL, QLoRA, LoRA and regular fine-tuning.

2.4.1 *Transformer-based Task-Adaptation Methods*

Transformer-based task-adaptation methods can be broadly categorized into two main approaches: fine-tuning methods and training-free methods. Fine-tuning adapts pre-trained LLMs for specific tasks by optimizing their weights through training on task-specific data, enabling the generation of task-related predictions. In contrast, training-free methods such as ICL use pre-trained models without additional parameter updates, emphasizing computational efficiency [15].

Fine-tuning approaches

Fine-tuning is a strategy that customizes pre-trained LLMs for specific downstream tasks in NLP. It is the process of retraining pre-trained LLMs for a particular task by adding a task-specific head on top of the LLM and further optimizing the pre-trained weights with respect to the task or by adding additional layer on top of the existing model with task-specific data. The resulting fine-tuned model generates predictions given new data that is related to the task. Beyond regular fine-tuning which involves training the entire model, researcher have developed alternative like LoRA and QLoRA methods to enhance the efficiency during training and reduce the memory requirements especially for large models [11, 21].

Regular fine-tuning: This is one of the most common approachs for adapting LLMs to a specific task. In this approach, the LLM is first pre-trained on a large corpus of text and then fine-tuned on a smaller dataset of labeled examples for the specific task. The fine-tuning process involves updating the weights of the LLM using backpropagation and gradient descent. Regular fine-tuning requires a large amount of labeled data for the specific task, which can be a limitation. However, it has been shown to be effective in improving the performance of LLMs on a wide range of tasks. Moreover, the need of huge computational resources for training LLM which consists of millions or billions of parameters makes it impractical for using the regular fine-tuning method [33].

LoRA: LoRA is a technique that enables efficient fine-tuning of LLMs while consuming less memory. It introduces pairs of rank-decomposition weight matrices, known as update matrices, to the existing weights of the model. Instead of training all model's weights for each task, LoRA only trains these newly added weights while freezing the pre-trained model, reducing the number of trainable parameters for downstream tasks. This plays a crucial role in enabling the model to adjust to various tasks by avoiding the optimization of all parameters for each task. This method has been applied to fine-tune more than 1000 models, demonstrating state-of-the-art results even with smaller models [21].

QLoRA: QLoRA, which is an extension of the LoRA technique, focuses on enhancing efficiency of LoRA by quantizing the weight values of the original network and therefore reducing the memory requirements. It backpropagates gradients throught a frozen, 4-bit quantized pretrained LLM. This approach incorporates several strategies to reduce memory including the utilization of a 4-bit data type called NormalFloat. 4-bit quantization is a method that reduces the size of LLMs to make them compatible with less powerful hardware by reducing their disk storage requirements [12]. This technique achieves this by representing the model's weights and activations using fewer bits instead of using 16-bit or 32-bit quantization, which is the default data type for representing weights when training or doing inference with a LLM. Furthermore, QLoRA applies double quantization, which quantizes the quantization weights and activations to further enhance memory efficiency. Additionally, it implements paged optimizer, which transfer data between the CPU and GPU to avoid out-of-memory issue on the GPU. The most successful model produced through the QLoRA approach, named Guanaco, surpasses all previous models in terms of performance on the Vicuna benchmark. Guanaco achieves an impressive 99.3% of ChatGPT's performance with only 24 hours of fine-tuning conducted on a single GPU [11].

Training-free approaches

Unlike fine-tuning methods, training-free methods does not conduct parameter update or additional training and directly makes use of pre-trained LLMs to make predictions. One quite popular training-free approach for adjusting LLMs for a downstream task is ICL [15].

ICL: In recent years, the field of NLP has witnessed a growing interest in a novel approach known as ICL. ICL has emerged as a promising paradigm for various NLP tasks, including sentiment analysis, paraphrase detection, natural language inference, and open-domain question answering. Within ICL, LLMs operate by making predictions solely based on contexts enriched with a limited number of input-output examples. A small set of examples is utilized to construct a context that serves as task demonstration, and this context is combined with a query or a question to create a prompt, as shown in Figure 2.8. Based on the provided prompt, the LLM generates an answer by extracting insights from the given examples [15].

2.4.2 *Advantages and Limitations of ICL*

2.4.2.1 *Advantages of ICL*

ICL offers several advantages over fine-tuning methods such as regular fine-tuning, LoRA, and QLoRA. Here are some key advantages of ICL.

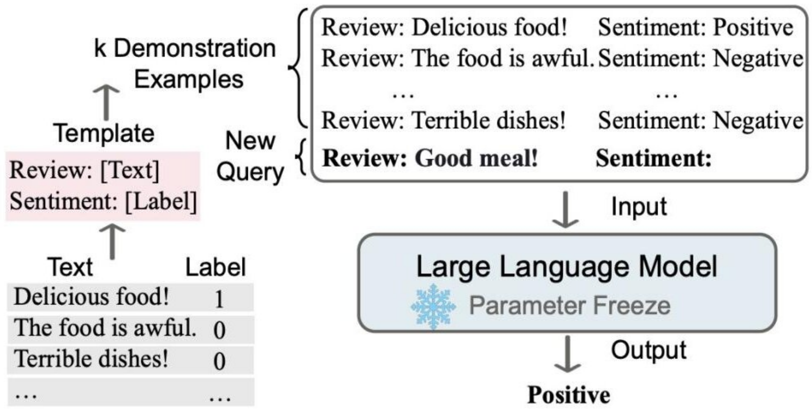

Figure 2.8: Example of ICL with text classification. Reprinted from [15]

1. **Data Efficiency:** ICL leverages LLMs to make predictions based on augmented contexts, requiring only a few examples. Some domains require a large amount of labeled data which can quite often be challenging. ICL is especially in these cases much more efficient than regular fine-tuning or LoRA, which often requires a relatively large amount of labeled data for training [6].

2. **Training-Free:** In contrast to fine-tuning methods, ICL does not perform any parameter optimization but instead it leverages pre-trained LLMs with given context. This makes this approach much faster and smoother [15].

3. **Low Complexity:** Unlike LoRA and QLoRA, which aim to reduce the number of trainable parameters or memory footprint by adding new layers to the pre-trained LLM, ICL does not require additional modifications to the model architecture. It can leverage existing LLMs without introducing additional complexity [11, 15, 21].

2.4.2.2 *Limitations of ICL*

However, it's important to acknowledge that ICL may not be effective in every situation. Here are some drawbacks and limitations of this approach.

1. **Limited performance:** The main drawback of ICL is that results obtained throught this approach have significantly underperformed state-of-the-art fine-tuned models which are trained on extensive datasets [6].

2. **Prompt engineering:** Prompt engineering is the process of designing the prompt for a LLM which consists of some instructions and examples related to the task. It is a time-consuming and often requires domain knowledge. Furthermore, the performance of ICL depends on several factors including the choice, the order, and number of examples in the

demonstration set, which makes it challenging to find the optimal set of demonstrations [15].

3. **Performance increase with scale:** Researchers have demonstrated that LLMs become more performant and capable at maximizing the utilization of contextual information as they increase in size, as illustrated in Figure 2.9 [6]. As a result, it requires a significant amount of compute hardware to use these models on inference. However, several innovative methods have been developed to run distributed inference like Text Generation Inference from HuggingFace [22] or vLLM [25].

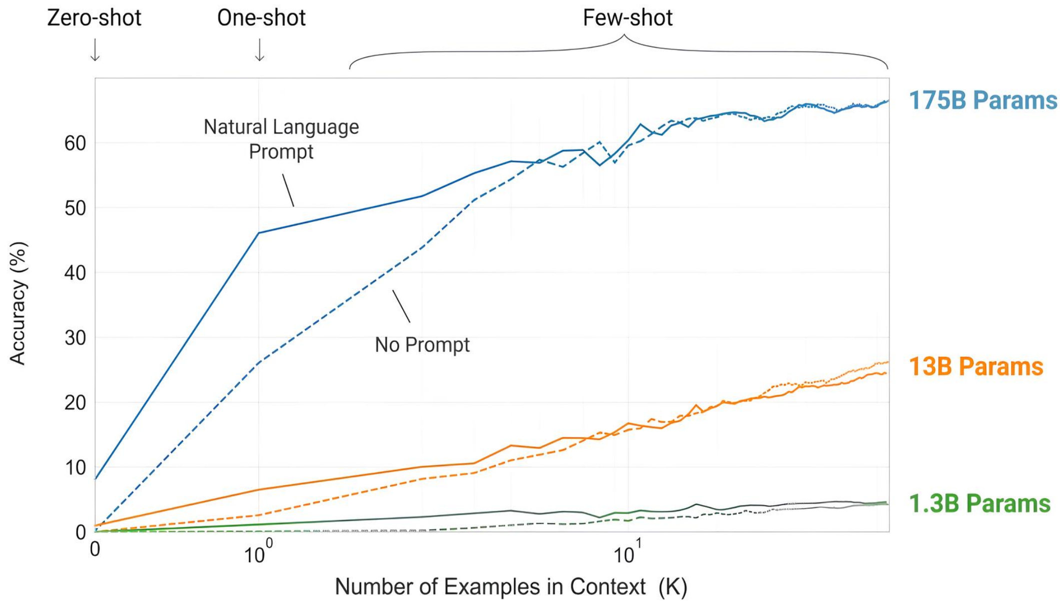

Figure 2.9: As model size increases, its performance in few-shot scenarios improves. Reprinted from [6]

2.4.3 *Goal Formulation*

The ICL situation in GPT-3 can be modeled as a problem of conditional text generation. To clarify, the likelihood of producing a specific target y is dependent on both the context C, which encompasses k in-context examples, and the input x. As a result, the conditional probability can be formulated as follows:

$$p_{\mathrm{LM}}(y|C,x) = \prod_{t=1}^{T} p(y_t|C,x,y_{<t}) \tag{2.7}$$

where LM refers to the language model's parameters and C represents a context string that combines k training instances using the special character "\n". To mitigate the computational requirements of fine-tuning, LLMs are utilized in an ICL approach, as explained earlier. Unfortunately, it has been demonstrated that GPT-3's performance tends to vary significantly with different in-context examples [32].

The objective of this thesis is to address this issue by selecting the most relevant and informative examples that enhace the few-shot performance. In this study, given a set of questions, I investigate two primary scenarios:

1. Using human-generated labels as ground truth answers to the questions.
2. Using labels generated through instructing GPT-3.5-turbo to answer questions using Zero-Shot-CoT. The motivation for including LLM-generated labels is that, in many domains and use-cases, manually labeling data demands high effort and expenses.

Given N labeled training data $D_{\text{train}} = \{(d_i = (x_i, r_i, y_i))_{i=1}^{N}\}$, and M labeled test data $D_{\text{test}} = \{(d_i = (x_i, r_i, y_i))_{i=1}^{M}\}$, where x_i, r_i, and y_i represent the question, reasoning chain, and answer, respectively, and k denotes the desired number of in-context examples, the objective is to select the most relevant examples from the training data. The demonstration set is constructed as follows: $C = [d_{\sigma(1)}, \ldots, d_{\sigma(k)}]$, where $\sigma(1), \ldots, \sigma(k)$ are the indices of the selected demonstrations. These chosen in-context examples, denoted as C, along with the test question x_{test} and the task instructions I, which collectively form the prompt, are fed to the LLM (LLM_θ). The LLM (LLM_θ) generates an answer in response as follows: $\hat{y}_{\text{test}} = LLM_\theta([I, C; x_{\text{test}}])$. This process is repeated for each x_{test} from D_{test}, and the predicted answers $\hat{y}$ are compared with the ground truth labels y. The accuracy is then computed as the evaluation metric.

2.4.4 *Existing Prompting Methods*

LLMs have revolutionized NLP by providing state-of-the-art performance on various NLP tasks. Some of the most common NLP tasks include text classification, question answering, and text generation. However, LLMs can also be leveraged for more complex tasks that require reasoning such as symbolic, arithmetic and commonsense tasks. This work focuses on arithmetic reasoning tasks which involves solving various mathematical problems such as addition, subtraction, multiplication, division, algebraic word problems etc. Few-shot prompting is a prompting technique used to interact with LLMs in NLP by providing a prompt which consists of instructions and a few examples related to a specific task, based on which a LLM generates a response. Few-shot prompting can be categorized into two major paradigm: CoT prompting and standard prompting [63].

2.4.4.1 *Few-shot chain-of-thought prompting vs few-shot standard prompting*

Standard prompting is the standard way of prompting a LLM which involves manually designing a few example that demonstrate a specific task or selecting them from a given labeled dataset. The demonstration set includes the question and the final answer for each example. In contrast to this method, chain-of-thought prompting is a few-shot prompting method in NLP that involves generating a chain of thought, which is a series of intermediate reasoning steps which leads to the final answer. Every demonstration is made up of a query along with a reasoning chain, in which the reasoning chain is a rationale (a series of intermediate reasoning steps) and a final answer. The goal of chain-of-thought prompting is to improve the ability of LLMs to perform better at complex reasoning tasks by breaking down the main question into multi-step sub-questions. 2.10 illustrates the difference between these two prompting methods. Recent developments have demonstrated that LLMs are much more capable at solving reasoning tasks when the reasoning chain is included in the prompt especially in complex cases with many reasoning steps and with LLMs that have more than 100B parameters [63].

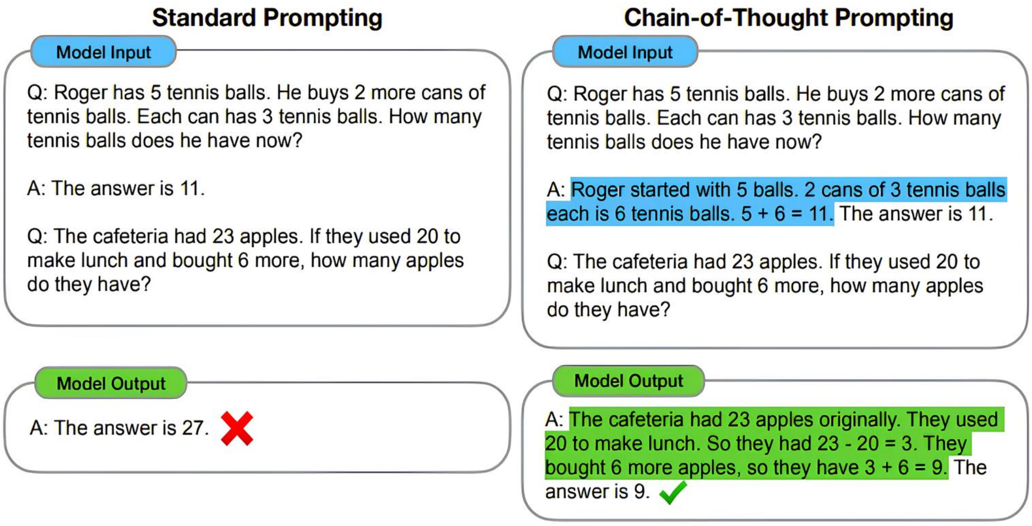

Figure 2.10: Example of standard and chain-of-thought prompting. Reprinted from [63]

2.4.4.2 *Few-shot chain-of-thought prompting strategies*

Investigations by [31, 32] have revealed that the effectiveness of ICL can depend significantly on the choice of in-context demonstrations. Furthermore, the formatting of the prompt, such as wording or order of demonstrations plays an import role [68]. A recent study [35] even questioned the necessity of having precise input-output mappings, indicating that using incorrect labels in examples only marginally drops the performance. This work focuses on the selection of the most relevant in-context examples by investigating several methods and evaluating their performance. There are several few-shot chain-of-thought prompting methods including Random-CoT [63], Auto-CoT [66], Active-CoT [14], Retrieval-CoT [66]. These methods have shown promising results in improving LLM performance on complex reasoning tasks. Next, I explore how these methods function and introduce newly proposed approaches, providing detailed explanations through pseudocode.

