Iterative Large Language Models Evolution through Self-Critique

by

Qianxi Li

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Training large language models (LLMs) often requires extensive human supervision and struggles with modeling long-range text semantic dependencies. To address these challenges, we introduce our framework **ELITE** — **E**volving **L**anguage models **I**teratively **T**hrough self-critiqu**E** — inspired by human learning processes such as self-critique and experience-based refinement. **ELITE** allows an LLM to autonomously generate, refine, and iteratively improve self-critiques ability by learning a mapping from questions-and-answers to critique labels through supervised fine-tuning (SFT). This approach drastically reduces the amount of human annotations from 2550 to 98 demonstrations by using a small set of prompt examples for initial configuration. Experimental results demonstrate that **ELITE** significantly outperforms an existing self-evolution baseline by 11% and SFT baselines by 4.8% on in-domain tasks. It also shows a performance improvement of 13.8%, 11.2%, and 7.2% on three out-of-domain data sets (SQuAD, BoolQ, and GSM8K), demonstrating generalization. The **ELITE** training framework can thus enable more adaptive, intelligent LLM systems that can improve themselves with relatively little human assistance.

Preface

This thesis is an original work by Qianxi Li. The core idea of this paper — using Large Language Models (LLMs) generated natural language critiques for fine-tuning LLMs — is introduced in my paper "LaFFi: Leveraging Hybrid Natural Language Feedback for Fine-tuning Language Models", which was accepted in the Human-Centric Representation Learning Workshop at the 38th Annual AAAI Conference on Artificial Intelligence.

Acknowledgements

First and foremost, I express my heartfelt gratitude to my supervisor, Professor Matthew E. Taylor, for his insightful guidance and unwavering support over the past two years of my graduate studies. This journey has been long and challenging, but Matt's mentorship has made the process significantly easier, more engaging, and immensely rewarding.

I also want to acknowledge Noah's Ark Laboratory, where the initial ideas for this thesis were conceived and preliminary experiments were conducted to validate these concepts. I extend my deepest thanks to my co-supervisor, Professor Jun Jin, whose guidance in exploring the literature on large language models and human-in-the-loop methods was invaluable, and who provided exceptional support during my internship.

I am profoundly grateful for the generous assistance and friendship from my friends at the Intelligent Robot Learning Lab (IRLL) and the Alberta Machine Intelligence Institute (Amii). I have gained tremendous knowledge and insights from all of you.

Lastly, I extend my deepest appreciation to my family for their constant support and encouragement. Without their love and unwavering support, I would not have been able to reach this far.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Transformer [1]-based large language models (LLMs), such as GPT-4 [2], LLaMA 2 [3], and Qwen [4], have shown remarkable success in complex language understanding tasks across various fields, including healthcare [5], finance [6], programming [7], and so on. These models excel due to their ability to leverage vast amounts of training data and the powerful representational ability of the Transformer architecture, which capture intricate patterns and relationships in language. At the same time, these models undergo massive self-supervised pre-training to capture general grammars, contexts, and semantics of human language, followed by supervised fine-tuning (SFT) to tackle more complex tasks requiring deeper linguistic comprehension. However, as these training paradigms near a performance plateau due to the exhaustive use of available data and increased challenges in modeling long-range complex text semantic dependencies, innovative training methodologies have become increasingly important for building more powerful LLMs.

The core problem we consider is: **Can an LLM improve its reasoning ability with self-generated critiques?**

Reasoning ability is important for LLMs because it enables them to generate coherent, accurate, and contextually appropriate responses, which is crucial for their effective application in complex and high-stakes environments. Also, by understanding how LLM self-improve itself, we are able to create more intelligent, adaptable, and reliable AI systems that can continuously improve their

performance and reliability without constant human oversight.

Inspired by human self-critique processes where individuals evaluate and refine their actions to optimize outcomes, we introduce **ELITE** (**E**volving **L**anguage models **I**teratively **T**hrough self-critiqu**E**) framework, as illustrated in Figure 3.1. In this context, we define self-critique as the process by which the LLM assesses the appropriateness of its own responses to given questions, providing explanations and improvements based on internal feedback mechanisms. Our framework **ELITE** employs an iterative learning process where LLMs generate multiple self-critiques, refine them, and reinforce the question-answer-critique mappings through supervised fine-tuning. By optimizing prompt examples in each iteration to better represent the self-critique dataset, the LLM progressively generates higher-quality self-critiques, improving task performance. The iterative process should run a few times to make sure that the prompt examples are diverse enough to represent the fine-tuning dataset, as shown in Figure 4.5. We demonstrate **ELITE**'s superiority over an existing self-evolution baseline, LMSI [8], and over SFT baselines through its performance on four natural language generation benchmarks, highlighting its excellent in-domain evaluations and generalization capabilities. The implementation of **ELITE** is publicly available at https://github.com/IRLL/LLM_self_improvement.

1.1 Novelty

Our novelties include:

- **Iterative Self-Critique Generation:** Unlike typical supervised fine-tuning methods that rely on static, human-annotated datasets, our approach leverages Large Language Models (LLMs) to iteratively generate self-critiques. These self-critiques serve as dynamic self-supervised fine-tuning labels, enhancing the self-critique ability of the LLM during training. This iterative process allows the model to continuously refine its understanding and improve its performance autonomously. Compared to SOTA methods, it requires less human annotations, and can be applied to various task types. The empirical performance of our framework

also outperforms two baselines: a strong supervised fine-tuning approach and LMSI, a self-improvement algorithm.

- **Refinement and Selection in 2D Text Representation Space:** Our framework refines and selects self-critiques and representative prompt examples in a reduced 2D text representation space. This process involves using PCA for dimensionality reduction and k-Means for clustering, ensuring that the most representative examples are chosen for subsequent iterations. This approach contrasts with SOTA methods, which often use static or manually-selected examples without leveraging a textual representation space for dynamic example optimization. By employing this technique, we reduce noise and enhance the quality of the training data, leading to more efficient and effective fine-tuning.

1.2 Contributions

Our contributions include:

- Proposing a novel framework that enables LLMs to self-evolve by iteratively reinforcing self-critique capabilities through supervised fine-tuning.
- Developing a novel dynamic prompt example selection approach and illustrating its effectiveness through an ablation study.
- Experimentally evaluating **ELITE**'s performance across different model scales and diverse Natural Language Processing (NLP) benchmarks.

The **ELITE** framework not only reduces the cost and resource requirements for improving LLMs but also enables them to self-evolve. As a general framework, it can be integrated with existing training methods and has the potential to be applied to foundation models across a wider range of modalities. **ELITE** paves the way for more adaptive and intelligent systems that can learn and optimize themselves through self-critique in ever-changing environments.

1.3 Thesis Outline

In Chapter 1, we introduce the problem we consider in this thesis, the challenges we find in literature, and the motivation. We also outline the contributions and novelties of this work.

In Chapter 2, we review the foundational concepts of Large Language Models (LLMs), including self-evolution and parameter-efficient methods. We also identifies gaps in the existing literature that the present work aims to address.

In Chapter 3, we introduce the ELITE framework. We detail the definitions and concepts essential to this approach. The chapter explains the iterative learning process, the dynamic selection of prompt examples, the generation of self-critiques using a majority voting approach, and the parameter-efficient fine-tuning method we used for LLMs.

In Chapter 4, we describe the experimental setup, datasets we used, and baseline methods we used for comparison. We present the core results of the ELITE framework, exploring the effects of self-training iterations and majority voting response quantities. Additionally, we examine the clustering and selection process of prompt examples through a qualitative analysis, concluding with an ablation study that demonstrates the impact of initial prompt examples and the selection process on the framework's performance.

Finally, in Chapter 5, we summarize the key findings, discussing how the ELITE framework enhances LLM performance through autonomous self-critique and refinement. We highlight the framework's potential impact. We also outline the limitations of the study, acknowledging the constraints of the experimental setup and suggesting directions for further investigation.

Chapter 2

Background

In this chapter, we first introduce the Transformer architecture and the encoder and decoder LLMs developed from it. We then provide an overview of the self-evolution paradigm for LLMs and discuss how critique information is used to enhance LLM quality within this framework. Next, we cover various parameter-efficient methods employed to make model training and inference more efficient. Finally, we conclude by summarizing the gaps in the LLM self-evolution literature.

2.1 The Transformer Architecture

The Transformer architecture [1], revolutionized the field of natural language processing (NLP) by enabling more efficient and scalable models for a wide range of tasks. Unlike traditional recurrent neural networks (RNNs) and long short-term memory networks (LSTMs) [9], the Transformer model can process input data in parallel, can scale to long sequences, and less likely suffer from the vanishing gradient problem, which makes the model training and inference more efficient and the representational ability stronger. Below we introduce the attention mechanism, the key designs of the Transformer architecture and its contributions to the field of Artificial Intelligence (AI).

2.1.1 The Attention Mechanism

The Transformer architecture is built upon the concept of self-attention, which allows the model to weigh the importance of different words in a sequence relative to each other. This mechanism enables the model to capture long-range dependencies more effectively than RNNs and LSTMs. The core component of the Transformer is the attention mechanism, which can be represented mathematically as:

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

$$Q = W_Q E, K = W_Q E, V = W_Q E \tag{2.2}$$

where Q (query), K (key), and V (value) are matrices mapped from the input embeddings matrix E by three learnable projection matrices W_Q, W_K, and W_V, and d_k is the dimension of the key vectors.

Below we explain the meanings of the four matrices mentioned above:

- Input embeddings (E): In the context of NLP, E has the shape (d_{input}, $d_{embedding}$), with d_{input} amount of tokens in the input sequence and each mapped into a vector with the length $d_{embedding}$. The mapping from the input sequence I to the input embeddings matrix E is done by a learnable projection matrix W_{embed}. Similar to W_Q, W_K, and W_V, these learnable matrices are first initialized randomly and they gradually learn to perform their designed functionality during training. E is designed to generate an embedding for each input tokens with the same size so that the subsequent module can operate on these vectors, extract complex relations and generate more complex representations.

- Query (Q): The Query matrix learns to represent the current input vector sequence for which we want to find relevant information from the other vectors in the sequence during training. Query acts as a search term when we do a dot product between Query and Key matrix, such that, for each vector in Query and Key matrix, their dot product mathematically measures

how close these two vectors similar to each other. A higher dot product value means a higher relevance level.

- Key (K): The Key matrix learns to better represent the current input vector sequence used as the keys for comparison during training. It is used to compare against the Query to see how much attention should be given to each input vector in the sequence.

- Value (V): The Value matrix also learns to represent the current input vector sequence during training, similar to the Key matrix. However, while the Key is used for comparison, the Value is used to store the actual information that will be combined (i.e., the weighted sum) based on the attention scores derived from the Query and Key comparison. The intuition is that, after the dot product between Query and Key matrices and weighted by the Softmax function, we know how much different key vectors contribute to each query vectors. But to generate more complex representations for each input vector, we need to aggregate information from the value vectors that are relevant to the query vectors with different weights.

2.1.2 Key Designs of the Transformer

The Transformer consists of an encoder and a decoder, each composed of multiple layers of self-attention and feed-forward neural networks. The encoder processes the input sequence to generate contextualized representations, while the decoder uses these representations to produce the output sequence. Each encoder layer comprises two main components: a multi-head attention mechanism and a position-wise fully connected feed-forward network.

Multi-Head Attention

The multi-head attention mechanism allows the model to focus on different parts of the input sequence simultaneously. This is achieved by having multiple different attention heads attend to different dependencies in the input sequence. The formula for multi-head attention is given by:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{2.3}$$

where each attention head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{2.4}$$

Here, W_i^Q, W_i^K, and W_i^V are the learned projection matrices for the i-th head, and W^O is the output projection matrix.

Feed-Forward Networks

Each layer in the encoder and decoder includes a position-wise feed-forward network, which consists of two linear transformations with a ReLU activation in between. The ReLU activation is defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{2.5}$$

The Feed-Forward Network in each Transformer block is defined as:

$$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2 \tag{2.6}$$

where W_1, W_2, b_1, and b_2 are learned parameters.

Positional Encoding

Since the Transformer does not inherently capture the order of tokens in the sequence, it incorporates positional encodings to provide information about the position of each token. The positional encodings are added to the input embeddings and are defined as:

$$\mathrm{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{2.7}$$

$$\mathrm{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{2.8}$$

where pos is the position, d_{model} and i is the i-th dimension of the input embedding. It is defined in such way with the following reasons:

- Unique value for each position and dimension. The exponential term defines the frequency of the sin and cos function with respect to the dimension i and position pos, the result positional value is unique for each i and pos.
- Proper range of values. The usage of sin and cos function is to constrain the range of positional values so that, when the position vector adds to the input embedding, it does not destroy the information contained in the input embedding.

Note that the constant 10,000 defined in the two formulas above is to ensure the uniqueness of the positional encodings, as recommended in the original Transformer paper [1].

With multi-head attention, Transformer can complex complex representation between two vectors. With feed-forward networks in each Transformer blocks, the complex representations can be used for learning to handle more complex down-stream tasks. And since order matters in sequential data, the usage of positional encoding helps Transformer consider order information instead of treating a sentence as a set of words with no order.

2.1.3 Contributions to Different Domains

The introduction of the Transformer architecture has had profound implications across various areas of NLP and beyond. In NLP, Transformers have set new benchmarks in tasks such as machine translation, text generation, and question answering. Models like BERT [10], GPT-3 [11],

and T5 [12], which are all based on the Transformer architecture, have achieved state-of-the-art performance on numerous benchmarks.

In addition to NLP, the Transformer architecture has been successfully applied to other domains, including computer vision and reinforcement learning. Vision Transformers (ViTs) [13] have demonstrated that Transformers can outperform convolutional neural networks (CNNs) [14] on image classification tasks by treating image patches as sequences of tokens. In reinforcement learning, Transformers have been used to model policies and value functions, leveraging their ability to handle long-range dependencies and large state spaces [15, 16].

The Transformer architecture's flexibility and scalability have made it a cornerstone of modern AI research, driving innovation and enabling the development of more powerful and efficient models across diverse applications.

2.2 Large Language Models

This section reviews the key stages in the evolution of LLM architectures.

2.2.1 Encoder Models: BERT, RoBERTa, CTRL, and ALBERT

The introduction of the Bidirectional Encoder Representations from Transformers (BERT) marked a significant advancement in NLP. BERT is an encoder-based model that captures context from both directions, making it highly effective for various NLP tasks. BERT is trained using two primary objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The MLM objective involves randomly masking some tokens in a sentence and training the model to predict these masked tokens, formulated as:

$$L_{MLM} = -\sum_{i=1}^{N} \log P(w_i|\mathbf{w}_{\setminus i}) \tag{2.9}$$

where w_i is the masked token, $\mathbf{w}_{\setminus i}$ represents the surrounding context and N is the number of

tokens in the sentence.

The NSP objective requires the model to predict whether a given sentence B follows sentence A in the original text:

$$L_{NSP} = -\sum_{j=1}^{M} [y_j \log P(y_j|A,B) + (1-y_j)\log(1-P(y_j|A,B))] \tag{2.10}$$

where y_j is a binary label indicating whether sentence B follows sentence A, M is the number of sentence pairs in the training batch. $P(y_j|A,B)$ is the predicted probability that B is the next sentence after A for the j-th sentence pair.

Beyond BERT, several other encoder models have significantly impacted the field of NLP. One such model is RoBERTa (Robustly optimized BERT approach) [17], which builds upon BERT by optimizing its pre-training procedure. RoBERTa modifies BERT's training by removing the NSP objective, increasing the batch size and training data, and extending the training time. These changes result in a model that outperforms BERT on many NLP benchmarks without altering the underlying architecture. Another prominent encoder model is ALBERT [18], which addresses the issue of model size and training efficiency. ALBERT achieves this by sharing parameters across layers and factorizing the embedding parameterization, significantly reducing the number of parameters while maintaining performance. The loss function for ALBERT is similar to BERT's MLM objective, but with a focus on efficiency:

$$L_{MLM}^{\text{ALBERT}} = -\sum_{i=1}^{N} \log P(w_i|\mathbf{w}_{\setminus i}, \Theta)$$

where Θ denotes the shared parameters across layers, w_i is the masked token, $\mathbf{w}_{\setminus i}$ represents the surrounding context and N is the number of tokens in the sentence. These encoder models demonstrate the continuous evolution in optimizing training procedures and improving efficiency without sacrificing performance.

2.2.2 Decoder Models: GPT Series, Transformer-XL, LLaMA

The shift towards decoder-only models has been exemplified by the Generative Pre-trained Transformer (GPT) model series. The decoder-only models, by its name, does not contain an encoder part to encode input tokens as representations. Instead, it uses the previous generated tokens and the current token to predict as the input to generate the next token. A full sentence is then generated by iteratively appending previous generated tokens to the input and predicting the next token. The decoder-only model is designed for tasks where generation is based solely on the input sequence. Also, there is one class of models named encoder-decoder models, which combines the ideas of encoder and decoder models. It uses the encoder component to process input context into representations, and uses the decoder component to generate tokens, based on both the previous generation sequence and the input context representation generated by the encoder component.

The original GPT model introduced the idea of fine-tuning a pre-trained transformer on specific tasks. GPT-2 [19] expanded on this concept by significantly increasing the model size and training on a more diverse dataset, demonstrating impressive language generation capabilities.

GPT-3, with up to 175 billion parameters, pushed the boundaries of what LLMs could achieve. It leverages the Transformer decoder architecture to generate coherent and contextually relevant text based on a given prompt. The primary training objective for GPT models is auto-regressive language modeling, where the model is trained to predict the next token in a sequence:

$$L_{AR} = -\sum_{t=1}^{T} \log P(x_t | x_{<t}) \tag{2.11}$$

where x_t is the token at position t and $x_{<t}$ represents the preceding context.

Beyond the GPT series, Transformer-XL [20] introduced a novel segment-level recurrence mechanism and relative positional encoding, addressing the context fragmentation issue in standard Transformers. This allows Transformer-XL to capture longer-term dependencies than its predecessors. The recurrence mechanism is represented by:

$$h_t^n = \text{TransformerBlock}(h_t^{n-1}, [h_{\tau+1}^{n-1}, ..., h_t^{n-1}]) \tag{2.12}$$

where h_t^n is the hidden state at time step t and layer n.

CTRL (Conditional Transformer Language model) [21] is another significant model that incorporates control codes to guide text generation according to specific attributes or styles. This innovation allows users to steer the output more effectively, making it highly useful for various applications, from creative writing to content generation.

Recent advancements include models like the LLaMA series of models. LLaMA 1 [22] introduces a series of architectural modifications aimed at optimizing training and inference efficiency. It incorporates layer pre-normalization before the attention and feed-forward layers, in contrast to the post-normalization used in the vanilla Transformer. This change helps stabilize training for deep networks. The feed-forward network is expanded with an additional intermediate normalization layer, improving gradient flow and convergence.

Each of these decoder models showcases advancements in handling longer contexts, increasing model capacity, and providing more control over text generation. They demonstrate how continuous improvements in model architecture and training techniques contribute to the growing capabilities of language models in various applications.

2.3 Self-evolution of the LLMs

In this section, we provide an overview of existing research on self-evolution techniques aimed at enhancing LLMs, with a focus on training time improvement.

Tao et al. [23] surveys the current progresses of LLM and LLM-based agent self-evolution approaches, and proposed a unified framework to categorize different approaches. Some works focus on LLM self-improvement during the training time. SELF-INSTRUCT [24] focuses on generating diverse and complex instruction data using an evolutionary approach. Initially, a simple instruction

is created and then evolved into more complex or new instructions through various operations, such as adding constraints, deepening the reasoning steps, or complicating the input. This process is repeated multiple times to generate a large and diverse dataset. The generated instructions are then filtered to remove failed or invalid instructions, ensuring high quality and diversity in the final dataset. WizardCoder [25] focuses on increasing the quality of the training dataset by evolving simpler code instructions into more complex ones using specific modifications tailored for coding tasks. These modifications include adding constraints, replacing common requirements with specific ones, introducing more reasoning steps, and incorporating debugging and complexity constraints to generate a diverse and challenging set of code instructions. WizardLM [26] generates complex and diverse instruction data by using a language model to iteratively evolve initial instructions into more intricate versions and create new, varied instructions. This process includes specific steps to increase the complexity and reasoning required for each instruction and to generate entirely new instructions that cover a broader range of topics and skills. MetaMath [27] focuses on improving mathematical problem-solving abilities in LLMs by creating a diverse and high-quality dataset called MetaMathQA. This dataset is generated by rewriting existing mathematical questions using forward and backward reasoning, rephrasing, and augmenting answers, which is then used to fine-tune models for enhanced mathematical reasoning skills. LMSI [8] focuses on enhancing the performance of LLMs by using a method where the model generates its own training data through multiple reasoning paths and selects the most consistent answers using majority voting. This self-generated data is then used to fine-tune the model, significantly improving its reasoning abilities without relying on ground truth labels. AlpaGasus [28] involves generating the dataset by using augmentation techniques to create diverse training data. First, they apply answer augmentation to create multiple correct answers for a single question. Then, they use question bootstrapping to generate new questions from existing ones. Filtering and refining involve scoring each data pair and selecting only those above a certain threshold to ensure high quality. Star [29] creates high-quality training data by using LLMs to score and filter out low-quality samples. They generate the dataset by prompting a powerful language model like ChatGPT to rate each instruction-input-response triplet and only keep those with scores above a certain threshold. This threshold-based filter-

ing method ensures that only the most accurate and helpful data points are used for fine-tuning. GRATH [30] involves generating pairs of truthful and untruthful answers to questions using the model itself, then fine-tuning the model by optimizing for the preference of truthful answers over untruthful ones, iteratively improving its accuracy. They filter out low-confidence data by ensuring that only answers formatted correctly and aligned with the few-shot demonstrations are used in the training process.

2.4 Learning from critique in self-evolution

In this section, we investigate learning from critique paradigm researchers used in self-evolution techniques for LLMs.

In LLMs self-evolution, different kinds of critique are used to evolve an LLM towards different desired goals, the critique can come from either the LLM itself or the environment. Some works uses LLMs as critics to generate numerical critique scores. DLMA [31] focuses on aligning with human expectations without relying on human-annotated preference data. The methodology involves generating preference data using contrastive prompt pairs, evaluating the responses with a probability-based self-rewarding score, and then aligning the LLM using direct preference optimization (DPO) based on these scores. LSX [32] focuses on enhancing AI model performance by having the model explain its decisions to itself. The methodology involves a learner model and an internal critic model; the learner makes predictions and generates explanations, while the critic evaluates the explanations and provides feedback to improve the learner. This iterative process, tested on image classification tasks, aims to improve model generalization, mitigate confounding factors, and produce more relevant and faithful explanations. Self-Alignment [33] focuses on improving the factual accuracy of LLMs by using self-evaluation to align generated responses with factual information. The methodology involves generating multiple responses to a prompt, evaluating their factual accuracy using self-evaluation, and then fine-tuning the model using preference data derived from these evaluations. Factuality scores are generated based on how well the responses align with internal knowledge, and the model is trained to prioritize responses with higher scores, ensuring

the scores are accurate for DPO [34] training by validating them against known facts.

Another line of works use a type of more informative critique: textual critique. CAI [35] uses a list of principles as the only human oversight and uses LLM to generate self-critiques and revision, then use SFT to fine-tune on the revisions. Self-Refine [36] focuses on improving outputs of LLMs through a process of iterative self-feedback and refinement. The methodology involves generating an initial output using the LLM, having the same LLM provide critique on this output, and then refining the output based on the critique, repeating this cycle until the desired quality is achieved. The method does not require any additional training or supervised data, leveraging the model's inherent capabilities to generate and refine its own critique iteratively. TextGrad [37] proposes a framework that back-propagates textual feedback provided by LLMs to optimize individual components of compound AI systems. This framework transforms AI systems into computation graphs where variables receive natural language critiques from LLMs, guiding the optimization process. Critiques and feedback are used as "textual gradients," offering detailed and interpretable natural language suggestions that describe how each variable should be adjusted to improve the system's performance. Reflexion [38] uses a framework that enhances the learning of language agents through verbal self-reflection rather than traditional fine-tuning. In this approach, the agent receives feedback in the form of verbal summaries generated from its own performance, which are stored in an episodic memory buffer. These reflective summaries provide detailed, actionable insights that guide the agent's decision making in subsequent trials, improving its performance over time by learning from its previous mistakes. LaFFi [39] involves a four-step process: generating answers using a pre-trained LLM, annotating these answers with natural language feedback from both AI and human annotators, using supervised learning to predict the feedback for fine-tuning, and applying LoRA for parameter-efficient fine-tuning. This approach integrates natural language feedback directly into the fine-tuning process to improve the model's performance. The natural language feedback includes detailed evaluations of the correctness and appropriateness of the model's responses, which help the model learn better mappings between input contexts, the questions, its answers, and the anticipated feedback.

ELITE takes an approach that uses the LLM itself to generate and refine self-critique labels.

The critique labels will be used in SFT as we want the LLM to predict what critique it should receive for certain answers.

2.5 Parameter Efficient Methods for LLMs

LLMs have achieved significant success across various NLP tasks due to their impressive capacity and ability to generalize from vast amounts of data. However, these models come with substantial computational and storage requirements, posing challenges for efficient deployment, especially in resource-constrained environments. This section explores parameter-efficient methods aimed at addressing these challenges, focusing on quantization techniques and the Low-Rank Adaptation (LoRA) series methods. We use quantization techniques to convert our model weights to 4-bit so that less GPU memory and computation is required to run our framework. We use LoRA methods to further reduce GPU memory and computation usage and fine-tune our model more precisely by updating certain components in an LLM.

2.5.1 Quantization Methods

Quantization is a technique that reduces the precision of the weights and activations of neural networks, thereby decreasing the memory footprint and computational cost. It can be broadly categorized into post-training quantization and quantization-aware training. In this work we use post-training quantization to speed up the inference process and use quantization-aware training to reduce the memory usage during training.

Post-training quantization converts a pre-trained model's weights to lower precision, typically from 32-bit floating point (FP32) to 16-bit (FP16), 8-bit (INT8), or even lower. This process can reduce model size and inference latency with minimal impact on model accuracy [40]. Quantization-aware training, on the other hand, incorporates quantization into the training process itself, allowing the model to adjust to the lower precision and maintain higher accuracy [41].

Recent advances in quantization methods have led to the development of techniques such as

dynamic quantization, which adjusts the quantization parameters on-the-fly based on the input data distribution, and mixed-precision quantization [42], which applies different precision levels to different parts of the model. These techniques ensure a better trade-off between model performance and efficiency.

2.5.2 Low-Rank Adaptation Methods

Low-Rank Adaptation (LoRA) [43] is a method used to adapt large pre-trained language models to specific tasks efficiently by reducing the number of parameters that need to be fine-tuned. LoRA achieves this by injecting trainable low-rank matrices into each layer of the model. In this work, we use LoRA as part of our framework to help the training data update certain modules of the LLM.

An overview of LoRA is shown in Figure 2.1. Given a pre-trained weight matrix $W_0 \in \mathbb{R}^{d\times k}$, LoRA decomposes the update to this weight matrix into two smaller matrices $A \in \mathbb{R}^{d\times r}$ and $B \in \mathbb{R}^{r\times k}$, where $r \ll \min(d, k)$. The adapted weight matrix W can be expressed as:

$$W = W_0 + \Delta W \tag{2.13}$$

where $\Delta W = AB$. The matrices A and B are initialized such that A is a random Gaussian matrix and B is a zero matrix. During training, only A and B are updated, while W_0 remains fixed. The forward pass through the adapted layer becomes:

$$y = W_0 x + \Delta W x = W_0 x + ABx \tag{2.14}$$

where $x \in \mathbb{R}^k$ is the input to the layer. By constraining ΔW to be low-rank, LoRA reduces the number of trainable parameters, making the adaptation process more efficient.