2.4.4.3 *Embedding Generation and Uncertainty Estimation*

Various prompting methods involve the processes of generating question embeddings and evaluating model uncertainty. Thus, I initially introduce two simple essential algorithms: Embedding Generation responsible for creating question embeddings, and Uncertainty Estimation which computes the model's uncertainty concerning the questions. To prevent redundant pseudocode, these algorithms are implemented just once, allowing their use across various prompting methods.

Embedding Generation

The Embedding Generation algorithm, a foundational procedure used to convert a set of questions into their respective embeddings. This process is achieved through the utilization of a dedicated sentence encoder.

Algorithm 1 Embedding Generation

Require: Set of training questions $Q = \{x_i \in D_{\text{train}}\}$, sentence encoder SE_θ
Ensure: List of embeddings E

```
procedure GENERATE_EMBEDDING(Q, SE_θ)
    Create an empty list E to store embeddings
    for each question x_i in Q do
        Add e_i = SE_θ(x_i) to E
    end for
    return E
end procedure
```

EMBEDDING GENERATION ALGORITHM

1. The process starts by initializing an empty list E for storing the embeddings.

2. Iteratively, the algorithm processes each question x_i within the input set Q. For each question encountered, the sentence encoder SE_θ, which in this thesis is the OpenAIEmbedding model from OpenAI, takes as input the question and generates its embedding e_i, which is added to E. The Auto-CoT method from [66], instead, uses the Sentence-BERT model introduced in [51].

3. The algorithm returns the question embeddings E which can be used for further computations.

The pseudocode of this method can be found in Algorithm 1.

Uncertainty Estimation

The Uncertainty Estimation algorithm involves assessing uncertainty of a LLM associated with a given question.

Algorithm 2 Uncertainty Estimation

Require: Set of training questions $Q = \{x_i \in D_{\text{train}}\}$, LLM LLM_θ, number of trails tr, temperature te, uncertainty metric function UM, in-context demonstrations AC
Ensure: List of uncertainty values U

procedure ESTIMATE_UNCERTAINTY($Q, LLM_\theta, tr, te, UM, AC$)
    Create an empty list U to store uncertainty values
    for each question x_i in Q **do**
        Create an empty list $\hat{Y}_i$ to store predictions for x_i
        for $j = 1$ to tr **do**
            Compute prediction $\hat{y}_i^{(j)}$ using LLM_θ with $[AC, x_i; te]$ as input: $\hat{y}_i^{(j)} = LLM_\theta(AC, x_i; te)$
            Add $\hat{y}_i^{(j)}$ to $\hat{Y}_i$
        end for
        Compute uncertainty u_i based on predictions in $\hat{Y}_i$: $u_i = \text{UM}(\hat{Y}_i)$.
        Add u_i to U
    end for
    return U
end procedure

UNCERTAINTY ESTIMATION ALGORITHM

1. The process is initiated by initializing an empty list U dedicated to storing the estimated uncertainty values.

2. The algorithm then proceeds to iterate through each question x_i in the provided set Q. For each question x_i, the following steps are executed:

   a) The LLM LLM_θ is invoked to generate tr potential answers by using temperature $tr > 0$ which ensures slightly randomness in the answer generation. The answers are aggregated into a set of predictions $\hat{Y}_i$ representing the potential outcomes.

   b) The uncertainty value u_i for the question x_i is calculated using the uncertainty metric function UM, which takes as input the aggregated potential answers $\hat{Y}_i$ and assesses the level of uncertainty within the predictions. The computed uncertainty value u_i is added to the list of uncertainty values U.

3. The outcome of the algorithm is the list of uncertainty values U, which provides valuable insights into the uncertainty associated with the questions.

The pseudocode of this method can be found in Algorithm 2.

Input parameters

A comprehensive list of input parameters shared across all prompting methods, which will be explained in the next sections, is presented below.

D_{train}	Training set containing questions and answers.
k	Number of in-context demonstrations to select.
SE_θ	Sentence encoder model for text-to-embedding conversion, specifically applied to questions.
LLM_θ	LLM utilized for inference and uncertainty estimation.
seed	random seed used to ensure reproducability when using random-based approaches.
tr	Count of queries made to "LLM_θ" for estimating question uncertainty.
te	Temperature of LLM_θ, introducing slight randomness during multiple queries to estimate question uncertainty ($te > 0$).
UM	Uncertainty metric function, used for uncertainty computation (e.g., entropy, variance, disagreement, self-confidence).
AC	In-context examples for querying a LLM for uncertainty estimation; can also be empty for Zero-Shot scenarios.
β	Weight for uncertainty in the F1-score metric.
top_p	Percentage for selecting the top top_p% F1-scores.
greedy	Flag for deterministic selection based on highest F1-score or weighted-random sampling using probabilities.
p	the number of demonstrations to select for the first phase, which is used by Diverse-Active-KMeansPlusPlus-Retrieval-CoT prompting method.

2.4.4.4 *Zero-Shot-CoT*

A LLM does not need examples to perform a certain task. By just providing the task instruction and "Let's think step by step", we can elicit LLMs to generate the reasoning chain that leads to the final answer, as ilustrated in 2.11. This method is called Zero-Shot-CoT. [23] have demonstrated that LLMs are decent zero-shot reasoners, as they perform well on several benchmarks.

2.4.4.5 *Random-CoT*

Random-CoT involves randomly selecting a few examples and their corresponding chains of thought to create the prompt for the LLM.

2.4.4.6 *Retrieval-CoT*

[32] proposed kNN in-context selection method which involves the selection of k semantically similar in-context examples of a test example based on cosine similarity. Those selected examples and the test question constitute the prompt which is fed to a LLM for generating an answer. Intuitively, the

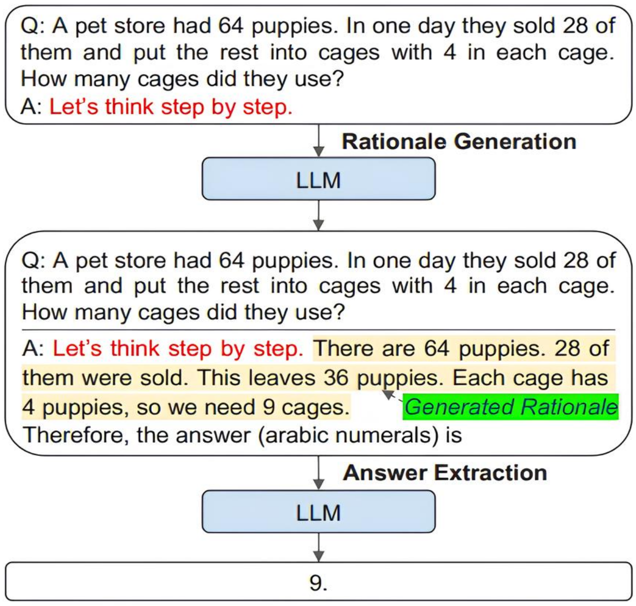

Figure 2.11: Zero-Shot-CoT. Reprinted from [63]

examples chosen using this strategy are likely to provide more valuable input that can fully leverage LLM's text generation capabilities. [32] assess the proposed method's performance on various tasks: sentiment classification, table-to-text generation, and question answering. Across these benchmarks, selecting prompts based on retrieval consistently outperforms the random selection baseline in terms of performance. Notably, this approach wasn't evaluated on reasoning tasks. However, [66] did assess this method on reasoning tasks, referring to it as Retrieval-Q-CoT. I maintain a consistent naming template for various methods and use the term Retrieval-CoT.

Algorithm 3 Retrieval-CoT

Input: Training set D_{train}, LLM LLM_θ, sentence encoder SE_θ, number of in-context demonstrations k, test question x_{test}
Output: Demonstration list C
procedure RETRIEVAL_COT($D_{\text{train}}, LLM_\theta, SE_\theta, k, x_{\text{test}}$)
  $Q = \{x_i \in D_{\text{train}}\}$
  $e_{\text{test}} = SE_\theta(x_{\text{test}})$
  for each $x_i \in Q$ **do**
    $e_i = SE_\theta(x_i)$
    $s_i = \frac{e_{\text{test}} \cdot e_i}{\|e_{\text{test}}\|_2 \|e_i\|_2}$
    Add s_i to S
  end for
  Get largest k similarities from S in descending order
  with indices $\{\sigma(1), \ldots, \sigma(k)\}$
  $C = [(x_{\sigma(1)}, r_{\sigma(1)}, y_{\sigma(1)}), \ldots, (x_{\sigma(k)}, r_{\sigma(k)}, y_{\sigma(k)})]$
  return C
end procedure

RETRIEVAL-COT ALGORITHM

1. Given test question x_{test} from test set D_{test}, the sentence embedding is generated using a sentence encoder model SE_θ, which takes the test question x_{test} as input and produces a vector representation as follows: $e_{\text{test}} = SE_\theta(x_{\text{test}})$.

2. For each question x_i from training set D_{train}, the sentence embedding $e_i = SE_\theta(x_i)$ is generated, and the cosine similarity with the test question's embedding $s_i = \frac{x_{\text{test}} \cdot e_i}{\|x_{\text{test}}\|_2 \|e_i\|_2}$ is computed.

3. The largest k similarities s_i's are selected in descending order, with indices $\{\sigma(1), \ldots, \sigma(k)\}$.

4. Finally, the algorithm returns the demonstration list $C = [(x_{\sigma(1)}, r_{\sigma(1)}, y_{\sigma(1)}), \ldots, (x_{\sigma(k)}, r_{\sigma(k)}, y_{\sigma(k)})]$, which is constructed by the selected questions and their corresponding answers.

5. The selected in-context examples and the test question are passed to the LLM LLM_θ, which generates an answer $\hat{y}_{\text{test}} = LLM_\theta([C; x_{\text{test}}])$

The pseudocode of this method can be found in Algorithm 3. Figure 2.12 illustrates a visual example.

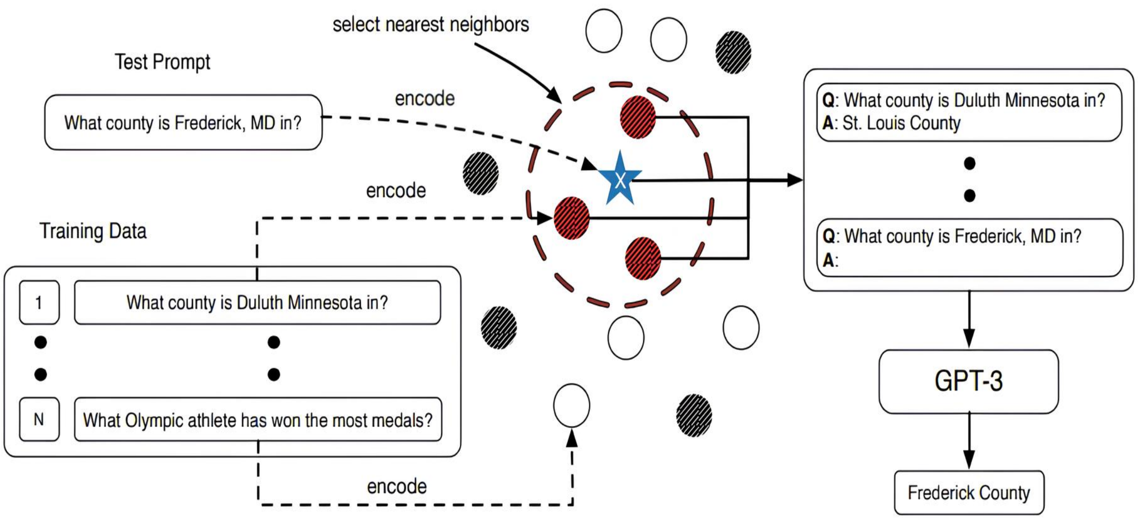

Figure 2.12: White dots: unused training samples; grey dots: randomly sampled training samples; red dots: training samples selected by the k-nearest neighbors algorithm. Reprinted from [35]

Impact of distance-based selection on GPT-3 few-shot performance

[32] have investigated how the distance between the in-context examples influences the performance of GPT-3, by conducting a comparison on the Natural Questions [24] dataset, focusing on two distance-based selection methods. For the evaluation, 100 test questions are randomly sampled and

the average Exact Math scores is reported. Given a test example, the first approach select the ten farthest training instances as in-context examples, while the second method utilizies the ten nearest neighbors. As shown in 2.13, it is evident that using the nearest neighbors as in-context examples leads to significanlty improved results compared to using farthest ones.

Method	Closest	Farthest
Accuracy	**46.0**	31.0

Figure 2.13: Exact Math score on the ten nearest and farthest neighbors within a subset of 100 test samples from the Natural Questions dataset. Reprinted from [32]

Retrieval-CoT May Fail due to Misleading by Similarity

In Retrieval-CoT, as [66] used Zero-Shot-CoT with a LLM for generating labels, there is a risk of obtaining wrong demonstrations (i.e., demonstrations with incorrect answers). Intuitively, after similar questions to a test question are retrieved, wrong demonstrations caused by Zero-Shot-CoT may mislead the same LLM to reason similarly with a wrong answer (e.g., replicating mistakes) for the test question This is known as "misleading by similarity". [66] demonstrated that "misleading by similarity" is a factor contributing to the inferior performance of Retrieval-CoT based on the results of the following experiment.

[66] applied Zero-Shot-CoT to all 600 questions in the MultiArith dataset. From these, 128 questions (denoted as Q) are identified where Zero-Shot-CoT gives incorrect answers, accounting for an error rate of 21.3% (128 out of 600). Among the Q questions where Zero-Shot-CoT fails, those where Retrieval-CoT or Random-CoT also fail as unresolved questions are considered. To calculate the unresolved rate, the number of unresolved questions is divided by 128 (the number of questions in Q). A higher unresolved rate suggests a method is more likely to make mistakes like Zero-Shot-CoT. The unresolved rate of Retrieval-CoT (46.9%) is considerably higher than that of Random-CoT (25.8%), as depicted in Figure 2.14. This indicates that when similar questions are used for test questions, Retrieval-CoT is negatively influenced by being misled by similarity [66].