Many researchers extend LoRA to make it more robust. LoRA Dropout [44] applies random noises to the trainable low-rank matrices and it increases the sparsity to prevent overfitting during fine-tuning. Laplace-LoRA [45] uses a post-hoc Laplace approximation approach [46] to the LoRA

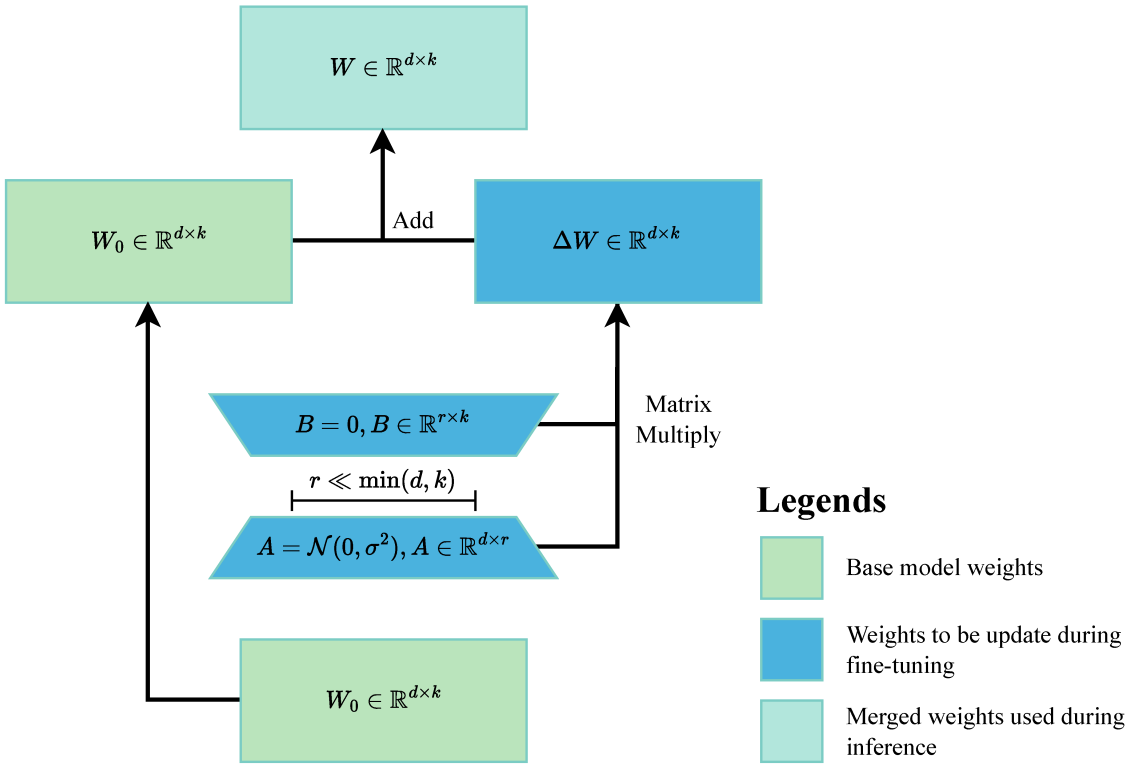


Figure 2.1: An overview of LoRA. It decomposes the weight matrix into two low-rank matrices to reduce active parameter sizes.

parameters to prevent fine-tuned LLMs from over-confidence. LoRA+ [47] uses different learning rate for different LoRA matrices A and B. Some works use multiple LoRA modules in the framework to improve model performance. LoRAHub [48] uses a gradient-free method Shiwa [49] to aggregates various LoRA modules trained on different tasks. MOELoRA [50] uses a Mixture-of-Experts (MOE) approach to apply LoRA in a multi-task learning setting, which outputs multiple LoRA modules. It uses a task-specific gate function to assign weights to different expert modules and the final parameters in used will be a weighted sum of all the expert weights.

2.6 Gaps in Literature

Despite significant advancements in LLMs and their self-evolution techniques, several gaps persist in the existing literature. These gaps hinder the scalability and effectiveness of these models in practical applications. Three critical aspects where the current literature falls short are:

- **The reliance on extensive human annotation**
- **The lack of generality**
- **The lack of scalability**

2.6.1 Reliance on Extensive Human Annotation

Many fine-tuning frameworks still heavily depend on human annotation to provide feedback and corrections, which limits their scalability. For instance, LaFFi relies on extensive human annotations to provide accurate critiques, making the process time-consuming and labor-intensive. In contrast, ELITE significantly reduces the need for human intervention by requiring only a small set of human-annotated prompt examples to provide the necessary examples for an LLM to follow.

2.6.2 Lack of Generality

Existing frameworks often struggle with generality, as they are tailored to specific tasks or domains. For example, SELF-REFINE focuses on improving code readability and mathematical reasoning, but may not generalize well to other tasks without significant modifications. Promptbreeder, while effective in evolving prompts for specific reasoning tasks, requires domain-specific problem descriptions and mutation prompts, limiting its applicability across diverse domains. Reflexion focuses on self-reflecting code-generation agents, which are highly specialized. LMSI employs training set questions and refined LLM self-generated responses as labels to iteratively fine-tune the model; this approach is specifically designed for problems with exact numerical answers. ELITE is designed to be task-agnostic, leveraging a flexible critique generation process that can adapt to various contexts and tasks, thereby enhancing its generality.

2.6.3 Lack of Scalability

SELF-INSTRUCT uses an amount of manually annotated seed tasks to make an LLM generate variations of the training questions from the initial seed tasks. However, to compose data with more complex tasks, more manual annotated tasks must be collected. In contrast, ELITE requires a small set of human annotations to bootstrap the training process, and this requirement does not change even if the LLM requires more data in the dataset for fine-tuning. Also, it uses parameter-efficient methods that allow us to adapt and improve using fewer computational resources as model and dataset sizes grow.

In summary, ELITE offers a robust and scalable framework that minimizes the need for human annotation, it is a general LLM training method, and has the ability to scale if more data is necessary for training. In the next chapter, we cover the details of the **ELITE** framework.

Chapter 3

The ELITE Framework

In this chapter, we will first provide a detailed explanation of the definitions and concepts essential to our framework. We particularly focus on the self-critique process. Following that, we will delve into the details of the ELITE framework, explaining its iterative learning process. We will then describe the methodology for selecting prompt examples dynamically. Next, we will discuss the critique generation process, which employs a majority voting approach to select critiques. Finally, we will cover the parameter-efficient fine-tuning method used in our framework to optimize memory and computational resources.

3.1 Definitions

In SFT, a labeled dataset is typically used for fine-tuning. The input consists of a question for the LLM to answer, and the output is the ground truth answer preferred by humans. In a more specialized form of supervised fine-tuning known as instruction fine-tuning, the context of the task is added to the input for the LLM to follow [51]. During fine-tuning, the LLM optimizes itself using cross entropy loss on the provided SFT dataset, eventually learning an effective mapping from (context, question) tuples to answers.

ELITE adopts a similar SFT approach to improving LLM performance. However, instead of

relying on a dataset with (context, question) tuples and their corresponding answers as labels, we use an iterative process to generate and refine a fine-tuning dataset. This dataset consists of (context, question, answer prediction) tuples as inputs, with the self-critiques generated by the LLM itself serving as the labels. Our goal is to let the LLM learn a mapping from the task context, question and the response output by the LLM to a natural language critique, such that the LLM learns what critique it may receive if it outputs certain responses. This idea is first proposed in LaFFi [39], which the authors use both human and LLM to collect such a critique prediction dataset with external critiques as the supervised fine-tuning labels. Humans can generate accurate critiques but the annotation process is time consuming. Although LLMs can generate critiques in a short time, the quality of the generation is noisy without further refinement. Thus, in LaFFi, the LLM fine-tuned on the LLM generated dataset performs worse than the model fine-tuned using human-annotated dataset. In contrast, our method allows the LLM to progressively enhance its performance using self-generated critiques with a majority voting mechanism to refine the critique quality, and we gradually refine the quality of our critique dataset by refining the prompt examples we used in each iteration.

Table 3.1 lists the key abbreviations we use. We define a question $q \in$ question dataset D_{q}, an answer prediction $a \in$ answer prediction dataset D_{a} generated by an LLM M, self-critique generation prompt p_{critique}, current prompt examples used to generate answer and critiques e_{curr}. We define the **self-critique** process in the context of LLM training as follows. For a given question q, an answer a given by an LLM M, and a critique generation prompt p_{critique}, a self-critique is the process of letting M evaluate the appropriateness of a to q, analyze the reason for this appropriateness, and provide a more appropriate answer if applicable. For each $(q, a, e_{\text{curr}}, p_{\text{critique}})$ tuple, we first sample a set of critique candidates $D_{\text{c\_raw}}$ from $M(q, a, e_{\text{curr}}, p_{\text{critique}})$ using Temperature Sampling with the default temperature 1.0, after critique refinement we get one critique c, which then stored in a critique dataset D_{c}.

Table 3.1: This table includes the key abbreviations we used in the **ELITE** framework.

Full Form	**Notation**
Answer Generation Prompt	p_{a}
Answer Prediction	a
Answer Prediction Dataset	D_{a}
Critique	c
Critique Candidate	c_raw
Critique Candidate Dataset	D_{c_raw}
Critique Dataset	D_{c}
Critique Generation Prompt	p_{critique}
Current Prompt Examples	e_{curr}
Desired Iterations	I
Initial Prompt Examples	e_{init}
Large Language Model	M
Prompt Example Amount	k
Question	q
Question Dataset	D_{q}
Text Encoder	E

3.2 Framework Details

In this section, we first provide an overview of our proposed framework **ELITE**. Then, in the remainder of the chapter,, we detail all the key steps and aspects of our framework.

3.2.1 Framework Overview

Figure 3.1 provides an overview of **ELITE**. The first iteration begins with a dataset of instructions and questions.

In Step 1, combined with initial human-labeled prompt examples, we use an LLM to generate answer predictions.

In Step 2, the LLM is tasked with critiquing these answers to assess their appropriateness for the corresponding questions, along with providing reasons. Self-Refine [36] chooses to use self-critiques during inference to refine output, but we choose to use them during training so that no additional overhead during inference. We apply majority voting to filter out noisy responses, retaining only high-quality critiques.

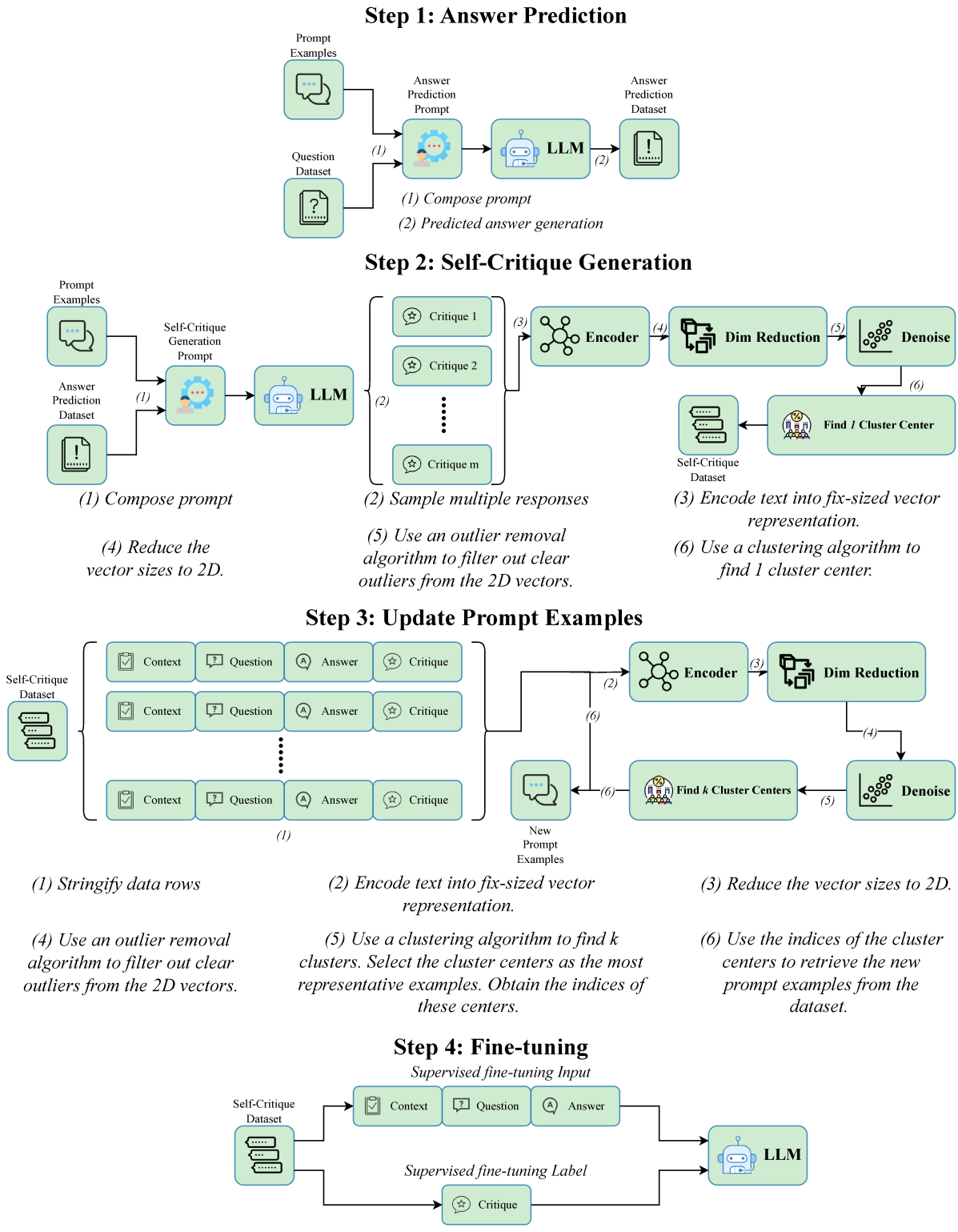


Figure 3.1: Overview of the **ELITE** framework.

In Step 3, to ensure each iteration uses the most representative examples, we cluster the critique data rows in a reduced 2D representation space. We then select rows whose representation vectors are closest to the cluster centers to serve as representative examples for the subsequent iteration.

In Step 4, we fine-tune the same LLM using the self-annotated critique dataset, with inputs being the task instruction, question, and predicted answer, and the output being the annotated critique. The LLM is trained with the objective of accurately predicting the critique it should receive for the given inputs.

Our framework repeats Step 1 to Step 4 several times. Below, we discuss the key aspects of our framework.

3.2.2 Predicted Answer Generation

In this subsection, we explain our design for the Step 1 of our framework, as shown in Figure 3.2 and in Line 14 of Algorithm 1. To start the process, our framework first requires: (1) An initial, human-annotated prompt example dataset, and (2) A question dataset that includes tasks with diverse formats. The size of these two datasets will be covered in Section A.2. We use the initial human-annotated prompt example dataset as exemplars for the LLM so that it knows what kind of responses are desired. We use the question dataset as part of the supervised learning input for the fine-tuning process in Step 4. Since our objective is to obtain a self-critique dataset at each iteration, to ensure that the self-critiques are generated based on the current understanding of our LLM, we use the LLM to generate an answer prediction for all the questions inside the question dataset. In this case, we ensure that all the generated answers come from the LLM's current knowledge. Later, when we use the self-critique data generated on these answers and train the LLM with these data, the LLM learns better whether these answers are appropriate, which affects the LLM's subsequent inference quality. The prompt we use for answer prediction is in Table A.2.

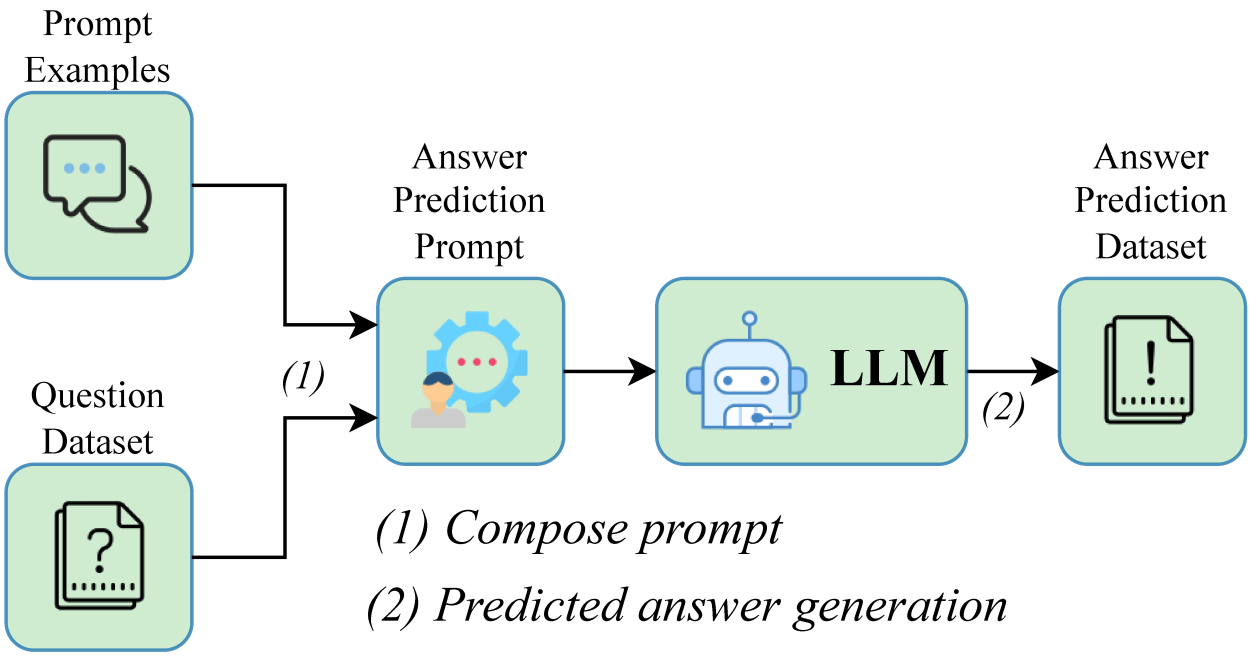


Figure 3.2: In Step 1, we use an LLM to generate an answer dataset, given a set of prompt examples and a set of questions.

3.2.3 Self-Critique Generation

In this subsection, we explain our design for Step 2 of our framework, as shown in Figure 3.3 and in Line 16 of Algorithm 1. To generate a high-quality critique dataset, we explore various critique generation prompt formats and employ a majority voting approach to select responses. Given the same prompt input, the LLM samples a series of critique responses. Since these are free-form responses, they can vary significantly and still convey similar semantics. After we sampled a series of self-critiques from the LLM, we first use a pre-trained text encoder model to convert responses with different lengths into same-length, high-dimensional vectors. We use BERT as our text encoder model since it is pre-trained on massive textual data with the combined loss of masked language modeling and next sentence prediction loss, and it provides rich, context-aware embeddings that capture the nuances of language. We then apply a dimension reduction algorithm — principal component analysis (PCA) [52] — to reduce each high-dimensional vector to a 2D vector, chosen for its simplicity and interpretability. Then we use outlier detection to eliminate noisy data, and select the response at the cluster center as the final critique. In Algorithm 1, line 2 we also detail

our clustering process in the form of an algorithm. This method ensures that the selected response is representative of the responses in the semantic space understood by the text encoder model and PCA. The prompt we use for self-critique generation is in Table A.3.

Algorithm 1 The algorithm of **ELITE**.

1: **Input:** Question dataset D_q, answer prediction dataset D_a, model M, text encoder E, prompts $\{p_\text{inference}, p_\text{critique}\}$, current prompt examples e_curr, human annotated examples e_h, prompt example amount k
2: **function** GET_CLUSTER_CENTERS(2d_representation_list, center_amount)
3: # Encode a list of strings into vector representations using the text encoder.
4: vec_encoded $\leftarrow E(2d\_representation\_list)$
5: # Reduce the dimension of the vector representations to 2D using PCA.
6: dimension_reduced $\leftarrow$ PCA(vec_encoded)
7: # Remove the outliers from the 2D vectors.
8: clean_dimension_reduced $\leftarrow$ Outlier_Removal(dimension_reduced)
9: # Use k-Means algorithm to find certain amount of cluster centers.
10: **return** KMeans(clean_dimension_reduced, center_amount)
11: $e_\text{curr} \leftarrow e_h$
12: **for** each iteration $i \in \{0, 1, \dots I\}$ **do**
13: # Step 1: Answer prediction.
14: $D_\text{a} \leftarrow M(D_q, p_\text{inference}, e_\text{curr})$
15: # Step 2: Critique generation with majority voting.
16: $D_\text{c\_raw} \leftarrow M(D_q, D_\text{a}, p_\text{critique}, e_\text{curr})$
17: $D_\text{c} \leftarrow$ get_cluster_centers$(D_\text{c_raw}, 1)$
18: # Step 3: Obtain new prompt examples.
19: 2d_representation_list $\leftarrow$ str$(D_\text{c}, p_\text{critique})$
20: # Get k cluster centers, set as new examples.
21: $e_\text{curr} \leftarrow$ get_cluster_centers(2d_representation_list, k)
22: # Step 4: Fine-tuning.
23: SFT$(M, p_\text{critique}, D_\text{c})$

3.2.4 Prompt Example Selection

In this subsection, we explain our design for Step 3 of our framework, as shown in Figure 3.4 and in line 19 of Algorithm 1. To ensure that an LLM generates responses that meet human expectations, it is crucial to include a few representative examples in the prompt, especially when the inference responses for a dataset should exhibit varied output behaviors. Manually selecting several prompt examples that adequately represent the dataset can be challenging. In our approach, rather than using a static set of prompt examples, we dynamically change the examples in every iteration. This

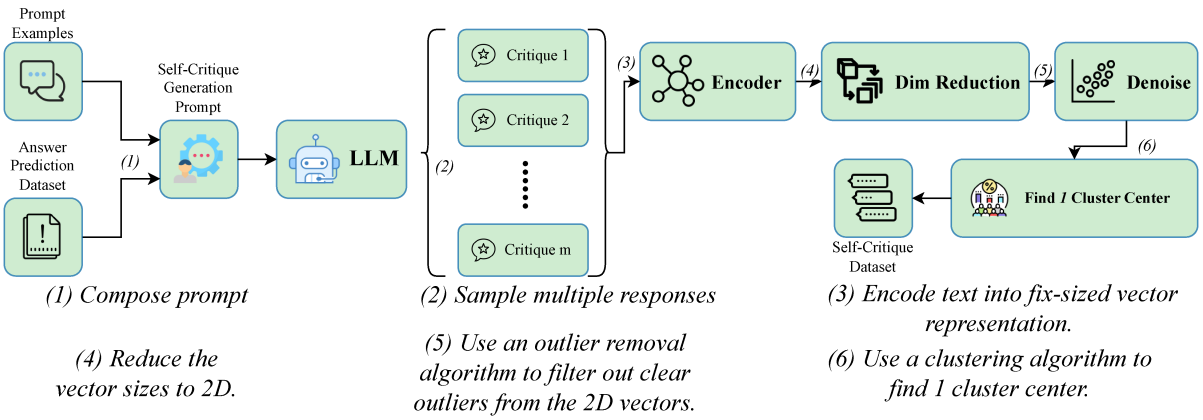


Figure 3.3: In Step 2, we use the answer prediction dataset and a set of prompt examples to generate critiques. For each (context, question, predicted answer) pair, we randomly sample m critiques with the LLM. Then we use majority voting to select only one critique as the final critique.

strategy compels the LLM to use the most representative examples from the 2D representation space of the critique dataset as the prompt examples for the next iteration's inference tasks.

Detailed procedures for prompt example optimization are detailed in Algorithm 1. To initialize the algorithm (line 11) we begin with human-annotated examples comprising context, question, answer, and critique. Each sub-task requires only a few prompt examples so that the LLM can mimic the format of these examples during inference and thus the size of our prompt example dataset is significantly smaller than our fine-tuning dataset. On line 19, after the LLM annotates the critique dataset, each row is concatenated into a single string. A pre-trained encoder model then encodes this string into a high-dimensional vector. We use BERT as our text encoder model since it is pre-trained on massive textual data with the combined loss of masked language modeling and next sentence prediction loss, and provides rich, context-aware embeddings that capture the nuances of language. We then employ PCA to reduce each high-dimensional vector to a 2D vector. An outlier removal algorithm [53] filters out data points at the distribution's edges (selected for its unsupervised nature and efficiency with large datasets, as well as its robustness against noisy data). Finally, we apply a clustering algorithm, k-Means [54], to identify k clusters and select k rows whose 2D representations are closest to the centers of the k clusters. These selected rows are formatted and used as prompt examples in the subsequent iteration. Compared to the extensive

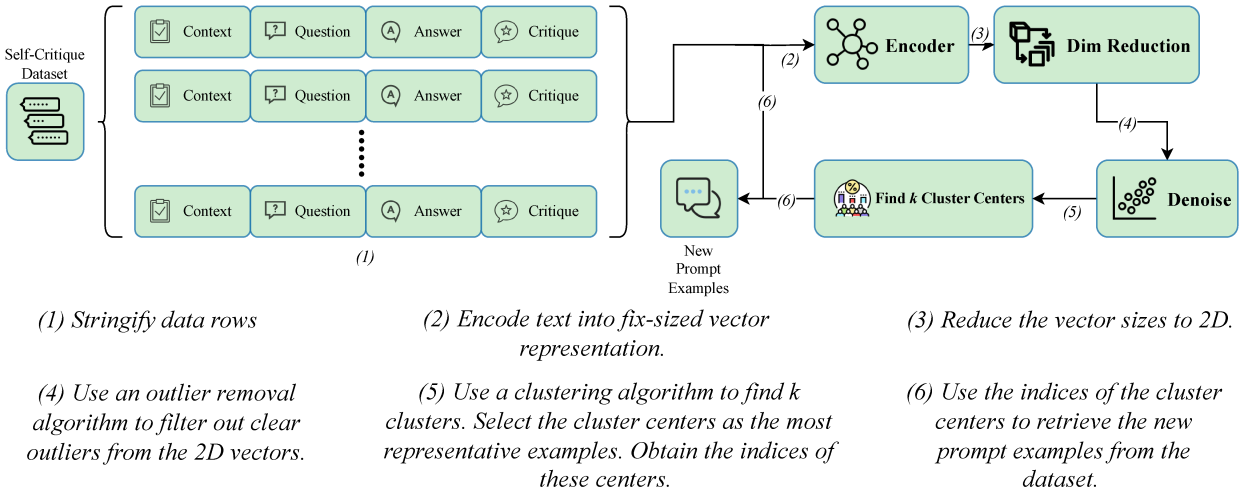


Figure 3.4: In Step 3, for each (context, question, answer prediction, critique) pair, we concatenate each pair into a full paragraph. A textual encoder model then encode the texts into high-dimensional representation vectors with the same dimension. These vectors are reduced to 2D and being clustered to select a certain number of cluster centers. Finally, we use the indices of these cluster centers to find representative prompt examples from the Self-Critique Dataset.

computational demands of LLM inference, the costs of using dimension reduction and clustering algorithms are negligible.

3.2.5 Parameter Efficient Fine-tuning

In this subsection, we explain our design for Step 4 of our framework, as shown in Figure 3.5 and in Line 23 of Algorithm 1. Since we obtained the self-critique dataset from Step 2, we can use it to fine-tune our LLM to build a better mapping from the (context, question, answer) tuple to the self-critique. We draw inspiration from LaFFi [39], which first used the idea of improving an LLM's reasoning ability by fine-tuning an LLM to predict what feedback it should receive from (context, question, answer) tuples. In our case, we use the concatenation of (context, question, answer) with proper prompt to ask the LLM predict what critique it should receive as the supervised learning input, and we use the self-critiques as the labels to fine-tune our LLM. The prompt format we used during fine-tuning can be found at Table A.4.

Considering the need for memory and computational efficiency, we employ a parameter-efficient method, LoRA [43], during fine-tuning. This method restricts updates to specific critical linear

modules within the LLM, as listed in Table A.1. For an LLM with 7B parameters, only **0.59%** of the total parameters are trained, significantly reduces the amount of GPU memory and training time required compared to a full fine-tuning.

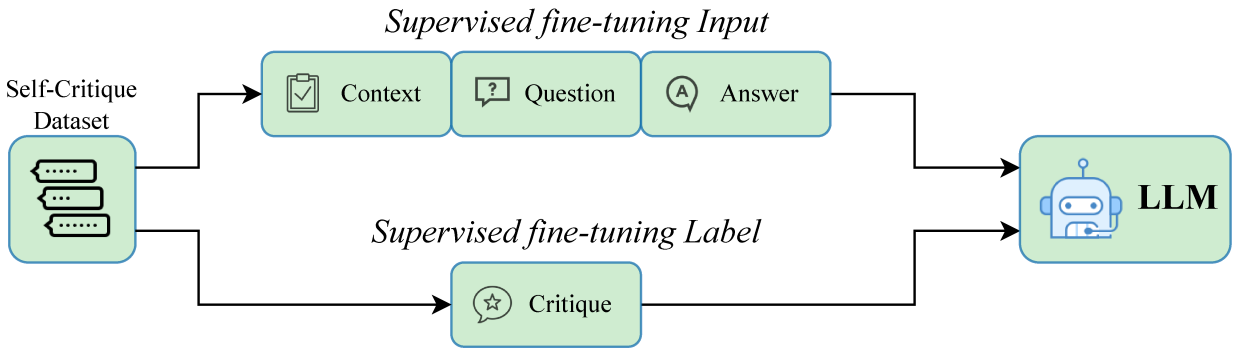


Figure 3.5: In Step 4, we use the (context, question, answer) pairs with prompt as our supervised fine-tuning input and use the self-critiques as our supervised fine-tuning label from the self-critique dataset for fine-tuning.

3.3 Conclusion

In this chapter, we introduced the ELITE framework, detailing its definitions and core concepts, particularly focusing on the self-critique process and the iterative learning approach. We described our methodology for answer generation, self-critique generation, dynamically selecting prompt examples, generating critiques through majority voting, and employing a parameter-efficient fine-tuning method to optimize memory and computational resources.

The next chapter will present the experimental results obtained using the ELITE framework. We will explore the effects of self-training iterations, the efficacy of majority voting in critique selection, and the impact of dynamically selected prompt examples.