Inspired by this finding, k-means is applied to a group 600 test questions into 8 clusters with similar questions, employing Zero-Shot-CoT to generate reasoning chains. The goal is to identify clusters where Zero-Shot-CoT frequently fails. Cluster 2 notably exhibits a high error rate of 52.3% in Figure 2.15, suggesting potential limitations in Zero-Shot-CoT when addressing common problems and topics in the target tasks. Let's call the cluster with the highest error rate as the 'frequent-error cluster' (e.g., Cluster 2 in Figure 2.15). Consequently, the imperfect nature of generated reasoning chains using a zero-shot approach presents the risk of retrieving numerous similar questions within a frequent-error cluster when employing similarity-based

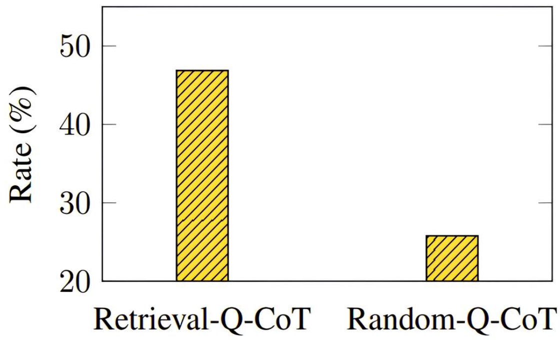

Figure 2.14: Unresolved rate. Reprinted from [66]

methods. In the case of the test question within the frequent-error cluster, Retrieval-CoT tends to create demonstrations with multiple similar errors more easily. Consequently, Retrieval-CoT often replicates similar mistakes as those generated by Zero-Shot-CoT for the in-context demonstrations, as demonstrated by the results in Figure 2.14 [66].

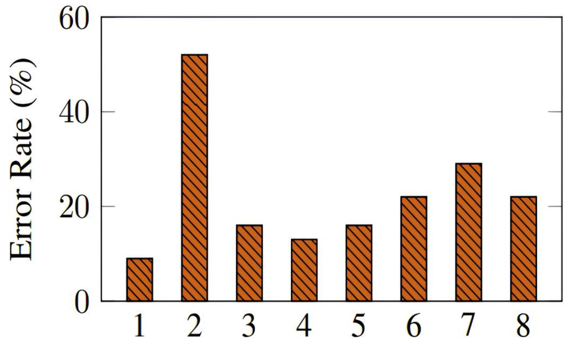

Figure 2.15: Error Rates Across Clusters. Reprinted from [66]

Retrieval-CoT May Select Redudant Examples

Retrieval-CoT retrieves in-context examples that are quite similar to the test example, which also makes them similar to each other. This might lead to the selection of redundant examples. However, the goal is to pick diverse and informative examples from which the LLM can learn. This represents a drawback of this prompting method.

2.4.4.7 *Auto-CoT and Diverse-CoT*

To mitigate the issue of "misleading by similarity" in Retrieval-CoT and reduce the impact of errors caused by Zero-Shot-CoT, [66] found that diversity matters for constructing the demonstrations. To also eleminate manual design of demonstrations, they proposed Auto-CoT, which is a prompting method that automatically samples questions from representative question sets to elicit chain-of-thought reasoning in LLMs. It utilizes Zero-Shot-CoT with a LLM for generating the answers without needing manually-designed

Algorithm 4 Phase 1: Clustering

Input: A set of questions $Q = \{x_i \in D_{\text{train}}\}$, the number of demonstrations k, sentence encoder SE_θ
Output: Sorted questions $x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots]$ for each cluster i ($i = 1, \ldots, k$)
procedure DIVERSE_COT(Q, k, SE_θ)
  $E = Generate\_Embedding(Q, SE_\theta)$ ▷ Go to *Generate_Embedding* procedure (Algorithm 1)
  Run KMeans(E, k)
  for each cluster $i = 1, \ldots, k$ **do**
    Sort questions $x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots]$ in ascending order of the distance to the cluster center
  end for
  return $x^{(i)}$ ($i = 1, \ldots, k$)
end procedure

Algorithm 5 Phase 2: Selection

Input: Sorted questions $x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots]$ for each cluster i ($i = 1, \ldots, k$) from Phase 1
Output: Demonstration list C
procedure CONSTRUCT($x^{(1)}, \ldots, x^{(k)}$)
  $C \leftarrow$ Empty list to store in-context demonstrations
  for $i = 1, \ldots, k$ **do**
    for $j = 1$ **to** $|x^{(i)}|$ **do**
      if $x_j^{(i)}, r_j^{(i)}$ satisfy selection criteria **then**
        Add $c_i = [Q : x_j^{(i)}, A : r_j^{(i)} \circ y_j^{(i)}]$ to C
        break
    end for
  end for
  return C
end procedure

demonstrations. Moreover, diversity helps to avoid the selection of redundant examples, which might be an issue with Retrieval-CoT.

Auto-CoT consists of two main phases: 1) question clustering, which entails grouping questions from a given training dataset into several clusters; and 2) demonstration sampling, where a representative question is chosen from each cluster, and its reasoning chain is generated using Zero-Shot-CoT. Despite using a clustering-based sampling method, there remains a possibility of selecting incorrect demonstrations. To tackle this, filtering criteria that prioritize shorter questions and rationales are utilized, helping to mitigate the limitations of imperfect Zero-Shot-CoT capabilities. Further details on this process are provided in a later section. It's important to note that while Retrieval-CoT chooses a different set of demonstrations for various test questions, Auto-CoT selects demonstrations from the training set, maintaining the same set of demonstrations for each test question [66].

In the study by [66], the assumption is that only a dataset of test questions is provided. The Auto-CoT method is applied to the rest of the test dataset to select in-context demonstrations, which are then evaluated using labels generated by Zero-Shot-CoT. In my adapted approach, designed for scenarios where both training and test datasets are available, I use a slightly modified version of this method. I refer to this adapted approach as Diverse-CoT. It is evaluated using both labels generated by Zero-Shot-CoT and labels generated by humans. Diverse-CoT selects in-context examples from the training dataset rather than the test dataset and uses both Zero-Shot-CoT-generated and human-generated labels, providing a broader perspective and enhancing the understanding of the method's performance in various conditions.

DIVERSE-COT ALGORITHM

1. The first phase involves the generation of embeddings for each question x_i from the training set D_{train} by invoking the "Generate_Embedding" procedure (Algorithm 1). This procedure computes the embedding list E, which is a list containing vector representation for each question in Q. The k-means clustering algorithm is applied to these question representations, resulting in k clusters of questions, where k is the desired number of in-context demonstrations [66]. Questions within each cluster are sorted into a list $x^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots]$ in ascending order based on their distance to the cluster's centroid.

2. In the second phase, the objective is to a sample representative question, that meets specific criteria, from each cluster. First, an empty list C is utilized to store in-context demonstrations. This list will be populated with selected questions and corresponding reasoning chains. For each cluster i in the range 1 to k, the algorithm iterates through the sorted list $x^{(i)}$. For each question $x_j^{(i)}$ in the sorted list of cluster i, the algorithm determines whether the question $x_j^{(i)}$ and its associated reasoning chain $r_j^{(i)}$ satisfy the predefined selection criteria. The selection criteria encompass two main constraints: number of tokens in the question constraint P and number of reasoning steps constraint R. The token constraint ensures that the question $x_j^{(i)}$ has no more than P tokens. This constraint is applied to avoid complex examples. The reasoning steps constraint ensures that the reasoning chain $r_j^{(i)}$ does not contain more than R reasoning steps. This constraint is used to filter out complex reasoning chains. If the selection criteria are met for a question $x_j^{(i)}$ within cluster i, a demonstration $c_i = [Q : x_j^{(i)}, A : r_j^{(i)} \circ y_j^{(i)}]$ is added to the demonstration list C. This demonstration encompasses the selected question, the reasoning chain, and the final answer. Finally, the algorithm returns the constructed demonstration list C, which contains representative questions meeting the selection criteria from each cluster.

The pseudocode of this method can be found in Algorithm 4. A visual illustration is shown in Figure 2.16.

Performance

Across ten public benchmark reasoning datasets, the experimental findings indicated that using GPT-3, Auto-CoT consistently either matches or surpasses the performance of the CoT paradigm, which relies on human-generated labels [66].

2.4.4.8 *Active-CoT*

[14] proposed a novel approach called Active-Prompt, designed to adapt LLMs to various tasks using task-specific example prompts augmented with

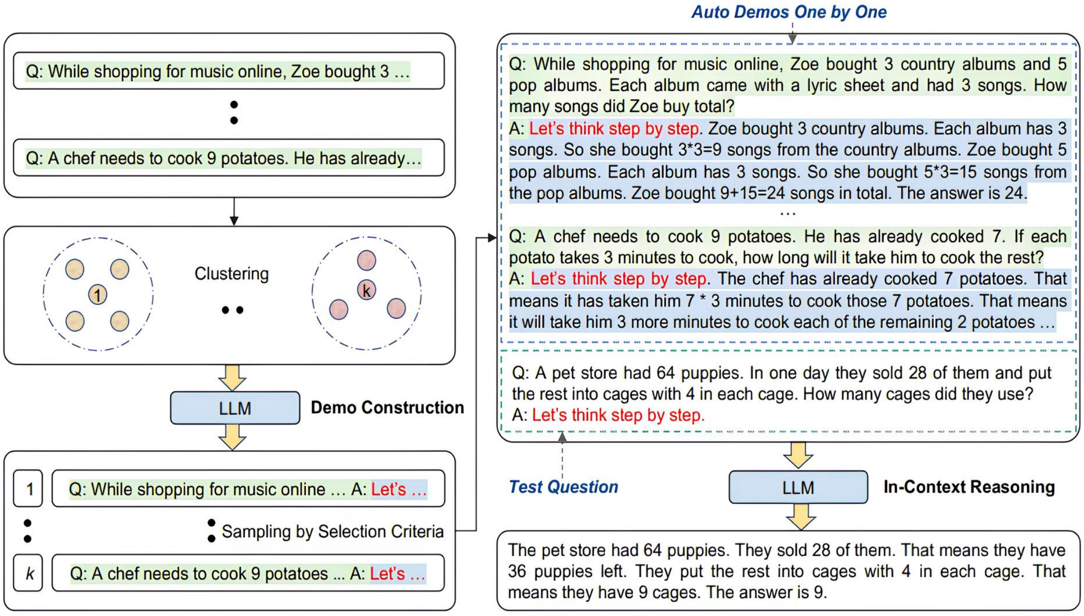

Figure 2.16: Example of Auto-CoT. Reprinted from [66]

human-designed CoT reasoning. To ensure consistency in naming across all methods, it is referred to as Active-CoT. The method involves identifying and selecting the most uncertain examples using a LLM, which are annotated with human-designed CoT reasoning. Additionally, in this thesis, Active-CoT is also used with LLM-generated labels. Drawing inspiration from the domain of uncertainty-based active learning, [14] introduced several metrics to quantify uncertainty including entropy, variance, disagreement etc. Moreover, they conducted in-depth analysis of different pool sizes, uncertainty metrics and zero-shot learning to demonstrate the effectiveness of this approach. It's important to note that while Retrieval-CoT chooses a different set of demonstrations for various test questions, Active-CoT selects demonstrations from the training set, maintaining the same set of demonstrations for each test question [66].

Uncertainty-based active learning

Active-CoT is related to uncertainty-based active learning [9, 53, 57] which aims to select the most informative data points for labeling in order to improve the performance of a machine learning model. Instead of randomly selecting data or following a fixed pattern, uncertainty-based active learning focuses on identifying the instances in a dataset where a machine learning model is most uncertain about its predictions. Recent investigations (as seen in [26, 55]) have highlighted the advantages of active learning-based approaches for fine-tuning LLMs in classification tasks.

Approaches for assessing uncertainty can be categorized into four distinct types, which are determined by the quantity (single or multiple) and the characteristics (deterministic or stochastic) of the Deep Neural Networks (DNN) [16].

- **Single deterministic methods** involve generating predictions through a single forward pass within a deterministic neural network. The assessment of uncertainty is typically achieved by using external approaches or by directly predicting it through the network itself [16].
- **Bayesian methods** encompass a range of stochastic DNN, specifically those in which two forward passes with the same input typically yield different outcomes [16].
- **Ensemble methods**, on the other hand, combine predictions from several distinct deterministic networks during the inference stage [16].
- **Test-time augmentation** techniques involve utilizing a single deterministic network to make predictions, but they enhance the input data during testing to generate multiple predictions. These multiple predictions are then used to assess the confidence level of the prediction [16].

Active-CoT utilizes the test-time augmentation technique for estimating the uncertainty as it leverages a single deterministic LLM on inference to generate several predictions, based on which the uncertainty is assessed.

Uncertainty Metrics

In order to choose a subset of questions from a given dataset, an unsupervised method is required. Previous research, as indicated in the study by [18], has demonstrated that reducing the uncertainty of the model contributes to improving its overall performance. Therefore, [14] introduced the uncertainty utilized by LLM as a metric for selecting the most informative data by infering a test question tr times to obtain tr potential answers. Various methods can be applied to estimate the uncertainty associated with a question. In the study by [14], four uncertainty metrics were considered: entropy, variance, disagreement, and self-confidence. For a given question q_i, $\hat{Y}_i = \{\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \ldots, \hat{y}_i^{(tr)}\}$ are tr answers generated by the LLM and UM is an uncertainy metric function which takes tr predictions as input and returns the uncertainty: $u_i = UM(\hat{Y}_i)$. This notation will be used to explain the various uncertainty metrics.

Entropy Entropy is used as one of the metrics to measure uncertainty in Active-CoT. It is a measure of randomness or the level of disorder in a given dataset or system. Assuming that there are h_i unique answers in tr total answers in $\hat{Y}_i$, in Active-CoT, entropy u_i for a given question q_i is calculated as follows:

$$u_i = UM(\hat{Y}_i) = -\sum_{j=1}^{h_i} \ln(P_\theta(\hat{y}_i^{(j)}|q_i)) \cdot P_\theta(\hat{y}_i^{(j)}|q_i) \tag{2.8}$$

In this context, $P_\theta(\hat{y}_i^{(j)}|q_i)$ stands for the frequency at which a particular predicted answer appears among all the predictions. The higher the entropy, the more uncertain the answer is. The objective of Active-CoT would be to find the question q_i that maximizes the entropy:

$$\arg\max_{i} - \sum_{j=1}^{h_i} \ln(P_\theta(\hat{y}_i^{(j)}|q_i)) \cdot P_\theta(\hat{y}_i^{(j)}|q_i) \tag{2.9}$$

As a result, questions with largest entropy will be selected [14].