Chapter 4

Empirical Evaluation

In this section, we first outline our evaluation objectives and the dataset we used, the baseline we used for comparison, and the general experimental setup. We then discuss the experimental results.

4.1 Evaluation Objectives

Our evaluation objectives are:

1. Evaluating the generalizability of the proposed framework across various NLP tasks in comparison to existing fine-tuning and self-improvement methods.
2. Investigating the impact of self-training iterations on model performance.
3. Analyzing the influence of the number of responses in the majority voting mechanism.
4. Assessing the effectiveness of clustering algorithms applied to textual representations.
5. Evaluating the effectiveness of the prompt example selection mechanism.
6. Determining the impact of initial prompt examples on the overall system performance.

Sections 4.6.1–4.6.6 each evaluate one of these objectives, first explaining our evaluation purpose,

and then detailing our experiment design to examine a certain aspect of our framework. This chapter ends by summarizing the results and drawing useful insights.

4.2 Dataset

We employed a subset of the multi-task language instruction dataset, Natural Instructions v1.0 [55], as our primary training dataset since it encompasses a diverse array of tasks such as question answering, classification, and entity extraction. It can also enhance the generalization capability of our LLM across a broad spectrum of unseen tasks. The dataset subset includes 2450 data rows spanning 49 different sub-tasks, with each row containing only instructions and questions, without any required ground truth answers. For initial prompt examples, we used 2 human-labeled task examples for 49 subtasks from the Natural Instructions dataset, which in total sum up to 98 examples.

Natural Instructions The Natural Instructions v1.0 dataset, developed by Allen AI, is a comprehensive collection designed to facilitate research and development in NLP. This dataset comprises a diverse set of 61 different sub-tasks and corresponding instructions, aimed at improving the generalization capabilities of large language models. In 61 different tasks, 49 of them compose the training set and 12 compose the evaluation set. Each sub-task in the dataset is accompanied by detailed instructions that describe the objective, input-output format, and examples, enabling models to understand and execute a wide range of language-based tasks. More details about the dataset can be found in Section A.2.

BoolQ The BoolQ (Boolean Questions) dataset [56] is a substantial resource in the realm of natural language understanding, consisting of 16,000 question-answer pairs. Created to facilitate research in reading comprehension and question-answering, BoolQ focuses on yes/no questions derived from real-world queries. Each entry in the dataset comprises a question, a corresponding passage from which the answer can be inferred, and a boolean answer.

SQuAD The SQuAD (Stanford Question Answering Dataset) [57, 58] is a large-scale dataset

designed to enhance machine reading comprehension and question-answering capabilities. It includes over 100,000 question-answer pairs derived from a wide range of Wikipedia articles. What sets SQuAD 2.0 apart is the inclusion of over 50,000 unanswerable questions, making it a more challenging and realistic benchmark for evaluating the robustness of models. Each entry in the dataset consists of a question, a passage from which the answer should be extracted, and the answer itself if it exists. This format ensures that models not only learn to provide precise answers when possible but also to recognize when a question cannot be answered based on the given text.

GSM8K The GSM8K (Grade School Math 8K) dataset [59] is a curated collection of 8,000 grade-school-level math word problems, designed to advance the field of mathematical reasoning in NLP models. Each problem in the dataset includes a detailed question requiring arithmetic, algebra, or basic logic to solve, along with a step-by-step solution. The format ensures that models can learn not just to find the correct answer, but to follow logical steps to reach that answer, mimicking the way a student would approach a math problem. This dataset is particularly useful for training models to handle complex, multi-step reasoning tasks, which is a significant leap from straightforward question-answering datasets.

4.3 Baselines

To demonstrate the effectiveness of our framework, we compared the performance of our framework **ELITE** against a no fine-tuning baseline and two different fine-tuning approaches using LLaMA 2 models at the 7B and 13B scales:

- **Models without fine-tuning (w/o FT)** No fine-tuning baseline represents the performance of the LLM when it operates solely on its pre-trained capabilities without any task-specific adjustments.
- **Supervised Fine-tuning (SFT)** SFT represents the conventional method of fine-tuning LLMs and serves as a baseline for the most widely used approach in the field. Comparing **ELITE** to SFT allows us to demonstrate the added value of our iterative self-critique mecha-

nism over traditional fine-tuning, which typically requires extensive labeled datasets and does not involve any autonomous self-evolution processes.

- **Language Model After Self-Improvement (LMSI) [8]** LMSI represents a robust baseline within the self-evolution paradigm for Large Language Models. An overview of LMSI can be found at Figure 4.1. It employs a self-evolution approach by sampling reasoning traces, refining these through majority voting, and then using the refined outputs as prompt examples for subsequent iterations to enhance LLM performance. We replicated the LMSI setup to ensure comparability.

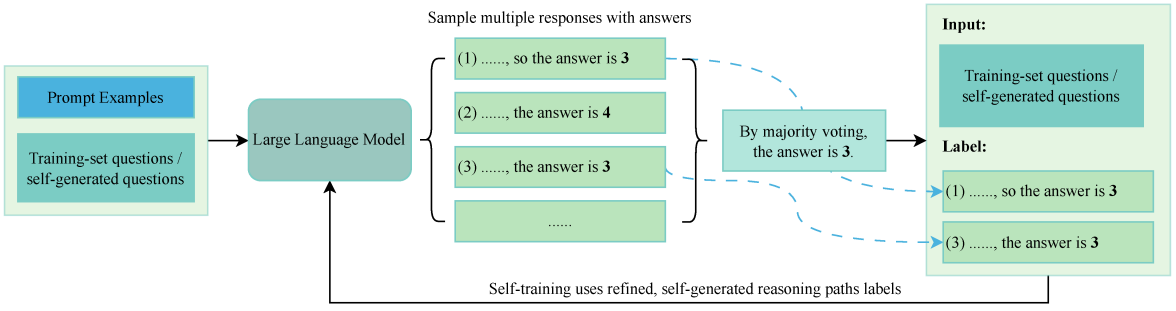


Figure 4.1: An overview of the LMSI approach. It uses majority voting to refine reasoning paths and use them as supervised learning labels to self-train the LLM.

4.4 Evaluation Tasks

We assessed the effectiveness of our framework using two distinct task categories:

- **Reading comprehension** For in-domain evaluation, we used the entire evaluation set of the Natural Instructions v1.0. Out-of-domain evaluations were conducted using the BoolQ dataset [56] for boolean answer selection and the SQuAD 2.0 dataset [57, 58] for answer extraction. We include three examples during evaluation.

- **Arithmetic reasoning** To further test the model's generalization capabilities, we included the GSM8K mathematics dataset [59] for out-of-domain evaluations, using its full test dataset.

The evaluation is done with three prompt examples provided to the LLM so that it can follow the instructions better.

Prompt formats used for evaluation are detailed in the Appendix (A.3).

4.5 Training Settings

Our experiments were conducted using the open-source, auto-regressive Transformer-based model, LLaMA 2 [3], available in 7 billion and 13 billion parameter configurations. The hyperparameter configurations are detailed in Table A.1. We used BERT [10] as our text encoder model. To optimize GPU memory usage and accelerate training, we implemented 4-bit quantization to reduce parameter precision. All experiments require at most 2 NVIDIA V100 16GB GPUs on Compute Canada. As LLM training and evaluation are both time and computation consuming process and the majority in LLM literature do not use multiple seeds, we choose to run each configuration only one time in the following experiments.

4.6 Result

In this section, we present the core results of **ELITE** in comparison to other benchmarks. We conduct experiments to explore the impact of self-training iterations, the quantity of majority voting responses. Additionally, we qualitatively examine the prompt example clustering and selection process by visualizing the prompt examples in an encoded 2D representation space. Finally, we perform an ablation study to demonstrate how the quality of initial prompt examples and the use of prompt example selection affect our framework's performance.

4.6.1 Comparison of the Baselines

In this subsection, we aim to **assess the generalization capability of the proposed framework across different NLP tasks and compare it to the no fine-tuning baseline, the**

Table 4.1: Results on 4 different benchmarks. Fine-tuning methods: (1) No fine-tuning models (w/o FT), also it represents **ELITE** running for 0 iterations; (2) Standard supervised fine-tuning (SFT); (3) Language model after self-improvement (LMSI); and (4) **ELITE**: our framework, 5 iterations. The numbers in parentheses refer to the performance relative to w/o FT.

		Datasets			
		Natural Instructions	**BoolQ**	**SQuAD 2.0**	**GSM8K**
Scales	Methods	ROUGE %	F1	F1	Accuracy
7B	(1) w/o **FT**	51.4	67.1	40.9	21.4
	(2) **SFT**	65.5 (+14.1)	73.4 (+6.3)	45.0 (+4.1)	20.6 (-0.8)
	(3) **LMSI**	59.3 (+7.9)	75.2 (+8.1)	47.4 (+6.5)	26.8 (+5.4)
	(4) **ELITE**	**70.3 (+18.9)**	**78.3 (+11.2)**	**54.7 (+13.8)**	**28.6 (+7.2)**
13B	(1) w/o **FT**	46.4	76.6	54.1	23.0
	(2) **SFT**	67.1 (+20.7)	80.3 (+3.7)	57.5 (+3.4)	24.8 (+1.8)
	(3) **LMSI**	55.7 (+9.3)	78.6 (+2.0)	56.4 (+2.3)	23.8 (+0.8)
	(4) **ELITE**	**74.8 (+28.4)**	**81.7 (+5.1)**	**64.1 (+10.0)**	**25.7 (+2.7)**

existing supervised fine-tuning baseline and the self-improvement method LMSI. A method is said to have good generalization ability if it can perform well on the evaluation tasks on which the formats are different from the tasks it was trained on. To improve an LLM, we want the model not only to perform well on seen tasks, but also to handle unseen (but similar) tasks. Thus, to investigate the generalization ability of **ELITE**, we compare it with the baselines mentioned in Subsection 4.3 and we evaluate different methods using the tasks mentioned in Subsection 4.4. To make sure the framework is robust to model scales, we ran experiments on both 7 billion and 13 billion parameter models.

Table 4.1 shows that **ELITE** consistently outperforms all other approaches across various model scales and on four distinct benchmarks. To generate this table, roughly 410 GPU hours is used for training and 330 GPU hours is used for inferecing. Resource-intensive configuration such as training ELITE with 13B model typically uses 120 GPU hours to finish one run on 2 V100 16GB GPUs. In the in-domain task evaluation using the Natural Instructions dataset, we use the ROUGE-1 [60] score in percentage to measure performance since it is a widely used text similarity measurement metrics in NLP, it is also computationally inexpensive to calculate. Measured by ROUGE, **ELITE** largely surpassed both standard SFT and LMSI. For the out-of-domain benchmarks, **ELITE** also

demonstrated superior generalization ability, showing performance gains across different task types.

ELITE exhibited the most improvement on the SQuAD 2.0 dataset, followed by BoolQ and GSM8K, compared to SFT and LMSI. This is likely due to the similarity in sub-task formats between our training dataset and SQuAD 2.0. Although BoolQ requires Boolean rather than free-form text answers, and GSM8K involves complex mathematical reasoning that is quite distinct from our training set, **ELITE** still managed to enhance performance. Notably, with the help of the self-critique generation mechanism, accuracy on the GSM8K benchmark improved from 21.4% to 28.6% for the 7B model. For LMSI, the performance often lags behind SFT. This is likely because LMSI is designed for tasks with absolute ground truth answers, such as math problems where the final answer is a specific number, allowing LMSI to easily select the majority as the final label. However, for tasks in Natural Instructions, the similarity between answers is more nuanced, making it harder for LMSI to capture, which results in lower improvements on these tasks.

Experiment insight: These results underscore that our approach, requiring no manually annotated supervised labels and minimal human effort in providing prompt examples, not only excels in in-domain tasks but also exhibits robust generalization capabilities.

4.6.2 The Effect of Self-Training Iterations

In this subsection, our evaluation objective is to **examine the effect of self-training iterations.** Each iteration of our framework introduces a new set of predicted answers, critiques, and prompt examples, continually changing the fine-tuning inputs and outputs. Since we do not have a stopping criterion for **ELITE** to follow, it is important to analyze how the number of self-training iterations impacts the fine-tuning performance so that we know when should we stop improving empirically.

We conduct our experiment on the evaluation set of Natural Instructions dataset to obtain our framework's performance on in-domain tasks, which the evaluation task instructions have similar formats as the training task instructions. And we also evaluate our framework on the evaluation set of the GSM8K dataset, which the task instructions are very different from the training task

instructions. We measure the performance on Natural Instructions in ROUGE-1 score and on GSM8K by F1 score, we choose to report the average of these two scores as an overall performance for the current iteration. We report the number of iterations range from 0 to 10 as this range is wide enough to show the trend. Also, we examine both 5 and 10 majority voting responses to ensure the pattern we observe as the number of iterations change holds for different response numbers. We use LLaMA 2 - 7B model for the experiments. Figure 4.2 shows the result.

At iteration number 0, the score corresponds to the 7B w/o FT result in Table 4.1. At iteration number 5, the 10-Majority-Voting score corresponds to the 7B **ELITE** result in Table 4.1. We observed that performance improves with each iteration up to the fifth iteration, after which it peaks and then begins to decline, eventually falling below the performance level of iteration 0, where no fine-tuning was applied. The same pattern appears in both 5-Majority-Voting and 10-Majority-Voting cases.

Experiment insight: This result suggests that an appropriate amount of iterations can enhance our model performance on different tasks while more iterations after may cause undesired behaviors, even negatively affect evaluation performance.

We will present a qualitative study to examine the undesired behavior from the prompt perspective in Subsection 4.6.5.

4.6.3 The Effect of Majority Voting Response Quantity

In this subsection, our evaluation objective is to **examine the effect of the majority voting response quantity.** Sampling a large number of responses from an LLM is both computationally intensive and the inference time grows quickly as more responses are sampled from the LLM, as shown in the red curve in Figure 4.3, necessitating a balance between cost and performance. In this experiment, we use the average performance on both Natural Instruction and GSM8K datasets as the overall score. We use the number of responses equal to 1, 2, 3, 5, 10, 15, 20 and 30 for experiments. To measure the inference cost, we report the average inference time per instance for two datasets.

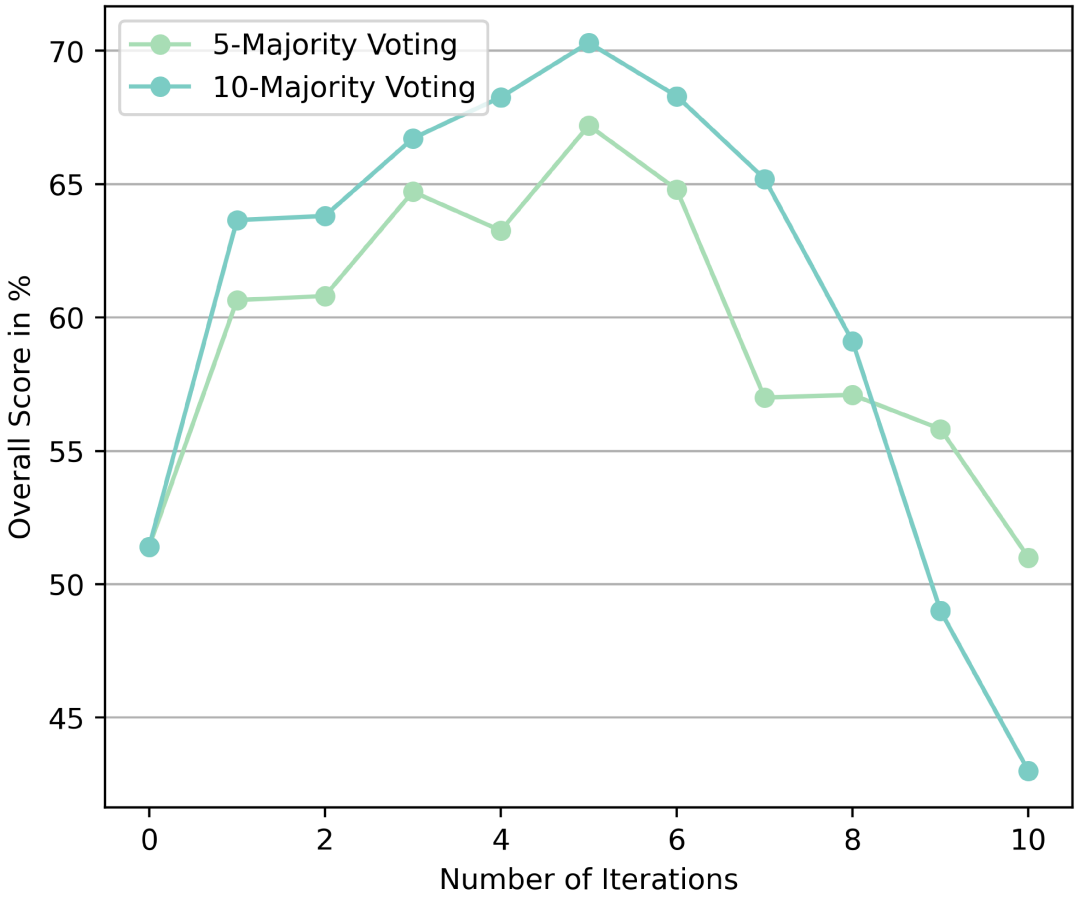


Figure 4.2: This figure illustrates the average performance of our framework on the Natural Instructions and GSM8K datasets with increasing numbers of iterations. Empirically, the performance peaks after the iteration 5 then start to decrease.

Figure 4.3 shows the impact of varying the number of majority voting responses during critique generation. We find that performance improves steadily with an increase in response quantity up to 10, beyond which performance plateaus.

Experiment insight: The result above indicates that increasing the number of responses beyond 10 does not yield further benefits and only adds to computational overhead, suggesting an near-optimal response count for effective performance without excessive resource use.

4.6.4 Efficacy of Clustering Algorithms on Textual Representations

In this subsection, our evaluation objective is to **examine the efficacy of clustering algorithms on textual representations**. Dimension reduction is extensively used in the machine learning community to enhance the accuracy of classification in high-dimensional data [61]. In **ELITE**, our goal is to select cluster centers within a 2D textual representation space using a clustering algorithm, as shown in step 3 in Algorithm 1. To achieve this, it is crucial to examine the clustering performance as the number of clusters increases since the clustering performance affect how representative our prompt examples are.

In this experiment, we choose to qualitatively examine the clustering quality by visualizing the clustering result on different tasks from the Natural Instructions dataset. We try cluster quantities of 2, 3, and 4 to examine the robustness of the clustering design. Our clustering algorithm is robust against different tasks and cluster quantities if it can clearly separate vectors into specified amount of clusters and this holds for various tasks. We present the result of subtask 41[1] and 61[2] from the Natural Instructions dataset. Detailed information on these two subtasks can be found in the Appendix (A.2.1).

Figure 4.4 qualitatively illustrates the vector distributions with an increasing number of cluster examples. From 2 to 4 clusters and for the subtask 41 and 61, all six plots exhibit effective clustering, showcasing distinct boundaries between clusters.

[1] `subtask041_qasc_answer_generation`
[2] `subtask061_answer_generation_ropes`

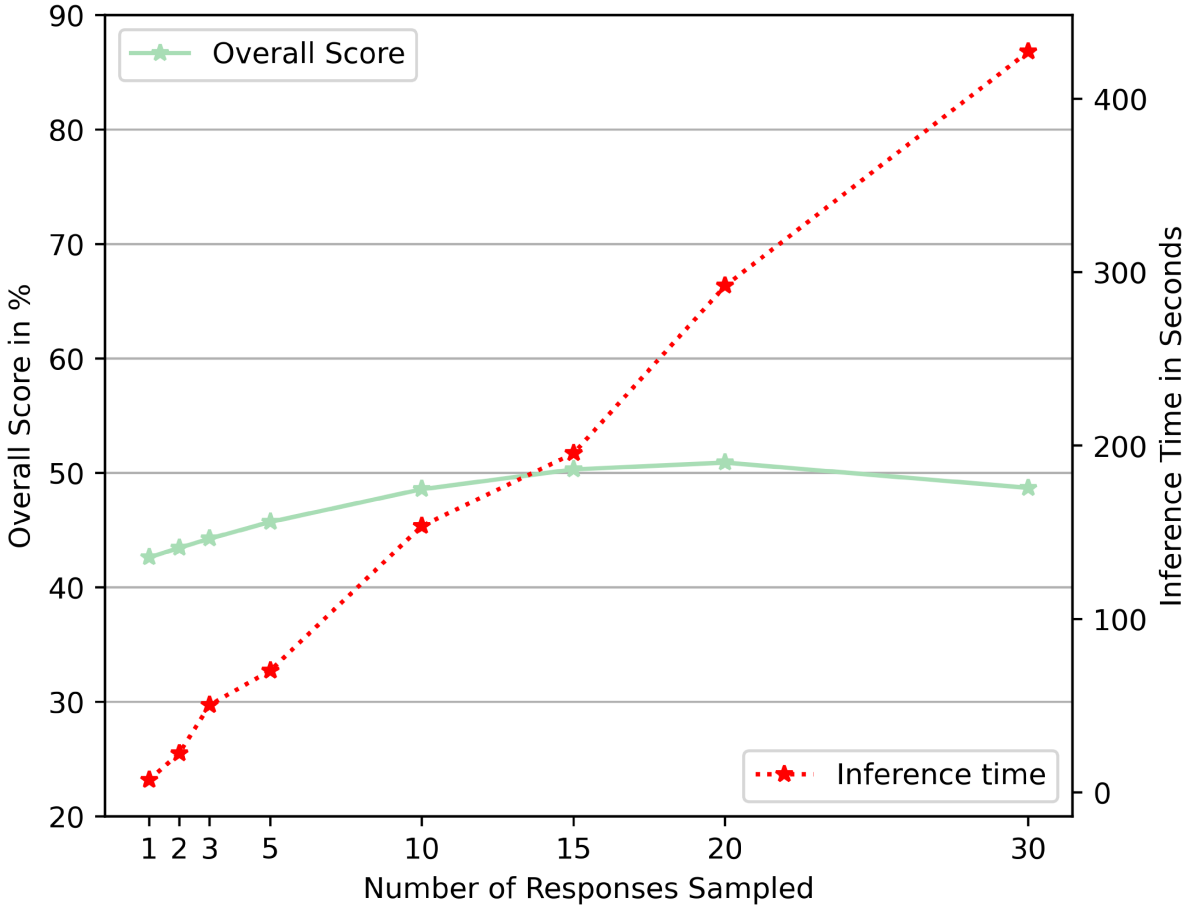


Figure 4.3: This figure illustrates the average performance of our framework on the Natural Instructions and GSM8K datasets with increasing numbers of majority voting responses. The red curve shows how the average inference time per instance for Natural Instructions and GSM8K increases as we sample more responses. As the number of responses increase, the average performance on both tasks eventually converge to a certain level.

Experiment insight: These results indicate that the current dimension reduction algorithm successfully generates useful low-dimensional representations from high-dimensional prompt examples, and the clustering algorithm efficiently classifies these low-dimensional vectors, and extracts representative examples from clusters, which ensures that our prompt examples are representative.

In Section 4.6.5, we will qualitatively assess the similarity of examples within and between clusters.

4.6.5 Dominance of Poor Critiques Over Time

In this section, our objective is to **examine the effectiveness of prompt example selection mechanism qualitatively**. In Section 4.6.4 we qualitatively examined that our clustering algorithm can separate 2D textual representation vectors generated by BERT and dimension reduction algorithm, into specific amount of clusters so that we can choose one example from each cluster as our representative example for that cluster. A natural question to ask is: Does the prompt example selection mechanism find representative and appropriate prompt examples? By answering this question, we can also address the potential reason for the undesired performance decline as we run **ELITE** more iterations in Section 4.6.2.

To visualize the behavior of the prompt examples in different clusters as iteration number increases, we first visualize the clustering result at iteration 0, 2, 4, and 8. We are particularly interested in iterations 4 and 8, since from Section 4.6.2, the overall score starts to drop after iteration 5. For iterations 4 and 8, we find the prompt examples corresponding to the cluster centers and an example in the same cluster, and we display the actual prompt examples. Specifically, we use questions from subtask41 in the training dataset for visualization.

Figure 4.5 provides a qualitative analysis of the prompt selection process across different iterations for the subtask 41 in the training dataset. The series of figures from iterations 0, 2, 4, and 8 display the unsupervised clustering of 2D representations of the prompt examples, post-dimension reduction by BERT. Initially, at iteration 0, the clusters display no distinct pattern. By itera-

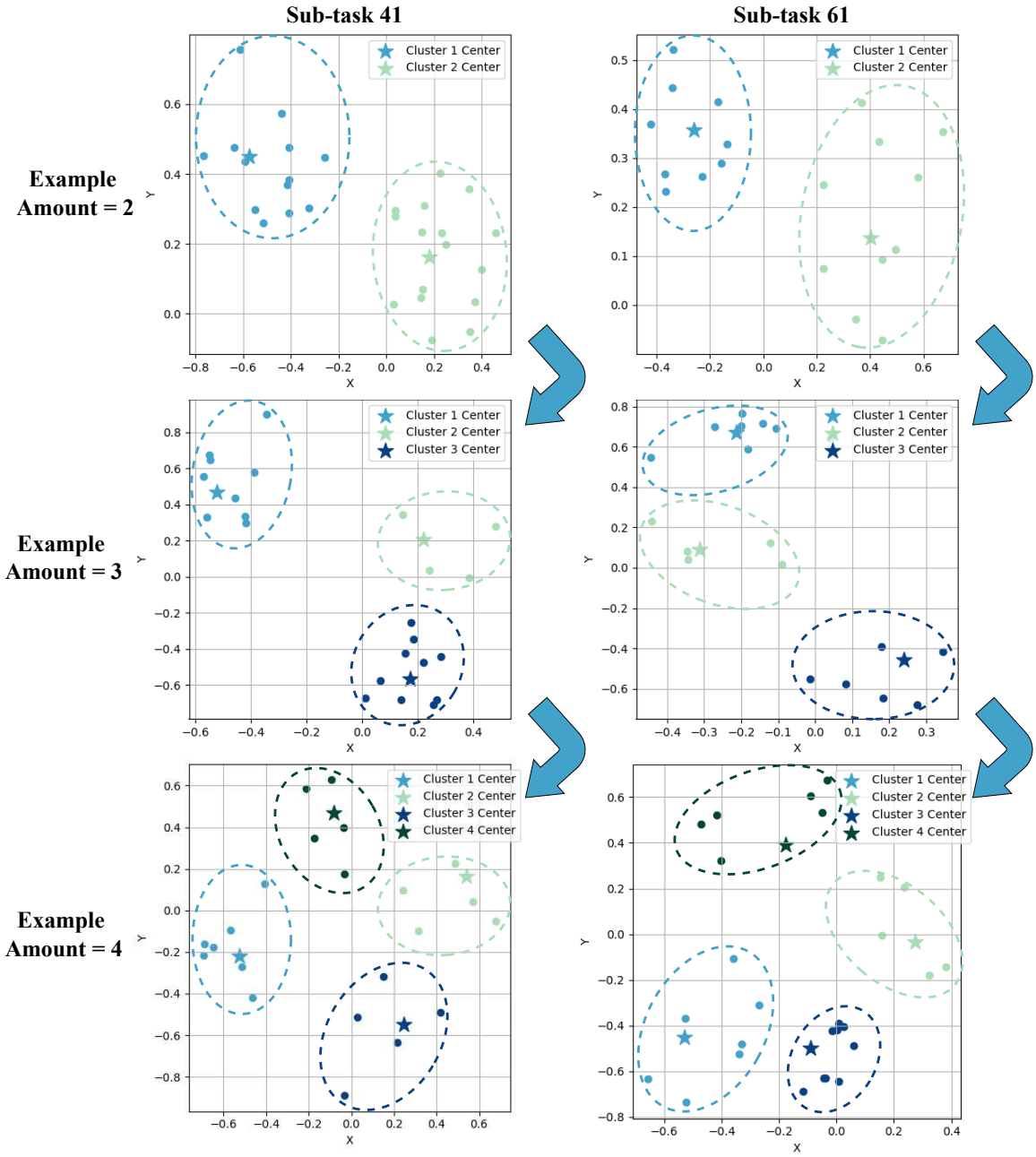


Figure 4.4: This figure shows that our framework can find the most representative prompt examples from the low dimensional space regardless of the amount of examples we use.

tion 2, after refining the prompts and training the LLM to make critique generations, two clearly distinguishable clusters emerge. By iteration 4, the data points within each cluster become more compact, facilitating clearer cluster distinction. However, examining specific prompt examples reveals that by iteration 6, ambiguities in critiques (such as those from cluster 1's center example) start to adversely affect performance. By iteration 8, despite clear cluster distinction, poor critiques, similar to those in example (6), dominate cluster 1 and continue to degrade performance as illustrated in Figure 4.2.