Disagreement Disagreement measures the diversity of tr several answers generated by a language model $\hat{Y}_i = \{\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \ldots, \hat{y}_i^{(tr)}\}$ for a given question q_i. It is determined by identifying the distinct answers in the predictions, resulting in a collection of h_i unique answers $\hat{H}_i = \{\hat{y}_i^{(1)}, \hat{y}_i^{(2)}, \ldots, \hat{y}_i^{(h_i)}\}$. Then the disagreement is calculated by $u_i = UM(\hat{Y}_i) = \frac{h_i}{tr}$ [14].

Variance proposed variance as uncertainty metrics which is a measure of spread in the predictions from the average prediction.

$$u_i = UM(\hat{Y}_i) = \frac{1}{tr-1} \sum_{j=1}^{tr} (\hat{y}_i^{(j)} - \bar{y}_i^{(j)})^2 \mid_{q=q_i} \tag{2.10}$$

where $\bar{y}_i^(j) = \frac{1}{tr} \sum_{j=1}^{tr} \bar{y}_i^{(j)}$ is the average prediction. As a result the objective of Active-CoT would be to find the question q_i that maximizes the variance:

$$\arg\max_{i} \frac{1}{tr-1} \sum_{j=1}^{tr} (\hat{y}_i^{(j)} - \bar{y}_i^{(j)})^2 \mid_{q=q_i} \tag{2.11}$$

As a result, questions with largest variance will be selected. This metric has its limitations because when answers are of various magnitudes, it can cause certain predictions to have a more significant impact on the computed variance. To address the challenge posed by these widely varying numerical values, [14] suggested a strategy to normalize the predictions by considering all the numbers mentioned within the respective questions. For example, given the question "Julie is reading a x1-page book. Yesterday, she was able to read x2 pages and today, she read twice as many pages as yesterday. If she wants to read half of the remaining pages tomorrow, how many pages should she read?" and a predicted answer y, the normalized prediction would be $\frac{y}{|x_1|+|x_2|}$ [14].

Self-Confidence Determining uncertainty can also be accomplished using the LLM itself, which is often referred to as self-confidence. Self-confidence measure can be acquired by interacting with the model through a predefined template T. For instance, a template could be created as follows: "For a given question q and its predicted answer a, provide a confidence assessment for the answer, selecting from the options (a) very confident, (b) confident, (c) not confident, or (d) wrong answer." Subsequently, the questions with the lowest confidence scores are identified.

$$u_i = \max_j P_\theta(\hat{y}_i^{(j)} | q_i) \tag{2.12}$$

where $P_\theta(\hat{y}_i^{(j)} | q_i)$ represents a categorical variable selected from a predefined set containing options such as "very confident," "confident," "not confident," and "wrong answer". As a result, the objective of Active-CoT would be to find the question q_i that minimazes the self-confidence:

$$\arg\min_i \max_j P_\theta(\hat{y}_i^{(j)} | q_i) \tag{2.13}$$

[14]

Algorithm 6 Active-CoT

Input: Set of training questions $Q = \{x_i \in D_{\text{train}}\}$, LLM LLM_θ, number of trails tr, temperature te, number of in-context demonstrations k, uncertainty metric function UM, in-context demonstrations AC
Output: Demonstration list C
procedure ACTIVE_COT($Q, LLM_\theta, tr, te, k, UM, AC$)
  $U = Estimate\_Uncertainty(Q, LLM_\theta, tr, te, UM, AC)$
  ▷ Go to *Estimate_Uncertainty* procedure (Algorithm 2)
  Get the largest k uncertainties from U in descending order
  with indices $\{\sigma(1), \ldots, \sigma(k)\}$
  $C = [(x_{\sigma(1)}, r_{\sigma(1)}, y_{\sigma(1)}), \ldots, (x_{\sigma(k)}, r_{\sigma(k)}, y_{\sigma(k)})]$
  return C
end procedure

ACTIVE-COT ALGORITHM

1. The first phase involves the uncertainty estimation for each question x_i from the training set D_{train} by invoking the "Estimate_Uncertainty" procedure (Algorithm 2). This procedure computes the uncertainty list, U, which contains the estimated uncertainties for each question in Q.

2. Next, the algorithm selects the k questions with the highest uncertainties in descending order. The selected questions are represented by their indices $\{\sigma(1), \ldots, \sigma(k)\}$.

3. The final output C, is a list of demonstrations that includes the selected questions $(x_{\sigma(1)}, \ldots, x_{\sigma(k)})$, along with their reasoning chains $(r_{\sigma(1)}, \ldots, r_{\sigma(k)})$ and final answers $(y_{\sigma(1)}, \ldots, y_{\sigma(k)})$.

The pseudocode of this method can be found in Algorithm 6.

Performance

Active-CoT outperformed Auto-CoT as well as Random baseline by significant margins in GSM8K, MultiArith, and AddSub, demonstrating high performance across these datasets encompassing arithmetic, commonsense, and symbolic reasoning [14].

2.4.5 *Selection criteria of simple demonstrations*

To annotate the data, one method involves utilizing GPT-3.5-turbo with the Zero-Shot-CoT approach, similar to the method presented in [66], which was used in Auto-CoT. However, it's important to note that using a LLM to generate answers including reasoning chains may result in inaccuracies. As a result, there is a risk of obtaining wrong demonstrations (i.e., demonstrations with incorrect answers). For instance, in the retrieval-based approach Retrieval-CoT, this may lead the LLM to reproduce similar mistakes when reasoning for the test question [66]. Therefore, for this prompting method, it becomes evident that the avoidance of wrong demonstrations may help enhace few-shot performance. Even with other prompting methods, the aim is to choose accurate demonstrations. This can be achieved through a selection criterion based on the number of token in the question and number of reasoning steps of an example.

In all methods, except for Diverse-CoT and Diverse-Active-KMeans-CoT, a selection criterion is applied before implementing the prompting strategy. This criterion aims to reduce the training dataset to a smaller subset of data, favoring simpler questions and rationales, as proposed by [66]. For Diverse-CoT and Diverse-Active-KMeans-CoT, this selection criterion is applied to each cluster due to computational efficiency.

I begin my analysis by examining how the number of tokens in questions and the number of reasoning steps affect the accuracy of generated rationales and why selecting simpler examples holds significance. Following this, I explain how I determined the selection criteria for my specific case.

2.4.5.1 *Effectiveness of Prioritizing Simplicity in Demonstrations*

In this section, I analyze the relationship between the complexity of in-context demonstrations and the accuracy of generated demonstrations, specifically rationales. The focus is on understanding whether the simplicity or complexity of the demonstrations plays a significant role in the accuracy of the generated rationales, as we aim to select correct and high-quality in-context demonstrations.

I first compare the distribution of the number of reasoning steps generated by GPT-3.5-turbo in the Zero-Shot-CoT setting with the ground truth distribution. This comparison provides valuable insights into how well the model aligns with the actual distribution of reasoning steps. In Figure 2.17, which includes boxplots for both distributions, we observe that GPT-3.5-turbo tends to generate a higher number of reasoning steps. The same pattern is observed in both the GSM8K and AQUA datasets. This observation emphasizes the need to limit the number of intermediate steps to ensure that we select demonstrations that adhere to the true distribution of reasoning steps.

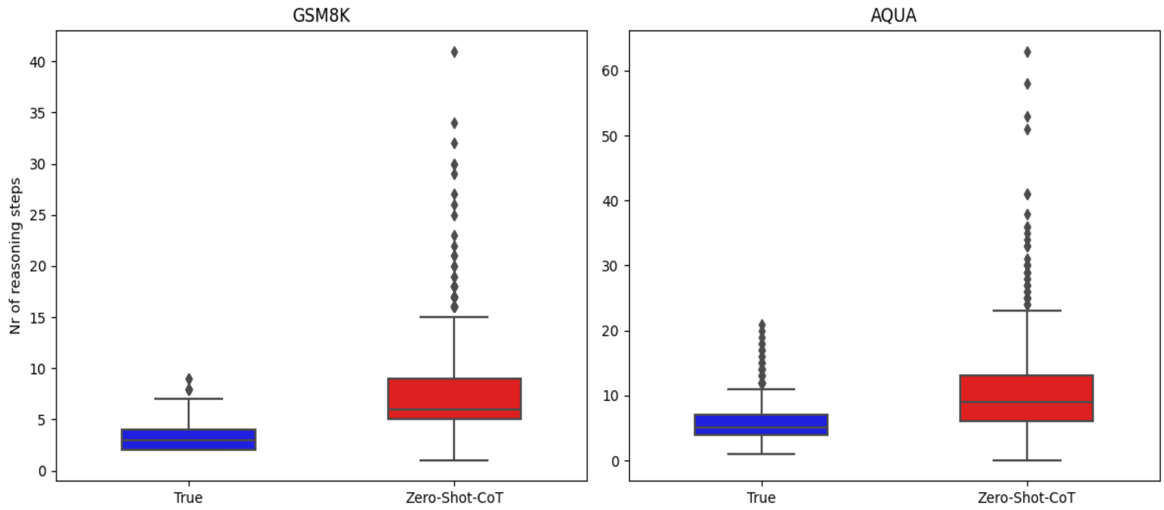

Figure 2.17: Distribution comparison of number of reasoning steps between human-generated labels (True) and LLM-generated labels with Zero-Shot-CoT.

To gain a deeper understanding about the relationship between the number of reasoning steps and the count of tokens in questions and their impact on misclassification, I examine the distribution of the number of intermediate steps and the number of tokens in questions for two distinct subsets within the GSM8K dataset: one comprised of correctly classified examples and another consisting of incorrectly classified examples. Similar analyses are conducted for the AQUA dataset. From the boxplots of the GSM8K dataset in Figure 2.18, it becomes evident that, in this dataset, the greater the number of intermediate steps generated by GPT-3.5, the higher the likelihood of an incorrect classification. Furthermore, this pattern remains consistent when assessing the number of tokens in the questions. Specifically, a greater number of tokens in the question is associated with a higher probability of misclassification. Similar insights are observed in the analysis of the AQUA dataset, as depicted in the boxplots presented in Figure 2.19.

2.4.5.2 *Selection Criteria Strategy: Finding the Optimal Values*

[66] used a selection criteria similar to those outlined in the hand-crafted demonstrations by [63]. The aim is to encourage the sampling of simpler questions (with no more than 60 tokens) and rationales (with no more than 5 reasoning steps) for labeling. They utilized a different model, text-davinci-002, for labeling, in contrast to my use of GPT-3.5-turbo.

Nevertheless, applying this criterion to the zero-shot-generated rationales by GPT-3.5-turbo in my case leads to a very low number of selected demonstrations. Out of 1000 training examples for GSM8K, only 191 examples were selected. Similarly, for AQUA, only 172 examples were selected. This reduc-

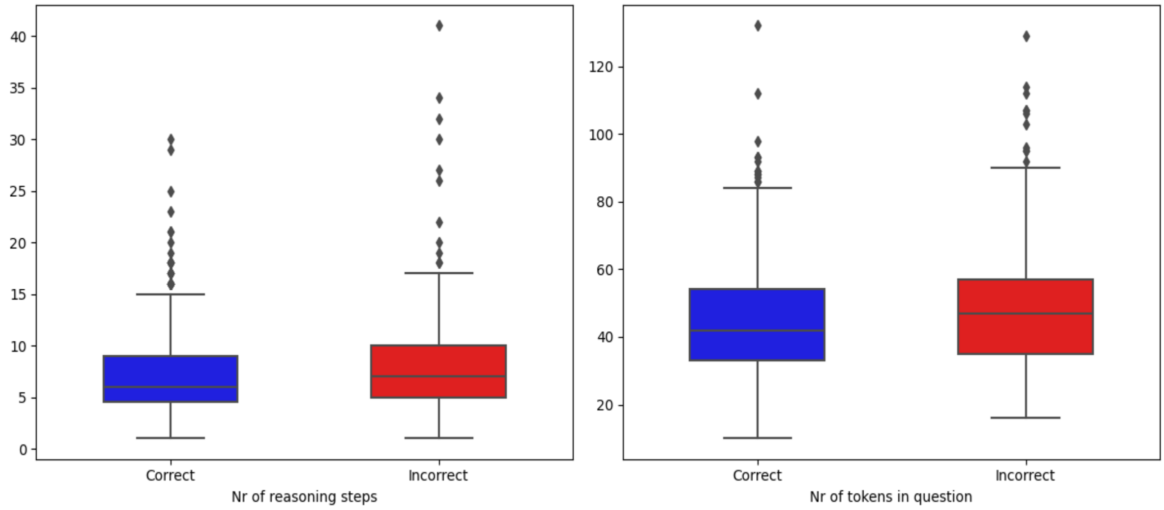

Figure 2.18: Distribution of number of tokens in questions and number of reasoning steps for GSM8K across two subsets: correctly classified and incorrectly classified.

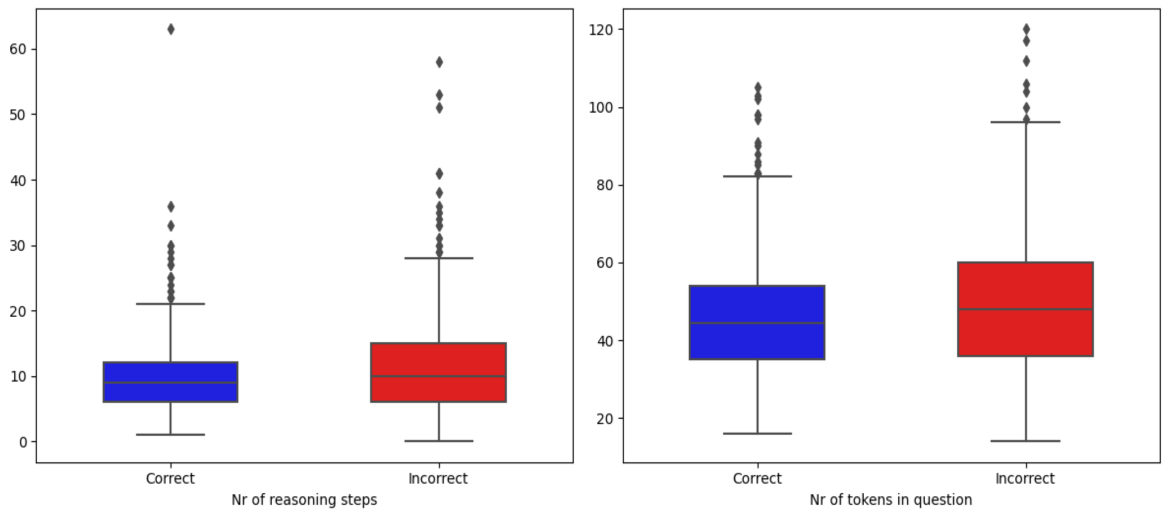

Figure 2.19: Distribution of number of tokens in questions and number of reasoning steps for AQUA across two subsets: correctly classified and incorrectly classified

tion in the training dataset to a very small size may omit informative examples crucial for few-shot prompting.