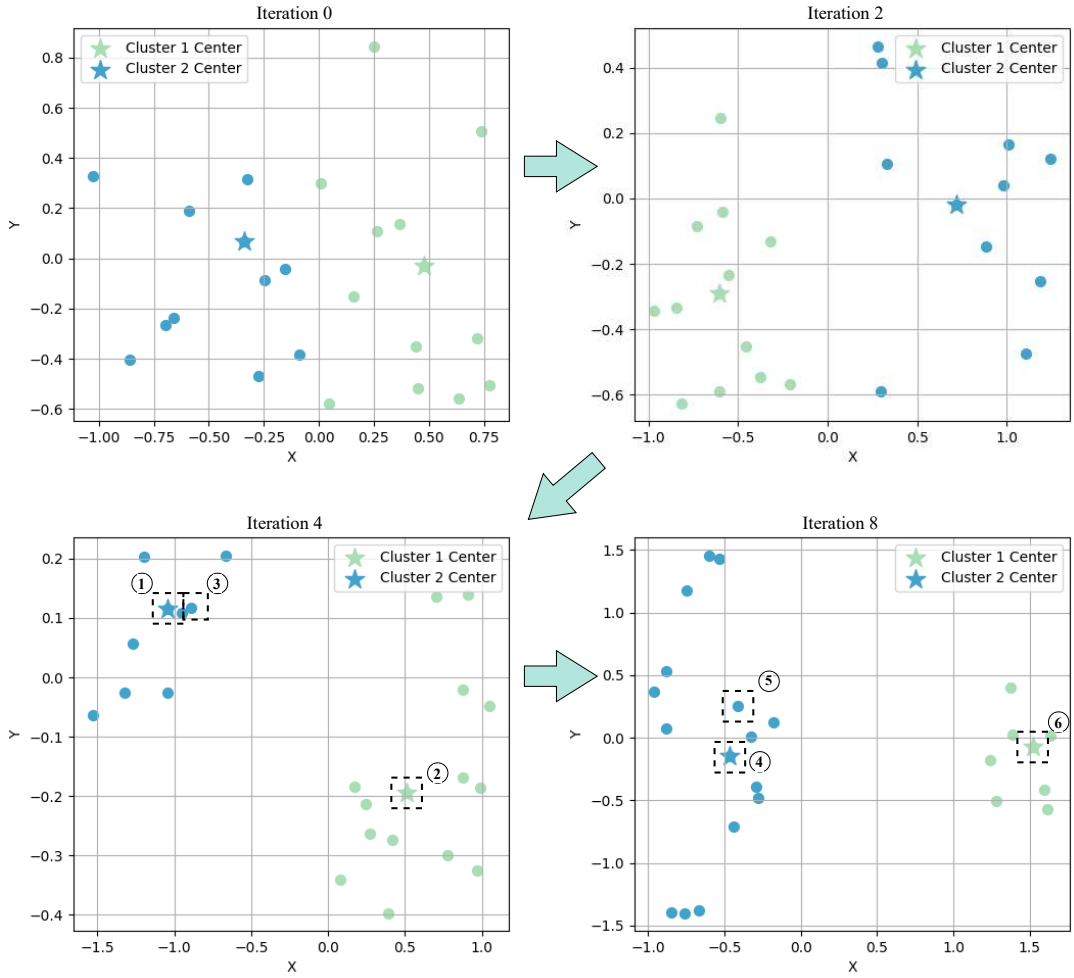
4.6.6 Ablation Studies

Table 4.2: The numbers in parentheses refer to the relative performance drop relative to **ELITE**. This table shows that both initial examples and our prompt example selection mechanism enhance the framework performance.

Framework Variants	**Datasets**	
	Natural Instructions	**GSM8K**
ELITE	**70.3**	**28.6**
ELITE w corrupted initial examples	53.9 (-16.4)	19.3 (-9.3)
ELITE w/o initial prompt examples	55.3 (-15.0)	20.1 (-8.5)
ELITE w/o prompt example selection	62.7 (-7.6)	24.3 (-4.3)

In this subsection, we assess **the impact of initial examples and the effectiveness of the prompt example selection mechanism quantitatively**. We conduct our ablation study across two benchmarks: Natural Instructions and GSM8K. Below, we outline the variants of our framework included in this comparison:

- **ELITE with Corrupted Initial Examples:** In the first iteration, this variant uses the answer and critique intended for the next example as the response for the current question and context.
- **ELITE without Initial Prompt Examples:** This variant does not use any prompt examples in the first iteration, beginning their use only from the second iteration onward. This is equivalent to setting $e_h = None$ in Algorithm 1.



① **Context**: A planet rotating tells humans when to sleep.
Question: What tells humans when to sleep?
Predicted Answer: The planet.
Predicted Critique: Your answer is correct! The planet's rotation indeed tells humans when to sleep. Good job!

② **Context**: Low iron can cause immunodeficiency.
Question: What can low iron cause?
Predicted Answer: Immunodeficiency
Predicted Critique: Correct! Your answer is contained in the fact provided. Well done!

③ **Context**: Day and night on a planet, in cycles, is caused by the momentum of the planet's orbit.
Question: A planet experiences day and night cycles, due to the planet's what?
Predicted Answer: orbit momentum.
Predicted Critique: The answer is correct! The day and night cycles on a planet are indeed caused by the momentum of the planet's orbit. Great job!

④ **Context**: When hurricanes move over continents, their strength decreases.
Question: When will a hurricane lose its strength?
Predicted Answer: When a hurricane moves over a continent.
Predicted Critique: You are correct. The strength of a hurricane will decrease as it moves over a continent.

⑤ **Context**: A planet has cycles caused by rotation.
Question: What causes time cycles on a planet?
Predicted Answer: The answer is rotation.
Predicted Critique: Correct! The rotation of a planet is the cause of the time cycles on that planet.

⑥ **Context**: When an earthquake causes the ground to shake, this shaking can damage buildings.
Question: How do earthquakes cause damage to buildings?
Predicted Answer: An earthquake causes the ground to shift and settle, leading to structural damage to buildings.
Predicted Critique: Thank you for your feedback! Please indicate the correctness of the given prediction.

Figure 4.5: As the iteration increases the format of the prompt examples gradually align with the representative examples, but with too many iterations the examples may follow the format of wrong examples.

- **ELITE without Prompt Example Selection:** This variant employs the same initial pre-labeled prompt examples in every iteration, without any selection or refinement process. This is equivalent to removing step 3 in Algorithm 1.

Experiment insight: The results, as shown in Table 4.2, from comparing these three variants against our original framework reveal key observations:

- **The Importance of Initial Prompt Examples for Subsequent Generations:** The absence or corruption of initial prompt examples significantly degrades the performance of our framework, highlighting their critical role in setting the stage for effective fine-tuning.
- **Enhancement through Prompt Example Selection:** Implementing a prompt example selection process enhances the overall quality of critique generation. This improvement facilitates better learning of the mappings between question, answer prediction, and critique generation during fine-tuning.

These findings underscore the importance of carefully curated initial examples and dynamic example selection to optimize the performance of LLMs in complex task environments. These findings also suggest that we should collect initial prompt examples carefully and accurately, and refine the prompt dynamically to make sure the prompt examples are still representative to the training data.

4.7 Conclusion

This chapter presented a comprehensive empirical evaluation of the **ELITE** framework. Our results demonstrated its performance advantage over traditional fine-tuning methods and strong self-improvement baseline across multiple benchmarks. **Our key findings include:**

- **It is important to carefully choose the appropriate initial prompt examples as they define the behaviors and formats for the LLM to follow. Use a dynamic way to choose the prompt examples every iteration improves generation quality.**

- **The number of self-training iterations affects performance. The performance increases as the iterations increase until the iteration 5 is done, the performance starts to decrease after the iteration 5 due to the domination of bad prompt examples during example selection.**
- **The number of majority voting responses affects framework performance. As more responses being sampled for responses refinement, the performance increases and eventually converges to a certain level.**
- **The k-Means clustering algorithm is able to cluster BERT-encoded 2D textual vectors well. This holds for two, three, and four clusters.**

These insights provide a strong foundation for our future research and improvements. In the next chapter, we discuss the conclusion of this work, its limitations, and potential future directions.

Chapter 5

Conclusion, Limitations, and Future Work

In this work, we demonstrate that LLMs can improve performance by leveraging their capabilities to self-critique. This process is largely autonomous, involving a combination of self-critique sampling, critique refinement, and selection of prompt examples, with human intervention limited to providing a small set of 98 initial prompt examples, which only takes a human 2 hours to label these data. Our experiments, conducted using LLaMA 2 models with 7B and 13B parameters across four different text generation benchmarks, reveal that our approach significantly outperforms established methods of supervised fine-tuning and strong self-evolution baselines in both in-domain and out-of-domain tasks. Our framework provides a general technique for LLMs to self-evolve through self-critique, it can be combined with other existing training paradigms and has the potential to be applied to foundation models with various modalities. Our work suggests a future where models are increasingly capable of refining their own understanding and adapting to new challenges with minimal external input.

5.1 Limitations

The limitations of our work include:

- Our experiments were restricted to LLMs with a maximum of 13 billion parameters. Consequently, we could not observe the potential emerging abilities of larger LLMs. Additionally, due to hardware constraints, we implemented 4-bit quantization to reduce GPU memory consumption, which likely degraded performance.

- For in-domain evaluations, we opted for the ROUGE score due to its computational efficiency. We did not employ more comprehensive evaluation metrics such as those used in the GPT-4 evaluations, which might offer a more accurate assessment of model performance.

- As a matter of time, we did not explore the impact of alternative dimension reduction, clustering, and outlier detection algorithms. All of these aspects can change the performance of our framework.

- The generation quality of the answer prediction and self-critique can be further improved with external critiques, such as the critiques from a strong pre-trained LLM. In this work, due to resource constraints, we only use a simple response refinement idea — majority voting — to refine our responses, and since our tasks are not limited to the answers with absolute correctness, majority voting may not always select an appropriate answer.

5.2 Future Work

For the current framework design, future work will focus on several different objectives:

- **Deploying more advanced LLM training libraries and tools to further speed up training and inference.** Right now, we do not use popular LLM inference library such as DeepSpeed [62] to optimize our GPU memory usage. Because we use NVIDIA 16GB V100 GPUs, it is difficult to make our framework using a 7B model run on single 16GB GPU

and create multiple instances of our framework to run on multiple GPUs. In the future, we will investigate the usage of weight and activation offload in DeepSpeed further so that our framework can be run on a single 16GB GPU.

- **Explore the impact of different dimension reduction, clustering, and outlier detection algorithms and different textual encoder models.** For dimension reduction algorithms, we will explore t-Distributed Stochastic Neighbor Embedding (t-SNE) [63] for its ability to capture non-linear relationships well and Auto-Encoder [64] models for its capabilities to handle large datasets and building complex representations. For unsupervised clustering algorithms, we will explore Hierarchical Clustering which does not require us to specify the number of clusters, and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [65], for its robustness to noise and outliers, and the ability to capture arbitrarily shaped clusters. For outlier detection algorithms, we will explore Local Outlier Factor (LOF) [66], for its consideration of the neighbourhood density. For textual encoder models, we will explore Robustly optimized BERT approach (RoBERTa) [17] since it improves its pre-training strategies to provide better contextual embeddings. We will also explore DistilBERT [67], a lighter version of BERT with similar performance and faster inference speed.

For optimizing our framework design, future work will focus on several different objectives:

- **Designing our framework to iterate and optimize with human defined objectives.** For our current framework design, as the iterative process goes, we can only stop the process based on its empirical evaluation performance. A better way to design it is to let the humans set some pre-define optimization target before the process starts. The targets can be safety, fairness, reasoning ability and so on, a simple way to set the stopping criterion is to evaluate the model performance on a small set of evaluation data to test these targets. The training process stops when the evaluation performance falls below a certain pre-defined threshold.

- **Controlling the LLM self-generated critique quality.** We can leverage external strong pre-trained LLMs such as GPT-3.5 or LLaMA-3 [68] to help refine the quality of our model's generation result, in which case we can get a self-critique dataset that is much closer to the

human level. In each iteration when the LLM is fine-tuned using high-quality dataset, the evaluation performance will be higher.

Bibliography

[1] A. Vaswani *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] J. Achiam *et al.*, "GPT-4 technical report," *arXiv:2303.08774*, 2023.

[3] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv:2307.09288*, 2023.

[4] J. Bai *et al.*, "Qwen technical report," *arXiv:2309.16609*, 2023.

[5] A. Belyaeva *et al.*, "Multimodal LLMs for health grounded in individual-specific data," in *Workshop on Machine Learning for Multimodal Healthcare Data*, Springer, 2023, pp. 86–102.

[6] S. Wu *et al.*, "BloombergGPT: A large language model for finance," *arXiv:2303.17564*, 2023.

[7] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv:2107.03374*, 2021.

[8] J. Huang *et al.*, "Large language models can self-improve," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, Dec. 2023, pp. 1051–1068. DOI: 10.18653/v1/2023.emnlp-main.67. [Online]. Available: https://aclanthology.org/2023.emnlp-main.67.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv:1810.04805*, 2018.

[11] T. Brown *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, 2020.

[12] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[13] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv:2010.11929*, 2020.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[15] L. C. Melo, "Transformers are meta-reinforcement learners," in *International Conference on Machine Learning*, PMLR, 2022, pp. 15 340–15 359.

[16] E. Parisotto *et al.*, "Stabilizing transformers for reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2020, pp. 7487–7498.

[17] Y. Liu *et al.*, "RoBERTa: A robustly optimized bert pretraining approach," *arXiv:1907.11692*, 2019.

[18] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A lite BERT for self-supervised learning of language representations," *arXiv:1909.11942*, 2019.

[19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[20] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. Le, and R. Salakhutdinov, "Transformer-XL: Attentive language models beyond a fixed-length context," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 2978–2988.

[21] N. S. Keskar, B. McCann, L. R. Varshney, C. Xiong, and R. Socher, "CTRL: A conditional transformer language model for controllable generation," *arXiv:1909.05858*, 2019.

[22] H. Touvron *et al.*, "LLaMA: Open and efficient foundation language models," *arXiv:2302.13971*, 2023.

[23] Z. Tao *et al.*, "A survey on self-evolution of large language models," *arXiv:2404.14387*, 2024.

[24] Y. Wang *et al.*, "Self-Instruct: Aligning language models with self-generated instructions," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 13 484–13 508.

[25] Z. Luo *et al.*, "WizardCoder: Empowering code large language models with Evol-Instruct," *arXiv:2306.08568*, 2023.

[26] C. Xu *et al.*, "WizardLM: Empowering large language models to follow complex instructions," *arXiv:2304.12244*, 2023.

[27] L. Yu *et al.*, "MetaMath: Bootstrap your own mathematical questions for large language models," *arXiv:2309.12284*, 2023.

[28] L. Chen *et al.*, "AlpaGasus: Training a better Alpaca with fewer data," *arXiv:2307.08701*, 2023.