When annotating with Zero-Shot-CoT, I maintain the assumption that we do not have access to the ground truth labels. This assumption remains in place even when selecting the optimal values based on the distribution of the number of intermediate steps and the number of tokens in questions, without reference to the distributions of the subset containing correctly classified examples and incorrectly classified examples. For determining a reasonable value for the maximum number of intermediate steps and the maximum number of tokens in questions, I use a percentile-based approach. Specifically, I choose a value falling between the 75th percentile and the 75th percentile plus 1.5 times the Interquartile Range (IQR). This approach is particularly suitable for non-normally distributed data. Any value beyond the 75th percentile plus 1.5 times the IQR is regarded as an outlier and thus excluded from the selection criteria. It's important to note that this approach may not be appropriate for all use cases, as it relies primarily on the data distribution and doesn't consider domain knowledge or external factors. For AQUA, I've chosen a maximal number of tokens in question of 70 and a maximal number of intermediate steps of 15, while for GSM8K, I've chosen a maximal number of tokens in question of 86 and a maximal number of intermediate steps of 12.

3

PROPOSED PROMPTING METHODS

Diversity may help to mitigate the issue of "misleading by similarity" [66] and [14] demonstrated that selecting demonstrations assosicated with high model uncertainy improved the few-shot performance [14]. This emphasizes the significance of both diversity and uncertainty in identifying the most informative examples. [14] considered the combination of diversity and uncertainty as an important future direction. Therefore, I propose new prompting methods: Diverse-Active-KMeans-CoT and Diverse-Active-KMeansPlusPlus-CoT. These methods aim to combine diversity and uncertainty, ensuring the selection of examples that are diverse from each other, particularly in terms of semantic representation of the questions, while also exhibiting high model uncertainty.

3.0.1 *Diverse-Active-KMeans-CoT*

Diverse-Active-KMeans-CoT combines the strategies from Diverse-CoT and Active-CoT. It follows a similar approach to Diverse-CoT with one key difference. While Diverse-CoT sorts questions within each cluster by their distance to the cluster center in ascending order, Diverse-Active-KMeans-CoT arranges questions within each cluster in descending order based on their uncertainty. This prompting method aims to prevent the issue of "misleading by similarity" by ensuring a diverse selection of questions. It achieves this by sampling questions from each cluster, focusing on diversity, while also addressing the challenge of model uncertainty. Specifically, it chooses examples associated with the highest uncertainty within each cluster, if they fulfill the specified selection criteria.

3.0.2 *Diverse-Active-KMeansPlusPlus-CoT*

This algorithm's objective is to choose a set of k demonstrations from a dataset of questions and answers, with a primary focus on diversity and questions where the model is uncertain. To achieve this, it introduces a weighted F1-score metric that considers both diversity and uncertainty. This weighting allows for the prioritization of either diversity or uncertainty.

The Diverse-Active-KMeansPlusPlus-CoT approach uses a similar strategy to KMeans++, which involves a smarter way to initialize the centroids compared to random selection in the KMeans clustering method [39]. It starts by selecting the in-context example with the highest uncertainty as the first demonstration. It then iteratively determines the next example to select based on the computed F1-score, which takes into account the similarity between the candidate questions and the questions already chosen in previous iterations, as well as uncertainty. It does so either by sampling based on the computed probability distribution, resulting from the F1-scores, or in a deterministic manner by selecting the example with the highest F1-score. This continues until k examples are selected. A high F1-score for a question indicates its relatively high diversity compared to the already selected questions and its relatively high uncertainty as well.

DIVERSE-ACTIVE-KMEANSPLUSPLUS-COT ALGORITHM

1. The first step involves the embedding generation and uncertainty estimation for each question x_i from the training set D_{train} by invoking respectively the "Generate_Embedding" procedure (Algorithm 1) and the "Estimate_Uncertainty" procedure (Algorithm 2). The "Generate_Embedding" procedure computes the embedding list E, whereas the "Estimate_Uncertainty" procedure computes the uncertainty list U.

2. Then, an empty list C is initialized to store in-context demonstrations. The initial step of the selection process involves identifying the example index $\sigma(1)$ from D_{train} with the highest uncertainty based on U. This selected example, denoted as $D_{\text{train}}[\sigma(1)]$, is added to C, and it is removed from the available question pool E as it has already been selected.

3. For the remaining $k-1$ demonstrations, the algorithm iteratively selects examples by adding them to C based on their similarity to the current C and their uncertainty. We iterate through the index z until we reach the desired number of in-context demonstrations k. Within each iteration, a demonstration with index $\sigma(z)$ is selected. Within this loop, we initialize empty arrays F to store the F1-scores and NU , NS which are needed to perform the normalization of uncertainties and similarities for each question x_i from D_{train}. This normalization is essential to prevent a disproportionate influence on the F1-score due to varying scales between the similarity or uncertainty measures. Inside this loop, we iterate over the training question embeddings that have not been selected yet and create an empty list S_i to store similarities for embedding e_i, which corresponds to the question x_i, with the examples already selected in C. Therefore, for each question embedding e_i from E, we compute the cosine similarity cos_sim between e_i and the embeddings in C, represented as e_j, by iterating through the embeddings of C with c_j. The cosine similarity is calculated as $cos\_sim(e_i, e_j) = \frac{e_i \cdot e_j}{\|e_i\|_2 \|e_j\|_2}$ for each pair of embeddings, which is then transformed using $\exp(-cos\_sim(e_i, e_j))$. This transformed value $s_{i,j}$ is then stored in the list S_i. The utilization of an exponential transformation aims to minimize the cosine similarity in order to maximize diversity and ensures a transformation resulting in a positive value as well.

4. For each e_i, we determine the maximum similarity from the array S_i, which is denoted as s_i, and add it to the list NS. Additionally, we

Algorithm 7 Diverse-Active-KMeansPlusPlus-CoT

Input: Training set D_{train}, the number of demonstrations k, sentence encoder SE_θ, LLM LLM_θ, number of trails tr, temperature te, an uncertainty metric function UM, in-context examples AC, weight for uncertainty β, probability top_p, greedy flag $greedy$ and $args$ includes all these arguments

Output: Demonstration list C

procedure DIVERSE_ACTIVE_KMEANSPLUSPLUS_COT($args$)

  $Q = \{x_i \in D_{\text{train}}\}$

  Create an empty list C to store in-context demonstrations.

  $U = Estimate\_Uncertainty(Q, LLM_\theta, tr, te, UM, AC)$

  ▷ Go to *Estimate_Uncertainty* procedure (Algorithm 2)

  $E = Generate\_Embedding(Q, SE_\theta)$

  ▷ Go to *Generate_Embedding* procedure (Algorithm 1)

  $\sigma(1) \leftarrow$ **Get** the index of the example d from D_{train} with the highest uncertainty based on U.

  Add $D_{\text{train}_{\sigma(1)}}$ to C.

  Remove $E_{\sigma(1)}$ from E.

  for $z = 2$ to k **do**

    Create empty lists F, NU, NS to respectively store F1-scores, normalized uncertainties and normalized similarities for $\{e_i\}$

    for each e_i in E **do**

      Create an empty list S_i to store similarities for e_i with current examples from C

      for each c_j in C **do**

        $cos\_sim(e_i, e_j) = \frac{e_i \cdot e_j}{\|e_i\|_2 \|e_j\|_2}$

        Add $s_{i,j} = \exp(-cos\_sim(e_i, e_j))$ to S_i

      end for

      $s_i \leftarrow$ **Get** the maximum similarity from S_i and **Add** to NS

      $u_i \leftarrow$ **Get** uncertainty of e_i from U and **Add** to NU

    end for

    for $w = 1$ to $|E|$ **do**

      $ns_w = ns_w / \sum_{i=1}^{|NS|} ns_i$

      $nu_w = ns_w / \sum_{i=1}^{|NU|} nu_i$

      Add $f1_w = \frac{(\beta^2+1) \cdot ns_w \cdot nu_w}{\beta^2 \cdot ns_w + nu_w}$ to F

    end for

    if not *greedy* **then**

      $top\_F1 \leftarrow$ **Get** the highest top_p% values from F

      Compute probabilities P using softmax on top_F1

      $\sigma(z) \leftarrow$ **Sample** an index randomly based on probability distribution P

    else

      $\sigma(z) \leftarrow$ **Get** index with the highest F1-score in F

    end if

    Add $c_i = D_{\text{train}_{\sigma(z)}} = [Q : x_{\sigma(z)}, A : r_{\sigma(z)} \circ y_{\sigma(z)}]$ to C

    Remove $E_{\sigma(z)}$ from E.

  end for

  return C

end procedure

retrieve the uncertainty u_i for e_i from the uncertainty array U and add it to the list NU.

5. After computing and storing the similarities and uncertainties for the questions in E, we proceed to normalize them to bring similarity and uncertainty into the same scale, thus avoiding the influence of specific values of high uncertainty or similarity. To achieve this, we iterate through the uncertainties NU and similarities NS, using index w from 1 to the length of E. Two operations are performed on each element: dividing each uncertainty nu_w by the sum of all uncertainties $nu_w = ns_w / \sum_{i=1}^{|NU|} nu_i$, and dividing each similarity ns_w by the sum of all similarities $ns_w = ns_w / \sum_{i=1}^{|NS|} ns_i$. This yields relative scores. The normalized uncertainty nu_w and similarity ns_w are then used in the computation of the F1-score, represented as $f1_w = \frac{(\beta^2+1)\cdot ns_w \cdot nu_w}{\beta^2 \cdot ns_w + nu_w}$. These F1-scores are added to the list F.

6. Following the computation of F1-scores for all e_i in E, we proceed to assess whether a greedy approach is specified (i.e., if *greedy* is true). If the greedy approach is selected, we identify the index $\sigma(z)$ with the highest F1-score in F. Alternatively, if the greedy approach is not selected (i.e., *greedy* is false), we select the top $top\_p\%$ scores from F, which are stored in top_F1. In this context, examples with relatively low F1-scores are excluded from the sampling process. These top F1-scores are transformed into probabilities P using softmax, which amplifies the differences between values, giving priority to high F1-scores. Subsequently, we perform random-weighted sampling based on the probability distribution P and obtain the selected index $\sigma(z)$.

7. The corresponding example from the training set, denoted as $c_i = D_{\text{train}_{\sigma(z)}} = [Q : x_{\sigma(z)}, A : r_{\sigma(z)} \circ y_{\sigma(z)}]$, is added to the demonstration list C, and the corresponding question embedding $E_{\sigma(z)}$ is removed from E. This entire process is repeated until we have selected k in-context demonstrations. The algorithm returns the list C containing k selected demonstration examples.

The pseudocode of this method is shown in Algorithm 7.

3.0.3 *Diverse-Active-KMeansPlusPlus-Retrieval-CoT*

Even though diversity may help prevent the selection of incorrect demonstrations and mitigate the issue of 'misleading by similarity,' insights from [32] have highlighted the influence of question similarity with the test question on few-shot performance. Therefore, I integrated the retrieval component into the algorithm Diverse-Active-KMeansPlusPlus-CoT and propose a new method, Diverse-Active-KMeansPlusPlus-Retrieval-CoT. This method is designed to address the challenge of selecting a diverse set of demonstrations that exhibit high model uncertainty, ensuring the selection of semantically similar questions related to a test question. It operates in two main phases: Diverse-Active-KMeansPlusPlus-CoT and Retrieval-CoT.

- **Phase 1: Diverse-Active-KMeansPlusPlus-CoT** In the first phase, the algorithm leverages Diverse-Active-KMeansPlusPlus-CoT. This process allows for the selection of p examples from the training set. The selected examples are characterized by two essential criteria: high model uncertainty and diversity. Diverse-Active-KMeansPlusPlus-CoT prioritizes diversity, minimizing the risk of "misleading by similarity" and avoiding the selection of redundant examples. Simultaneously, it aims to select examples characterized by high model uncertainty.

- **Phase 2: Retrieval-CoT** In the second phase, Retrieval-CoT further enhances the selection process. This phase is executed for a specific test question. It aims to identify and select the most similar questions from a given set of questions, which includes the ones selected during the first phase. This step ensures that the selected examples closely aligns with the test question.

The pseudocode of this method is shown in Algorithm 8.

Algorithm 8 Diverse-Active-KMeansPlusPlus-Retrieval-CoT

1: **Input:** Training set D_{train}, the number of demonstrations for Diverse-Active-KMeansPlusPlus-CoT p, sentence encoder SE_θ, LLM LLM_θ, number of trails tr, temperature te, uncertainty metric function UM, in-context examples AC, weight for uncertainty β, probability top_p, greedy flag $greedy$, $args_phase1$ includes all the previous arguments, number of in-context demonstrations k, test question x_{test}
2: **Output:** Demonstration list C
3: **procedure** DIVERSE_ACTIVE_KMEANSPLUSPLUS_RETRIEVAL_COT($Q, LLM_\theta, tr, te, p, UM, AC$)
4: $PC = Diverse\_Active\_KMeansPlusPlus\_CoT(args_phase1)$
▷ Go to $Diverse\_Active\_KMeansPlusPlus\_CoT$ procedure (Algorithm 7)
5: $PQ = \{x_i \in PC\}$
6: $C = Retrieval\_CoT(PQ, SE_\theta, k, x_{\text{test}})$
▷ Go to $Retrieval\_CoT$ procedure (Algorithm 3)
7: **return** C
8: **end procedure**

The main challenge of Diverse-Active-KMeansPlusPlus-Retrieval-CoT lies in determining the optimal value for p. If prioritizing uncertainty and diversity over similarity with the test question is crucial, p should be set to a lower value. Conversely, a higher value for p prioritizes the similarity with the test question. This is because a greater number of examples available for selection in the second phase increases the likelihood of choosing examples that closely resemble the test question.

4 EXPERIMENTS

A part of the material presented in the Experiments section has been reformulated based on outputs generated by ChatGPT [43].
Here is an example of a prompt:
I am going to present the evaluation results regarding my master thesis. You should write a well-structured output and compare the performance of the few-shot prompting methods against the random approach on the GSM8K dataset with human-generated labels. Ensure readability and clarity. Here are the results: The performance of Random-CoT is 79.5%. The performance of few-shot prompting methods is as follows: Active-CoT has 83.0% and Diverse-CoT has 78.1%. Do not make up any information. Only provide a comparison between these methods..

This the response generated by ChatGPT:
Title: Comparative Analysis of Few-Shot Prompting Methods vs. Random Approach on GSM8K Dataset with Human-Generated Labels
Introduction:
The evaluation results of various prompting methods on the GSM8K dataset with human-generated labels provide valuable insights into the efficacy of these techniques. In this study, we focus on comparing the performance of the Random-CoT approach against two prominent few-shot prompting methods: Active-CoT and Diverse-CoT.
Random-CoT Performance:
The Random-CoT method achieved a performance score of 79.5%. This approach randomly selects prompts to generate responses, serving as a baseline for comparison with more sophisticated prompting techniques.
Few-Shot Prompting Methods:
a. Active-CoT: The Active-CoT method demonstrated a notable performance improvement, achieving an accuracy of 83.0%. This approach involves strategically selecting informative prompts to enhance the model's understanding and response generation.
b. Diverse-CoT: The Diverse-CoT method, while competitive, exhibited a performance of 78.1%. This method aims to diversify prompts, providing the model with a broader range of context for generating responses.
Comparison and Analysis:
Active-CoT vs. Random-CoT:
Active-CoT outperforms Random-CoT by 3.5 percentage points, indicating that actively selecting informative prompts contributes to enhanced model performance. The strategic prompt selection in Active-CoT likely enables the model to focus on relevant information, leading to more accurate and contextually appropriate responses.
Diverse-CoT vs. Random-CoT:
Diverse-CoT performs slightly below Random-CoT by 1.4 percentage points, suggesting that diversifying prompts may not necessarily lead to a significant improve-

ment in model performance on this dataset.
While diversity in prompts is beneficial, the specific context and characteristics of the GSM8K dataset may not fully capitalize on the advantages offered by a diverse set of prompts.
Conclusion:
In conclusion, the evaluation results on the GSM8K dataset with human-generated labels highlight the effectiveness of few-shot prompting methods compared to the Random-CoT baseline. Active-CoT, with its strategic prompt selection, stands out as the most successful method, surpassing both Random-CoT and Diverse-CoT. These findings underscore the importance of prompt selection strategies in optimizing model performance in natural language processing tasks.

ChatGPT often generates long responses and may include inaccurate information, even when explicitly instructed not to. As a result, I have been careful in rephrasing its responses [41].

4.1 Experimental Setup

4.1.1 *Datasets and Evaluation Metric*

This work focuses on mathematical word problems, as illustrated in Figure 2.10. These problems serve as a measure of the arithmetic reasoning capabilities of LLMs. While these problems may appear straightforward to humans, they often pose challenges for LLMs, as observed in previous studies [20, 47]. In order to provide evidence for the efficiency of several few-shot prompting methods, experiments are conducted on two arithmetic reasoning datasets: GSM8K and AQUA.

GSM8K GSM8K includes 8,500 well-crafted grade school math problems, created by people. It's split into 7,500 problems for training and 1,000 for testing. These math problems usually require 2 to 8 steps to solve, involving basic math operations like addition, subtraction, division, and multiplication. They are designed to be solvable by an average middle school student [8]. The GSM8K dataset is provided in a file format where each line corresponds to a single grade school math problem, saved as a JSON dictionary. The JSON dictionary includes:

1. **Question:** A natural language definition of the problem to be solved.
2. **Answer:** The answer to the problem, which is formatted with calculation annotations and the final numeric solution is marked as the last line of the solution, preceded by "####."

AQUA The AQUA dataset consists of approximately 100,000 algebraic word problems, each structured as a JSON object comprising four key components:

1. **Question:** A natural language definition of the problem to be solved.

2. **Options:** Five possible choices (A, B, C, D, and E) are provided, with one of them being the correct answer.

3. **Rationale:** A natural language description that explains the solution to the problem.

4. **Correct:** This indicates which option among A, B, C, D, and E is the correct answer.

The AQUA dataset has been entirely created through crowdsourcing, using the methods explained in [30].

The evaluation metric used is exact match accuracy. The generated final answers by LLM are compared to the ground truth answers.

4.1.2 *Prompting Methods and Baselines*

4.1.2.1 *Baselines*

In my experiments, these three approaches are established as baselines:

- **Zero-Shot-CoT using GPT-3.5-turbo:** This prompting technique provides task instructions without any demonstrations. It is the baseline for the other few-shot chain-of-thought prompting methods.

- **Random-Standard:** A few-shot prompting approach that includes questions and final answers but excludes the reasoning chains. It is the baseline for both Random-CoT and the other few-shot standard methods.

- **Random-CoT:** A few-shot prompting method that, in contrast to Random-Standard, encompasses questions, final answers, and their associated reasoning chains. It is the baseline for the other few-shot chain-of-thought prompting methods.

4.1.2.2 *Prompting methods*

In my experiments, the following prompting methods were tested:

- **Diverse-CoT :** A prompting method that prioritizes selecting questions with diverse semantic meanings.

- **Active-CoT :** A prompting method, which aims to select examples in which LLM is most uncertain, focusing on reducing the model uncertainty.

- **Retrieval-CoT :** A prompting method, which selects questions that are most similar to the test question.

- **Diverse-Active-KMeans-CoT :** A prompting method, which partitions questions into clusters using KMeans and selects examples with high uncertainty within each cluster, combining diversity and uncertainty.

- **Diverse-Active-KMeansPlusPlus-CoT :** A prompting method, which prioritizes diversity and uncertainty in example selection using a weighted F1-score metric. It begins with the example having the highest uncertainty and iteratively selects the next examples based on a weighted F1-score considering the similarity between questions and previously chosen examples.
- **Diverse-Active-KMeansPlusPlus-Retrieval-CoT :** A prompting method, which is designed to choose a diverse set of demonstrations that exhibit high model uncertainty, with a focus on selecting semantically similar questions related to a test question.

Furthermore, Random-CoT, Diverse-CoT, Active-CoT, and Retrieval-CoT are also tested with standard prompting (without including the reasoning chains) and are referred to as respectively: Random-Standard, Diverse-Standard, Active-Standard, Retrieval-Standard.

4.1.3 *Infrastructure*

4.1.3.1 *Models and Infrastructure Setup*

To thoroughly evaluate the effectiveness and robustness of various prompting methods, it is essential to assess their performance across diverse models, including open-source LLMs. The following LLMs have been selected for this evaluation:

- **GPT-3.5-turbo:** This model is accessible through the UPTIMIZE GPT API, which is provided by Merck KGaA and serves as a proxy for the Azure OpenAI API. It offers capabilities on par with the Azure API but uses a distinct API key. Due to its exceptional reasoning abilities, this model is the primary choice for my experiments. I use the model version gpt-35-turbo-0613.
- **GPT-4:** GPT-4 is accessible via the OpenAI API and can be utilized locally without the need for specialized infrastructure. Given its enhanced capabilities compared to GPT-3.5-turbo, it is specifically evaluated in zero-shot scenarios (Zero-Shot-CoT).
- **falcon-40b-instruct:** Falcon-40B-Instruct is a causal decoder-only model with 40 billion parameters. It is recognized as one of the leading open-source models.
- **falcon-7b-instruct:** Falcon-7B-Instruct is a causal decoder-only model with 7 billion parameters. It is based on Falcon-7B and further fine-tuned using a combination of chat and instruct datasets. For assessing the performance of open-source LLMs, including falcon-40b-instruct and falcon-7b-instruct, I run them on an ec2-instance of type g5.12xlarge, equipped with four NVIDIA A10G Tensor Core GPUs, each with 96GB of memory and a total of 48 vCPUs.

4.1.3.2 *Distributed Inference*

Running LLMs for inference can be challenging, often resulting in unexpectedly slow performance, even when using expensive hardware. To address these challenges, I use vLLM, an open-source library designed for efficient LLM inference and deployment. vLLM utilizes PagedAttention, an innovative attention algorithm. It offers exceptional speed with:

- Cutting-edge serving throughput
- Effective management of attention key and value memory with PagedAttention
- Continuous batching of incoming requests
- Optimized CUDA kernels

Furthermore, vLLM seamlessly integrates with widely-used HuggingFace models, supporting high-throughput serving with various decoding techniques, including parallel sampling, beam search, and other options [25].

[25] demonstrated that vLLM achieves up to 24 times greater throughput than HuggingFace Transformers [65] and up to 3.5x higher throughput than Text Generation Inference [22] without requiring any modifications to the model architecture, thereby setting a new standard for LLM deployment. Importantly, vLLM is seamlessly integrated into the langchain, the primary package used in this study. This integration streamlines the use of OpenAI models as well as open-source models with prompt templates in langchain.

4.1.4 *Hyperparameters*

The results presented in this paper will be compared to those in [14]. However, a comparison with the work of [66] is not feasible, as the latter assumed the absence of training data and utilized the test set both for selecting in-context demonstrations and for evaluation. Notably, [14] used older models from the GPT-3.5 family at that time. Consequently, I conduct a comparison among the performance of different models. It's important to note that comparing prompting methods evaluated on different models may not yield a fair assessment.

I maintain consistency in hyperparameters with [14] for several aspects:

- In my experiments, the original training datasets were not used due to their size. Instead, a random sample of 1000 data points was drawn from the training set, following a similar approach as [14], to reduce computational costs. It's important to acknowledge that this approach may impact the accuracy of uncertainty estimation. Generally, a larger training dataset provides a more precise estimation of data distribution and uncertainty. With additional financial support, an improvement in the model's performance is anticipated [14]. To ensure a fair comparison, I use the same test set as used by [14].

- All few-shot prompting methods use the identical number of examples as utilized in [14]: 8 for GSM8K and 4 for AQUA.
- I use the same temperature value of 0.7 for uncertainty estimation, as used in [14].
- A limited number of manually crafted examples are utilized to enhance answer prediction during the uncertainty estimation phase. These annotated examples are sourced directly from [14], which they refer to as a "few-shot prompting trick", aimed at improving prediction reliability.
- In Active-CoT, as proposed by [14], human input is involved in annotating selected questions, with an expert annotator guiding logical thought processes. The focus is primarily on selecting appropriate examples, with less emphasis on intensive manual annotation. Although they explored the impacts of different annotators and the distinct effects of example selection and annotation, their results demonstrate that the annotators A and B consistently outperform baseline models. In addition to their primary annotator (annotator A), they also incorporate human-annotated explanations from the GSM8K dataset (annotator B). My thesis makes use of the annotations provided from the GSM8k dataset and therefore will be compared to the results of annotator B by [14].

While my approach aligns with [14] on certain hyperparameters, there are differences:

- While [14] uses a number of trails equal to 10, I use a number of trails equal to 5 to avoid high computational costs. A larger number of trails may lead to more precise estimation.
- Inference in [14] involves setting the temperature to 0.7 and making 40 attempts for each question to select the most consistent answer, known as self-consistency [62]. Active-CoT is combined with self-consistency in their approach. In contrast, my approach queries the LLM only once with a temperature of 0.0 during inference, thus not applying the self-consistency method. In all the few-shot prompting methods in my thesis, I do not use self-consistency during inference to ensure a fair comparison.

For methods with multiple hyperparameters, such as Diverse-Active-KMeansPlusPlus-Retrieval-CoT, I conduct hyperparameter tuning by exploring 3-5 combinations and report the best accuracy. In the case of Random-CoT, four different seeds were used for random in-context example sampling, and the average accuracy was reported.

It's important to note the dataset sizes in the evaluation. The GSM8K test set consists of 1,319 examples, while AQUA contains only 254. For instance, a 2% performance improvement on GSM8K corresponds to roughly 26 more examples correctly classified, whereas on AQUA, it amounts to just 5 examples.

4.2 RESULTS

In the sections that follow, I provide a summary of all the results and accuracies. Table 4.1 showcases the performance of CoT and standard prompting methods with human-generated labels, while Table 4.2 demonstrates the performance of these methods using LLM-generated labels with Zero-Shot-CoT.

4.2.1 *Comparison of Prompting Methods*

4.2.1.1 *Few-shot with GPT-3.5-turbo vs Zero-Shot-CoT with GPT-3.5-turbo*

I conducted an extensive evaluation of model performance in a few-shot setting across various scenarios and selection methods, comparing the performance of few-shot approaches on two datasets: GSM8K and AQUA against Zero-Shot-CoT with GPT-3.5-turbo. The scenarios I explored encompassed both CoT and Standard approaches, incorporating human-generated labels and labels generated by Zero-Shot-CoT using GPT-3.5-turbo.

1. **CoT approaches with human-generated labels:** In the first scenario, where CoT methods incorporated human-generated data, the few-shot-CoT methods demonstrated remarkable performance.

   GSM8K: With an average accuracy of 80.8%, they significantly outperformed the Zero-Shot-CoT approach on the GSM8K dataset, surpassing it by 11.4%.

   AQUA: Similarly, on the AQUA dataset, the few-shot-CoT methods achieved an average accuracy of 60%, outperforming the Zero-Shot-CoT approach (54.7%) by 5.3%.
   [43]

2. **CoT approaches with LLM-generated labels:** In the second scenario, I examined CoT approaches that leveraged Zero-Shot-CoT generated data with GPT-3.5-turbo.

   GSM8K: Few-shot-CoT prompting methods excelled with an average accuracy of 81% on the GSM8K dataset, surpassing the Zero-Shot-CoT approach by 11.6%.

   AQUA: On the AQUA dataset, few-shot-CoT approaches achieved an average accuracy of 56.7%, demonstrating a 2% improvement over the Zero-Shot-CoT approach (54.7%).
   [43]

3. **Standard approaches with human-generated labels:** In this scenario, I assessed standard approaches using human-generated labels.

   GSM8K: In this case, few-shot standard methods yielded an average accuracy of 71.1% on the GSM8K dataset, outperforming the Zero-Shot-CoT approach by 1.7%.

AQUA: Surprisingly, on AQUA Zero-Shot-CoT outperforms the few-shot standard methods (average 52.5%) by 2.2%.
[43]

4. **Standard approaches with LLM-generated labels:** Finally, I examined standard approaches using labels generated by Zero-Shot-CoT with GPT-3.5-turbo.

   GSM8K: The few-shot standard methods achieved an average accuracy of 70.6% on the GSM8K dataset. This performance slightly outperforms the Zero-Shot-CoT approach by a small margin of 1.2%.

   AQUA: However, when evaluated on the AQUA dataset, standard approaches yielded an average accuracy of 51.5%. In this case, the standard approaches were underperformed by the Zero-Shot-CoT approach, which achieved an accuracy of 54.7%, surpassing the standard methods by 3.2%.
   [43]

To conclude, few-shot methods have consistently demonstrated their effectiveness, outperforming Zero-Shot-CoT with GPT-3.5-turbo with a significant margin in different scenarios. However, it's important to note that on these datasets, Zero-Shot-CoT with GPT-3.5-turbo performs comparably with few-shot standard approaches only when labels are generated by Zero-Shot-CoT. These results highlight the potential of few-shot methods to enhance model performance across a range of contexts.

4.2.1.2 *Few-shot with GPT-3.5-turbo vs Zero-Shot-CoT with GPT-4*

In a similar fashion, I evaluate the average performance of various few-shot prompting methods across different scenarios when compared to Zero-Shot-CoT, utilizing GPT-4.

1. **CoT approaches with human-generated labels:**

   GSM8K: On the GSM8K dataset, Zero-Shot-CoT with GPT-4 achieved an accuracy of 87.6%, surpassing few-shot chain-of-thought methods with human-generated labels by 6.8%.

   AQUA: On the AQUA dataset, Zero-Shot-CoT with GPT-4 demonstrated an accuracy of 68.9%, outperforming few-shot methods by 8.9%.
   [43]

2. **CoT approaches with LLM-generated labels:**

   GSM8K: It outperformed few-shot chain-of-thought methods by 6.6% on the GSM8K dataset.

   AQUA: Impressively, on the AQUA dataset, it exhibited an even more significant advantage, surpassing the performance of few-shot meth-

ods by a remarkable 12.2%.
[43]

3. **Standard approaches with human-generated labels:**

   GSM8K: Zero-Shot-CoT with GPT-4 exhibited significant advantages over few-shot standard methods with human-generated labels, outperforming them by 16.5% on GSM8K.

   AQUA: On the AQUA dataset, it displayed a similar exceptional performance, surpassing other methods by 16.4%.
   [43]

4. **Standard approaches with LLM-generated labels:**

   GSM8K: For the GSM8K dataset, standard approaches with Zero-Shot-CoT-generated labels achieved an average accuracy of 70.6, underperforming Zero-Shot-CoT with GPT-4 by a signifcant margin of 17%.

   AQUA: On the AQUA dataset, standard approaches achieved an average accuracy of 51.5, which was also outperformed by Zero-Shot-CoT with GPT-4, by a significant margin of 17.4%.
   [43]

Based on these experimental results, it is clear that the GPT-4 model, when employed in a Zero-Shot-CoT setting, consistently outperforms few-shot prompting methods across various scenarios. These results demonstrate that GPT-4 exhibits superior capabilities in reasoning tasks compared to its predecessor, GPT-3.5-turbo. One reason behind the exceptional performance of GPT-4, particularly on the GSM8K dataset, could be attributed to the fact that OpenAI included a portion of the training set in the GPT-4 pre-training mix [44]. This approach might have contributed to the remarkable results observed in the evaluation.

4.2.1.3 *Few-Shot CoT vs Few-shot Standard*

I conducted an evaluation to compare the performance of few-shot CoT prompting methods with that of few-shot standard prompting methods across various scenarios. These scenarios included labels generated by humans and labels generated using Zero-Shot-CoT with GPT-3.5-turbo.

1. **Human-generated labels:**

   GSM8K: On the GSM8K dataset, they achieved 80.8% accuracy, surpassing the 71.1% accuracy of standard methods by 9.7%.

   AQUA: Similarly, on the AQUA dataset, few-shot CoT methods exhibited a 7.5% accuracy improvement over few-shot standard methods.
   [43]

2. **LLM-generated labels:**

GSM8K: With LLM-generated labels, few-shot CoT methods achieved an average accuracy of 81% on GSM8K, outperforming few-shot standard methods at 70.6% by 10.4%.

AQUA: Similarly, on the AQUA dataset, few-shot CoT methods exhibited a 5.2% accuracy improvement over few-shot standard methods. [43]

In conclusion, this evaluation strongly supports the use of few-shot chain-of-thought methods for reasoning tasks. The consistent performance improvement across scenarios, including both human- and LLM-generated labels, underscores the value of incorporating the reasoning chains in the in-context demonstrations. It's worth noting that even when utilizing GPT-3.5-turbo for labeling, few-shot CoT methods still enhance performance.

4.2.1.4 *Few-shot prompting methods vs Random*

In this section, I present the evaluation results of several few-shot prompting methods designed to assess the performance of GPT-3.5-Turbo in a few-shot setting. I employed various prompting techniques, including those incorporating reasoning chains (chain-of-thought approaches) and standard approaches, to explore their effectiveness in reasoning tasks. Furthermore, I conducted this evaluation under two distinct labeling scenarios: human-generated data, considered the ground truth, and labeling with Zero-Shot-CoT using GPT-3.5-turbo. This comprehensive approach provides a broader and more robust evaluation under different conditions. The performance of these few-shot methods is compared to Random approach.

Evaluation Results for Few-shot Chain-of-Thought Methods

I begin by presenting the evaluation results of few-shot Chain-of-Thought methods.

Human-generated labels:

GSM8K: On the GSM8K dataset, I observe that none of the prompting methods manages to outperform the baseline, Random-CoT. The accuracy achieved by the various methods is quite similar to that of Random-CoT, with only a small margin of difference, typically around 1-1.5%. This suggests that, for the GSM8K dataset, the performance of these prompting methods is on par with the Random-CoT baseline.

AQUA: However, on the AQUA dataset, we see a slightly different trend. Active-CoT manages to outperform Random-CoT, but the margin of improvement is relatively small, approximately 1.6%. Similarly, Diverse-CoT also outperforms Random-CoT with a smaller margin of around 1.1%.

In summary, for the GSM8K dataset, the prompting methods exhibit performance similar to Random-CoT, while on AQUA, we observe

slight improvements with Active-CoT and Diverse-CoT. The overall effect, however, is minimal, and the performance across these methods remains comparable to the Random-CoT baseline.
[43]

LLM-generated labels:

GSM8K: In the evaluation results on the GSM8K dataset, Diverse-CoT outperforms Random-CoT by a small margin of 1.1%, and Retrieval-CoT by 1.7%, and Diverse-Active-KMeansPlusPlus-CoT by 1.9%. Only Diverse-Active-KMeans-CoT, with an accuracy of 79.2%, underperforms Random-CoT by a slight margin of 0.3%. Meanwhile, methods like Active-CoT, with an accuracy of 83.0%, outperform Random-CoT by the largest margin of 3.5%, and Diverse-Active-KMeansPlusPlus-Retrieval-CoT, with an accuracy of 82%, outperforms by 2.5%.

AQUA: In the evaluation results on the AQUA dataset, Diverse-CoT, Active-CoT, and Diverse-Active-Retrieval-CoT outperformed Random-CoT by margins of 3.9%, 2.7%, and 3.3%, respectively. Retrieval-CoT demonstrated comparable performance with Random-CoT, with a slight underperformance by 0.7%. However, Diverse-Active-KMeansPlusPlus-CoT underperformed Random-CoT by a margin of 2.4%.
[43]

In conclusion, when utilizing labels generated by Zero-Shot-CoT with GPT-3.5-turbo, methods such as Active-CoT, Diverse-CoT and Diverse-Active-Retrieval-CoT exhibit a significantly performance advantage over Random-CoT. On the other hand, when working with labels manually labeled by humans, performance remains comparable to that of Random-CoT. Diverse-Active-KMeans-CoT methods appear to consistently underperform Random-CoT in both labeling scenarios and across both datasets. Additionally, Retrieval-CoT demonstrates comparable or even superior performance when compared to Random-CoT.

Evaluation Results for Few-shot Standard Methods

In the following, I assess the effectiveness of three few-shot standard approaches: Diverse-Standard, Active-Standard, and Retrieval-Standard, in comparison to Random-Standard.

Human-generated labels:

GSM8K: On the GSM8K dataset, Random-Standard demonstrates an average accuracy that outperforms the other prompting methods by an average margin of 1.6%.

AQUA: Meanwhile, on the AQUA dataset, Random-Standard again leads, but with a smaller margin of 0.3%.
[43]

LLM-generated labels:

GSM8K: In the evaluation with LLM-generated labels on the GSM8K dataset, Random-Standard remains the top performer with a margin of 0.7%.

AQUA: However, on the AQUA dataset, Random-Standard underperforms with a small margin of 0.4%.
[43]

Contrary to the results obtained with Few-shot Chain-of-Thought prompting methods, it is observed that Random-Standard achieves either comparable or superior performance under both labeling scenarios compared to the other prompting methods. These results suggests that prompting methods excel over Random sampling only when incorporating reasoning chains in the in-context demonstrations or using labels generated by GPT-3.5-turbo with Zero-Shot-CoT.

4.2.1.5 *Human-generated labels vs LLM-generated labels*

I aim to compare the average performance of few-shot prompting methods with human-generated labels against those with LLM-generated labels.

- **GSM8K:** On the GSM8K dataset, both approaches demonstrate comparable performance, with an average accuracy of approximately 81%.
- **AQUA:** However, on the AQUA dataset, few-shot prompting with human-generated labels, boasting an average accuracy of 60%, outperforms few-shot prompting methods, that utilize Zero-Shot-CoT for labeling, by a margin of 3%.

In conclusion, it is evident that few-shot prompting methods, when using labels generated by humans, exhibit performance comparable or superior to the methods that use Zero-Shot-CoT with GPT-3.5-turbo for labeling, although with a small margin.

4.2.2 *Comparison with prior Work*

In this analysis, I assess the performance of the GPT-3.5-Turbo model in a few-shot context, comparing it to its predecessors within the GPT-3.5 family, as highlighted by [14], including models like text-davinci-003, text-davinci-002, and code-davinci-002, as GPT-3.5-turbo was not yet available. This evaluation covers various few-shot chain-of-thought prompting methods.

- **GSM8K:** When employing Active-CoT, GPT-3.5-Turbo achieves an accuracy of 83%, clearly outperforming text-davinci-003, which attains an accuracy of 65.6%, with a significant margin of 17.4%. This difference in performance also holds true when comparing GPT-3.5-Turbo with Random-CoT, where it surpasses text-davinci-003 by an even larger margin of 20%. Moreover, GPT-3.5-Turbo outperforms text-davinci-002 by 17.8% when using Random-CoT, and by 11.9% with Active-CoT. It's worth noting that, despite the utilization of inference self-consistency to enhance few-shot performance for text-davinci-003 and text-davinci-002 [62], GPT-3.5-Turbo consistently demonstrates superior performance in a few-shot scenario.

- **AQUA:** For the AQUA dataset, GPT-3.5-turbo with Random-CoT demonstrates an accuracy of 60.7%, clearly outperforming text-davinci-002 with an accuracy of 44.1% and code-davinci-002, which achieves an accuracy of 53.1%, by significant margins of 16.6% and 7.6%, respectively. Furthermore, GPT-3.5-turbo with Active-CoT achieves an accuracy of 62.3%, surpassing text-davinci-003 and text-davinci-002 by significant margins of 14.3% and 12%.

In conclusion, it's evident that GPT-3.5-Turbo, even without the use of the self-consistency method, which has been demonstrated to further improve the performance of LLM [62], excels over older models from the GPT-3.5 family in few-shot settings for reasoning tasks, despite being smaller in size.

4.2.2.1 *Influence of uncertainty on accuracy*

I conducted an analysis to examine the influence of uncertainty, as reflected by disagreement values, on the accuracy of few-shot prompting. I initially selected 8 in-context demonstrations with a disagreement value of 1, representing cases where the model's predictions were relatively certain. These 8 demonstrations were used as context for running the GPT-3.5-turbo model, and the accuracy is reported. I then repeated the process, but this time, I selected in-context demonstrations with disagreement values of 3 and 5. Disagreement values of 5 represent instances where the model's predictions are associated with the highest level of uncertainty. For the AQUA dataset, I used 4 in-context demonstrations and followed a similar procedure.

- **Human-generated labels:**

  GSM8K: In the GSM8K dataset, an increase of around 1% in accuracy was observed when transitioning from a disagreement level of 1 to 3.

  AQUA: However, for the AQUA dataset, a more significant impact of uncertainty is observed. As the disagreement values increase, the model's performance improves, with the highest accuracy achieved for examples with a disagreement value of 5. The improvement in accuracy from a disagreement level of 1 to 5 is significant, with a 6.7% increase

- **LLM-generated labels:**

  GSM8K: For GSM8K, beginning with a disagreement of 1, the accuracy is 79.7. As we progress to a disagreement of 3, the accuracy improves to 80.3, and further increasing the disagreement to 5 results in an accuracy of 83.0.

AQUA: The AQUA dataset demonstrates a similar trend. For disagreement of 1, the accuracy is 55.1, and when we raise the disagreement to 3, the accuracy increases to 59.4. However, for disagreement 5, the accuracy slightly decreases, reaching 57.5.

These results are visualized in Figure 4.1. In some cases, transitioning from a disagreement level of 3 to 5 results in a slight decrease in accuracy. This phenomenon might be due to the relatively low number of trails conducted in this thesis, which was set at 5. Increasing the number of trails for uncertainty estimation may yield more accurate results in the future.

In summary, these findings suggest that choosing examples characterized by high model uncertainty can lead to improvements in model accuracy, and this effect is particularly evident under different labeling scenarios.

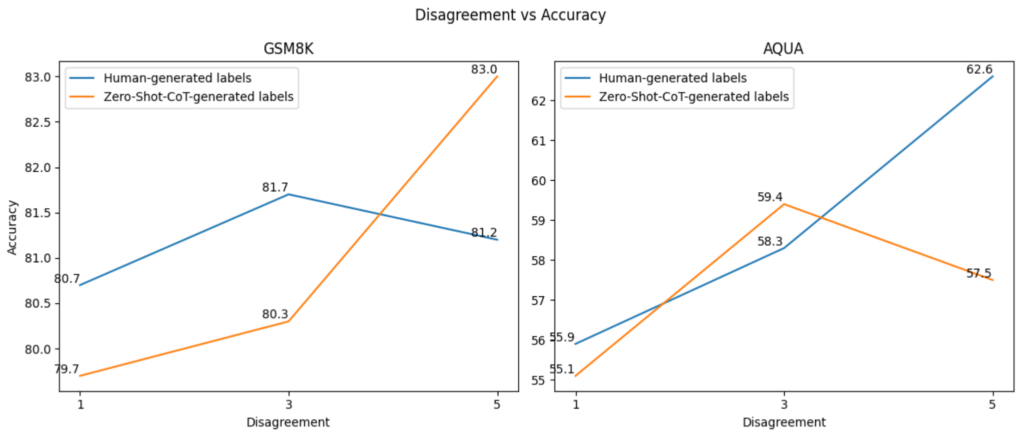

Figure 4.1: Disagreement vs Accuracy

4.2.3 *Falcon Models in few-shot Scenario*

I conducted an evaluation of the performance of open-source models, Falcon-40B-Instruct and Falcon-7B-Instruct, in a few-shot setting. The choice of using Diverse-CoT for selecting in-context demonstrations was made to gain insights into the general performance of these models in a few-shot scenario.

- **GSM8K:** On the GSM8K dataset, Falcon-40B-Instruct achieved an accuracy of 37.3%, significantly outperforming Falcon-7B-Instruct, which reached an accuracy of 5.4%. This difference of 31.9% underscores the superiority of Falcon-40B-Instruct in this setting.

- **AQUA:** Similarly, on the AQUA dataset, Falcon-40B-Instruct demonstrated better performance, surpassing Falcon-7B-Instruct by a smaller margin of 7.1%.

The results corresponding to this analysis are depicted in Table 4.3. On average, it was observed that employing the larger model version led to a

Table 4.1: Results of CoT and standard prompting method with human-generated labels

Method/Dataset	GSM8K	AQUA	**Average**
Baselines			
GPT-4 Zero-Shot-CoT	87.6	68.9	78.3
GPT-3.5-turbo Zero-Shot-CoT	69.4	54.7	62.1
CoT Methods			
Random-CoT	81.7	60.7	71.2
Diverse-CoT	80.2	61.8	71.0
Active-CoT	81.2	62.3	71.8
Retrieval-CoT	80.7	57.5	69.1
Diverse-Active-KMeans-CoT	80.4	60.2	70.3
Diverse-Active-KMeansPlusPlus-CoT	81.1	60.6	69.5
Diverse-Active-KMeansPlusPlus-Retrieval-CoT	80.4	59.6	70.0
Average	80.8	60.0	70.4
Standard Methods			
Random-Standard	72.7	52.8	62.8
Diverse-Standard	71.5	53.1	62.3
Active-Standard	70.4	50.4	60.4
Retrieval-Standard	69.8	53.8	61.8
Average	71.1	52.5	61.8

19.5% improvement in accuracy. This finding suggests that bigger models excel in few-shot prompting for reasoning tasks. This observation aligns with the findings presented in the study conducted by [6], which also highlights the superior performance of larger models in various few-shot settings.

In few-shot scenarios, Falcon-40B-Instruct and Falcon-7B-Instruct demonstrate inferior performance compared to GPT-3.5-turbo, with margins exceeding 30%. However, it's worth mentioning that, while I used the same instructions in the prompt as for GPT-3.5-Turbo, open-source models such as Falcon-40B-Instruct and Falcon-7B-Instruct may require additional prompt formatting before passing input to the LLM. The way the prompts were set up might have affected the results.

Table 4.2: Results of CoT and standard prompting method with LLM-generated labels with Zero-Shot-CoT

Method/Dataset	GSM8K	AQUA	**Average**
Baselines			
GPT-4 Zero-Shot-CoT	87.6	68.9	78.3
GPT-3.5-turbo Zero-Shot-CoT	69.4	54.7	62.1
CoT Methods			
Random-CoT	79.5	56.7	68.1
Diverse-CoT	80.6	60.6	70.6
Active-CoT	83.0	59.4	70.3
Retrieval-CoT	81.2	56.0	68.6
Diverse-Active-KMeans-CoT	79.2	52.0	65.6
Diverse-Active-KMeansPlusPlus-CoT	81.4	54.3	67.9
Diverse-Active-KMeansPlusPlus-Retrieval-CoT	82.0	60.0	71
Average	81.0	57.0	68.9
Standard Methods			
Random-Standard	71.3	50.0	60.7
Diverse-Standard	70.1	51.6	60.9
Active-Standard	70.7	52.0	61.4
Retrieval-Standard	70.3	52.2	61.3
Average	70.6	51.5	61.1

Table 4.3: Falcon models with Diverse-CoT

Model/Dataset	GSM8K	AQUA	**Average**
Falcon-40B-Instruct	37.3	18.5	27.9
Falcon-7B-Instruct	5.4	11.4	8.4

5 CONCLUSION AND FUTURE WORK

This thesis primarily focuses on assessing various prompting methods, such as Random-CoT, Active-CoT, Retrieval-CoT, and Auto-CoT, using different models and arithmetic reasoning datasets. I showcased the efficacy and superior performance of few-shot chain-of-thought methods like Active-CoT, Auto-CoT, and Retrieval-CoT over the Random baseline when employing labels generated by GPT-3.5-turbo under Zero-Shot-CoT. However, when using human-generated labels, the performance levels are comparable.

Moreover, I've highlighted how GPT-4, under Zero-Shot-CoT, outperforms GPT-3.5-turbo significantly in a few-shot setting across diverse datasets and labeling scenarios. I've delved into the performance analysis of open-source LLMs like Falcon-40B-Instruct and Falcon-7B-Instruct with Diverse-CoT, demonstrating that larger models enhance few-shot performance.

Additionally, I proposed new methods like Diverse-Active-KMeans-CoT and Diverse-Active-KMeansPlusPlus-CoT, combining diversity and uncertainty, which yielded performance comparable to the Random baseline. Furthermore, I proposed a prompting method Diverse-Active-KMeansPlusPlus-Retrieval-CoT that combines diversity, uncertainty, and retrieval component. The results demonstrate that the latter method surpasses the Random baseline by a significant margin when utilizing LLM-generated labels.

This thesis has set the stage for additional research in the field of few-shot prompting methods with LLMs, particularly for reasoning tasks. Here are some key directions for future exploration:

- **Fine-Tuning Open-Source LLMs :** Future work could involve fine-tuning open-source LLMs on datasets such as GSM8K and AQUA, especially smaller models. Innovative fine-tuning methods like LoRA and QLoRA can be applied to adapt these models to reasoning tasks and then compared to the performance of more advanced models like GPT-3.5-Turbo and GPT-4.

- **Scaling Up with More Training Data :** Expanding the research by using larger training datasets can offer a deeper understanding of few-shot performance. With more training data, the effectiveness of certain few-shot prompting methods may become more evident. For example, Active-CoT may benefit from a dataset that has examples where the model is less confident. Thus, experiments with larger training datasets can uncover new insights into few-shot performance.

- **Ensemble Methods :** Future research could explore the use of ensemble techniques, combining the strengths of multiple few-shot prompting methods to potentially enhance performance. This direction involves investigating how different approaches can be merged to improve overall performance in reasoning tasks. Considering the insights derived from this thesis, which emphasize the influence of diversity, uncertainty, and retrieval-based methods on few-shot performance, there is a need for developing a unified approach with fewer hyperparameters. This stands in contrast to the proposed Diverse-Active-KMeansPlusPlus-Retrieval-CoT, which, due to its optimization of the parameter p, presents a drawback that impacts few-shot performance.

BIBLIOGRAPHY

[1] Ebtesam Almazrouei et al. "Falcon-40B: an open large language model with state-of-the-art performance." In: (2023). URL: https://huggingface.co/tiiuae/falcon-40b-instruct.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[3] Aram Bahrini, Mohammadsadra Khamoshifar, Hossein Abbasimehr, Robert J. Riggs, Maryam Esmaeili, Rastin Mastali Majdabadkohne, and Morteza Pasehvar. *ChatGPT: Applications, Opportunities, and Threats*. 2023. arXiv: 2304.09103 [cs.CY].

[4] Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. *Open LLM Leaderboard*. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard. 2023.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. "A Neural Probabilistic Language Model." In: *J. Mach. Learn. Res.* 3 (2003), pp. 1137–1155. URL: https://api.semanticscholar.org/CorpusID:221275765.

[6] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. *Generating Long Sequences with Sparse Transformers*. 2019. arXiv: 1904.10509 [cs.LG].

[8] Karl Cobbe et al. *Training Verifiers to Solve Math Word Problems*. 2021. arXiv: 2110.14168 [cs.LG].

[9] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. *Active Learning with Statistical Models*. 1996. arXiv: cs/9603104 [cs.AI].

[10] DeepLearning.AI. *Generative AI with Large Language Models*. https://www.coursera.org/learn/generative-ai-with-llms/lecture/R0xbD/generating-text-with-transformers. 2023.

[11] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 [cs.LG].

[12] Tim Dettmers and Luke Zettlemoyer. "The case for 4-bit precision: k-bit Inference Scaling Laws." In: *Proceedings of the 40th International Conference on Machine Learning*. Ed. by Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett. Vol. 202. Proceedings of Machine Learning Research. PMLR, 2023, pp. 7750–7774. URL: https://proceedings.mlr.press/v202/dettmers23a.html.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[14] Shizhe Diao, Pengcheng Wang, Yong Lin, and Tong Zhang. *Active Prompting with Chain-of-Thought for Large Language Models*. 2023. arXiv: 2302.12246 [cs.CL].

[15] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. *A Survey on In-context Learning*. 2023. arXiv: 2301.00234 [cs.CL].

[16] Jakob Gawlikowski et al. *A Survey of Uncertainty in Deep Neural Networks*. 2022. arXiv: 2107.03342 [cs.LG].

[17] Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A. Smith. *RealToxicityPrompts: Evaluating Neural Toxic Degeneration in Language Models*. 2020. arXiv: 2009.11462 [cs.CL].

[18] Claudio Gentile, Zhilei Wang, and Tong Zhang. *Fast Rates in Pool-Based Batch Active Learning*. 2022. arXiv: 2202.05448 [cs.LG].

[19] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. *Transformer in Transformer*. 2021. arXiv: 2103.00112 [cs.CV].

[20] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. *Measuring Mathematical Problem Solving With the MATH Dataset*. 2021. arXiv: 2103.03874 [cs.LG].

[21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL].

[22] Hugging Face. *Text Generation Inference*. https://huggingface.co/docs/text-generation-inference/index.

[23] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. *Large Language Models are Zero-Shot Reasoners*. 2023. arXiv: 2205.11916 [cs.CL].

[24] Tom Kwiatkowski et al. "Natural Questions: A Benchmark for Question Answering Research." In: *Transactions of the Association for Computational Linguistics* 7 (2019). Ed. by Lillian Lee, Mark Johnson, Brian Roark, and Ani Nenkova, pp. 452–466. DOI: 10.1162/tacl_a_00276. URL: https://aclanthology.org/Q19-1026.

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. 2023. arXiv: 2309.06180 [cs.LG].

[26] Abdullatif Köksal, Timo Schick, and Hinrich Schütze. *MEAL: Stable and Active Learning for Few-Shot Prompting*. 2023. arXiv: 2211.08358 [cs.CL].

[27] Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. *Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning*. 2023. arXiv: 2303.15647 [cs.CL].

[28] Stephanie Lin, Jacob Hilton, and Owain Evans. *TruthfulQA: Measuring How Models Mimic Human Falsehoods*. 2022. arXiv: 2109.07958 [cs.CL].

[29] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. *A Survey of Transformers*. 2021. arXiv: 2106.04554 [cs.LG].

[30] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. *Program Induction by Rationale Generation : Learning to Solve and Explain Algebraic Word Problems*. 2017. arXiv: 1705.04146 [cs.AI].

[31] Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. *Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning*. 2022. arXiv: 2205.05638 [cs.LG].

[32] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. *What Makes Good In-Context Examples for GPT-3?* 2021. arXiv: 2101.06804 [cs.CL].

[33] Kai Lv, Yuqing Yang, Tengxiao Liu, Qinghui Gao, Qipeng Guo, and Xipeng Qiu. *Full Parameter Fine-tuning for Large Language Models with Limited Resources*. 2023. arXiv: 2306.09782 [cs.CL].

[34] Tomas Mikolov, Martin Karafiát, Luká Burget, Jan Honza ernocký, and Sanjeev Khudanpur. "Recurrent neural network based language model." In: *Interspeech*. 2010. URL: https://api.semanticscholar.org/CorpusID:17048224.

[35] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. *Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?* 2022. arXiv: 2202.12837 [cs.CL].

[36] Akihiro Nakamura and Tatsuya Harada. *Revisiting Fine-tuning for Few-shot Learning*. 2019. arXiv: 1910.00216 [cs.LG].

[37] Nikita Nangia, Clara Vania, Rasika Bhalerao, and Samuel R. Bowman. *CrowS-Pairs: A Challenge Dataset for Measuring Social Biases in Masked Language Models*. 2020. arXiv: 2010.00133 [cs.CL].

[38] Andrew Ng. *ChatGPT Prompt Engineering for Developers*. DeepLearning.AI. 2023. URL: https://learn.deeplearning.ai/chatgpt-prompt-eng/lesson/1/introduction.

[39] Richard Nock, Raphaël Canyasse, Roksana Boreli, and Frank Nielsen. *k-variates++: more pluses in the k-means++*. 2016. arXiv: 1602.01198 [cs.LG].

[40] OpenAI. *Models*. https://platform.openai.com/docs/models. Accessed: November 5, 2023.

[41] OpenAI. *Introducing ChatGPT*. https://openai.com/blog/chatgpt. Nov. 2022.

[42] OpenAI. *Aligning language models to follow instructions*. https://openai.com/research/instruction-following. Accessed: November 9, 2023. 2023.

[43] OpenAI. *ChatGPT*. https://chat.openai.com/chat. 2023.

[44] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[45] OpenAI. *GPT-4*. https://openai.com/research/gpt-4. Mar. 2023.

[46] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155 [cs.CL].

[47] Arkil Patel, Satwik Bhattamishra, and Navin Goyal. *Are NLP Models really able to Solve Simple Math Word Problems?* 2021. arXiv: 2103.07191 [cs.CL].

[48] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. *The RefinedWeb Dataset for Falcon LLM: Outperforming Curated Corpora with Web Data, and Web Data Only*. 2023. arXiv: 2306.01116 [cs.CL].

[49] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training." In: 2018. URL: https://api.semanticscholar.org/CorpusID:49313245.

[50] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners." In: 2019. URL: https://api.semanticscholar.org/CorpusID:160025533.

[51] Nils Reimers and Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. 2019. arXiv: 1908.10084 [cs.CL].

[52] Herbert E. Robbins. "A Stochastic Approximation Method." In: *Annals of Mathematical Statistics* 22 (1951), pp. 400–407. URL: https://api.semanticscholar.org/CorpusID:16945044.

[53] Guy Rotman and Roi Reichart. "Multi-task Active Learning for Pre-trained Transformer-based Models." In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 1209–1228. DOI: 10.1162/tacl_a_00515. URL: https://aclanthology.org/2022.tacl-1.70.

[54] Rachel Rudinger, Jason Naradowsky, Brian Leonard, and Benjamin Van Durme. *Gender Bias in Coreference Resolution*. 2018. arXiv: 1804.09301 [cs.CL].

[55] Christopher Schröder, Andreas Niekler, and Martin Potthast. *Revisiting Uncertainty-based Query Strategies for Active Learning with Transformers*. 2022. arXiv: 2107.05687 [cs.CL].

[56] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[57] Burr Settles. "Active Learning Literature Survey." In: 2009. URL: https://api.semanticscholar.org/CorpusID:324600.

[58] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. 2019. arXiv: 1909.09586 [cs.NE].

[59] Beatriz Stollnitz. *The Transformer Architecture of GPT Models*. Towards-DataScience. 2023. URL: https://towardsdatascience.com/the-transformer-architecture-of-gpt-models-b8695b48728b.

[60] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[62] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. arXiv: 2203.11171 [cs.CL].

[63] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. 2023. arXiv: 2201.11903 [cs.CL].

[64] Joseph Weizenbaum. "ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine." In: *Commun. ACM* 9.1 (1966), 36–45. ISSN: 0001-0782. DOI: 10.1145/365153.365168. URL: https://doi.org/10.1145/365153.365168.

[65] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing." In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. URL: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[66] Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. *Automatic Chain of Thought Prompting in Large Language Models*. 2022. arXiv: 2210.03493 [cs.CL].

[67] Wayne Xin Zhao et al. *A Survey of Large Language Models*. 2023. arXiv: 2303.18223 [cs.CL].

[68] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. "Calibrate Before Use: Improving Few-shot Performance of Language Models." In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 12697–12706. URL: https://proceedings.mlr.press/v139/zhao21c.html.

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