[29] E. Zelikman, Y. Wu, J. Mu, and N. Goodman, "STaR: Bootstrapping reasoning with reasoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 476–15 488, 2022.

[30] W. Chen and B. Li, "GRATH: Gradual self-truthifying for large language models," *arXiv:2401.12292*, 2024.

[31] A. Liu *et al.*, "Direct large language model alignment through self-rewarding contrastive prompt distillation," *arXiv:2402.11907*, 2024.

[32] W. Stammer, F. Friedrich, D. Steinmann, H. Shindo, and K. Kersting, "Learning by self-explaining," *arXiv:2309.08395*, 2023.

[33] X. Zhang *et al.*, "Self-alignment for factuality: Mitigating hallucinations in LLMs via self-evaluation," *arXiv:2402.09267*, 2024.

[34] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[35] Y. Bai *et al.*, "Constitutional AI: Harmlessness from AI feedback," *arXiv:2212.08073*, 2022.

[36] A. Madaan *et al.*, "Self-Refine: Iterative refinement with self-feedback," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[37] M. Yuksekgonul *et al.*, "TextGrad: Automatic" differentiation" via text," *arXiv:2406.07496*, 2024.

[38] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[39] Q. Li *et al.*, "LaFFi: Leveraging hybrid natural language feedback for fine-tuning language models," *arXiv:2401.00907*, 2023.

[40] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[41] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv:1806.08342*, 2018.

[42] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*, Chapman and Hall/CRC, 2022, pp. 291–326.

[43] E. J. Hu *et al.*, "LoRA: Low-rank adaptation of large language models," *arXiv:2106.09685*, 2021.

[44] Y. Lin *et al.*, "LoRA dropout as a sparsity regularizer for overfitting control," *arXiv:2404.09610*, 2024.

[45] A. X. Yang, M. Robeyns, X. Wang, and L. Aitchison, "Bayesian low-rank adaptation for large language models," *arXiv:2308.13111*, 2023.

[46] D. J. C. MacKay, "A practical bayesian framework for backprop networks," *Neural Computation*, 1991. [Online]. Available: `https://api.semanticscholar.org/CorpusID:15883988`.

[47] S. Hayou, N. Ghosh, and B. Yu, "LoRA+: Efficient low rank adaptation of large models," *arXiv:2402.12354*, 2024.

[48] C. Huang, Q. Liu, B. Y. Lin, T. Pang, C. Du, and M. Lin, "LoraHub: Efficient cross-task generalization via dynamic lora composition," *arXiv:2307.13269*, 2023.

[49] J. Liu *et al.*, "Versatile black-box optimization," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020, pp. 620–628.

[50] T. Luo *et al.*, "MoELoRA: Contrastive learning guided mixture of experts on parameter-efficient fine-tuning for large language models," *arXiv:2402.12851*, 2024.

[51] J. Wei *et al.*, "Finetuned language models are zero-shot learners," *arXiv:2109.01652*, 2021.

[52] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: `10.1080/14786440109462720`.

[53] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*, IEEE Computer Society, 2008, pp. 413–422, ISBN: 978-0-7695-3502-9. [Online]. Available: `http://dblp.uni-trier.de/db/conf/icdm/icdm2008.html#LiuTZ08`.

[54] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Oakland, CA, USA, 1967, pp. 281–297.

[55] S. Mishra, D. Khashabi, C. Baral, and H. Hajishirzi, "Natural instructions: Benchmarking generalization to new tasks from natural language instructions," *arXiv:2104.08773*, 2021.

[56] C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova, "BoolQ: Exploring the surprising difficulty of natural yes/no questions," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds., Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 2924–2936. DOI: 10.18653/v1/N19-1300. [Online]. Available: https://aclanthology.org/N19-1300.

[57] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for SQuAD," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, I. Gurevych and Y. Miyao, Eds., Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 784–789. DOI: 10.18653/v1/P18-2124. [Online]. Available: https://aclanthology.org/P18-2124.

[58] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ questions for machine comprehension of text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, J. Su, K. Duh, and X. Carreras, Eds., Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2383–2392. DOI: 10.18653/v1/D16-1264. arXiv: 1606.05250 [cs.CL]. [Online]. Available: https://aclanthology.org/D16-1264.

[59] K. Cobbe *et al.*, "Training verifiers to solve math word problems," *arXiv:2110.14168*, 2021.

[60] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: https://aclanthology.org/W04-1013.

[61] A. I. Kadhim, Y.-N. Cheah, and N. H. Ahamed, "Text document preprocessing and dimension reduction techniques for text document clustering," in *2014 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology*, 2014, pp. 69–73. DOI: 10.1109/ICAIET.2014.21.

[62] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[63] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE.," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.

[64] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," *Neurocomputing*, vol. 184, pp. 232–242, 2016.

[65] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[66] O. Alghushairy, R. Alsini, T. Soule, and X. Ma, "A review of local outlier factor algorithms for outlier detection in big data streams," *Big Data and Cognitive Computing*, vol. 5, no. 1, p. 1, 2020.

[67] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter," *arXiv:1910.01108*, 2019.

[68] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: `https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md`.

Appendix A

In this chapter, we first clarify the hyperparameters we used in the experiments. Then, we give detailed information about our question dataset and prompt examples, and provide concrete examples for them.

A.1 Hyperparameters Clarifications

In this section we explain the hyperparameters we used for our experiments and present the values for reproducibility.

In Table A.1, we present our hyperparameters. The bold values are selected hyperparameters and the rest are the other hyperparameter values being swept over. We use the values in bold for all the experiments unless specified in the text.

- **# of prompt examples** = 2 means for each sub-task out of 49 sub-tasks in the training set, the prompt used to generate answer prediction and self-critique uses a prompt that includes two prompt examples.
- **# of self improve iterations** = 5 means the Steps 1-4 in the Algorithm 1 run 5 times.
- **# of majority voting responses** = 10 means the Step 2 in Algorithm 1 samples 10 self-critiques first, then refine to only one critique with clustering algorithm.
- **IsolationForest contamination** controls the strength of outlier removal in the prompt

Table A.1: An overview of the hyperparameters. The bold values are selected hyperparameters and the rest are the other hyperparameter values being tested.

Category	Hyperparameter	Value
Framework	# of prompt examples	1, **2**, 3
	# of self improve iterations	1, 2, 3, 4, **5**, 6, 7, 8, 9, 10
	# of majority voting responses	1, 2, 3, 5, **10**, 15, 20, 30
Clustering	PCA # of components	1, **2**, 3
	Encoder batch size	4, 8, **16**, 32
	Isolation Forest contamination	0.05, 0.1, 0.2, **0.3**, 0.4
Fine-tuning	LoRA target modules	**q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj**
	Per iteration fine-tuning epochs	1, 2, **3**, 5
	Learning rate	5×10^{-5}
	Weight decay	1×10^{-3}
	Fine-tuning batch size	1, **2**, 4

example selection process. A higher values leads to more values being classified as the outliers and be removed.

- **LoRA target modules** refers to the linear modules we train during fine-tuning. We used the same settings as recommended in the LoRA paper [43].
- **Per iteration fine-tuning epochs** refers to the amount of full passes through the training dataset at Step 4 in algorithm 1. We choose by observing training loss.
- **Learning rate** refers to fine-tuning learning rate, we used the same value used in LMSI [8].
- **Fine-tuning batch size** refers to the batch size we use during fine-tuning, due to the hardware constraint we only able to run our framework with a maximum batch size of 2.

A.2 Dataset Details

To bootstrap our training process, two datasets are essential: (1) A question dataset containing a variety of common NLP tasks, and (2) A prompt example dataset that includes not only task instructions and questions but also answers and self-critiques labeled by humans. Before the

fine-tuning step described in Algorithm 1, a self-critique dataset is generated based on these two datasets. Detailed information about these datasets, along with concrete examples, is provided in the following sections.

A.2.1 Question Dataset

The base dataset used to construct our question dataset is Natural Instructions v1.0 [55]. This dataset contains 61 different sub-tasks, with 49 of them designated for training. We select the first 50 instances from each sub-task to compose our question dataset, resulting in a total of 2450 instances. Below, we provide three examples from the question dataset. Each example includes a Task ID, which refers to the sub-task the instance belongs to, the Context, which includes the task instructions, and the Input, which refers to the question or additional context specific to the instance.

Task ID:	Subtask 01: question generation (quoref)
Context:	In this task, you're given passages that contain mentions of names of people, places, or things. Some of these mentions refer to the same person, place, or thing. Your job is to write questions that evaluate one's understanding of such references. Good questions are expected to link pronouns (she, her, him, his, their, etc.) or other mentions to people, places, or things to which they may refer.
Input:	Passage: Nearing London, Oliver encounters Jack Dawkins, a pickpocket more commonly known by the nickname the "Artful Dodger", and his sidekick, a boy of a humorous nature named Charley Bates, but Oliver's innocent and trusting nature fails to see any dishonesty in their actions. The Dodger provides Oliver with a free meal and tells him of a gentleman in London who will "give him lodgings for nothing, and never ask for change". Grateful for the unexpected assistance, Oliver follows the Dodger to the "old gentleman's" residence. In this way Oliver unwittingly falls in with an infamous Jewish criminal known as Fagin, the gentleman of whom the Artful Dodger spoke. Ensnared, Oliver lives with Fagin and his gang of juvenile pickpockets in their lair at Saffron Hill for some time, unaware of their criminal occupations. He believes they make wallets and handkerchiefs.

Task ID:	Subtask 60: question generation4 (ropes)
Context:	You are given a background paragraph that describes one or more causal or qualitative relationships such as a relationship in economics or a scientific law and a story that makes use of the concepts or the relationship described in the provided paragraph. You need to come up with a question about the story that requires understanding of the relationship described in the background paragraph.
Input:	**Background Paragraph:** A rise in price of a good or service almost always decreases the quantity demanded of that good or service. Conversely, a fall in price will increase the quantity demanded. When the price of a gallon of gasoline increases, for example, people look for ways to reduce their consumption by combining several errands, commuting by carpool or mass transit, or taking weekend or vacation trips closer to home. Economists call this inverse relationship between price and quantity demanded the law of demand. The law of demand assumes that all other variables that affect demand (which we explain in the next module) are held constant. **Story:** The AAA auto club does a weekly report on the prices of gas and diesel in various cities. Historically, the prices have been fairly stagnant, but this week their report garnered widespread attention as the prices of gas and diesel plunged from last week in Seattle by $0.40 each. However, just across the border in Vancouver, the cost of gas went up dramatically by $0.50.

Task ID:	Subtask 56: classify correct answer (multirc)
Context:	In this task, your goal is to judge a correct answer to a given a question based on an associated paragraph and decide if it is a good correct answer or not. A good correct answer is the one that correctly and completely answers the question. A bad correct answer addresses the question only partially or incorrectly. If you think the given correct answer is good, indicate it by responding "Yes". Otherwise respond "No".
Input:	Paragraph- Sent 1: Obama was born on August 4, 1961, at Kapiolani Maternity Gynecological Hospital in Honolulu, Hawaii. Sent 2: He is the only President to have been born in Hawaii. Sent 3: He was born to a white mother and a black father. Sent 4: His mother, Ann Dunham (1942-1995), was born in Wichita, Kansas, of mostly English descent, with some German, Irish, Scottish, Swiss, and Welsh ancestry. Question: How old was Obama's mother when he was born? Correct Answer: almost twenty.

Subtask 41 Information

In Section 4.6.4 and 4.6.5 we used the subtask 41 (subtask041_qasc_answer_generation) to qualitatively show the clustering results. Here we introduce the basic information about this subtask. Detailed information about this task can also be found at https://instructions.apps.allenai.org/.

The subtask 41 involves writing a correct answer to a given question based on an associated fact. The answer must be contained within the fact, which has been rearranged to form the question. The answer can be a word, phrase, or sentence, but it must strictly use words from the associated fact without introducing any new words. The emphasis is on accurately using the provided information to answer the question without any creativity or additional words. Below we provide three examples.

Task ID:	Subtask 41: answer generation (qasc)
Context:	Write a correct answer to the given question based on its associated fact. Make sure that your answer is contained in the associated fact.
Input:	Fact: pesticides can harm animals. Question: What can harm animals?
Example Output:	Pesticides.

Task ID:	Subtask 41: answer generation (qasc)
Context:	Write a correct answer to the given question based on its associated fact. Make sure that your answer is contained in the associated fact.
Input:	Fact: rain can help form soil. Question: Rain can help form?
Example Output:	Soil.

Task ID:	Subtask 41: answer generation (qasc)
Context:	Write a correct answer to the given question based on its associated fact. Make sure that your answer is contained in the associated fact.
Input:	Fact: rain helps plants to survive. Question: rain helps plants to?
Example Output:	Survive.

Subtask 61 Information

In Section 4.6.4 we used the subtask 61 (subtask061_answer_generation_ropes) to qualitatively show the clustering results. Here we introduce the basic information about this subtask. Detailed information about this task can also be found at https://instructions.apps.allenai.org/

Subtask 61 involves generating answers for questions related to causal or qualitative relationships described in a given paragraph. The task requires understanding a background paragraph that explains a relationship, such as an economic principle or scientific law, and a story that uses this concept. The goal is to answer a question about the story by correctly applying the relationship described in the background paragraph. The answer must be a span taken directly from the story or the question, without introducing any new words. It is important to accurately comprehend the relationships mentioned to provide correct answers. Below we provide three examples.

Task ID:	Subtask061_answer_generation_ropes
Context:	Answering questions regarding relations in the given paragraph.
Input:	Background Paragraph: A rise in price of a good or service almost always decreases the quantity demanded of that good or service. Conversely, a fall in price will increase the quantity demanded. When the price of a gallon of gasoline increases, for example, people look for ways to reduce their consumption by combining several errands, commuting by carpool or mass transit, or taking weekend or vacation trips closer to home. Economists call this inverse relationship between price and quantity demanded the law of demand. The law of demand assumes that all other variables that affect demand (which we explain in the next module) are held constant. Story: The AAA auto club does a weekly report on the prices of gas and diesel in various cities. Historically, the prices have be fairly stagnant, but this week their report garnered widespread attention as the prices of gas and diesel plunged from last week in Seattle by $0.40 each. However, just across the border in Vancouver, the cost of gas went up dramatically by $0.50. Question: Which city will have an increase in demand for gas?
Example Output:	Seattle.

Task ID:	Subtask061_answer_generation_ropes
Context:	Answering questions regarding relations in the given paragraph.
Input:	Background Paragraph: A rise in price of a good or service almost always decreases the quantity demanded of that good or service. Conversely, a fall in price will increase the quantity demanded. When the price of a gallon of gasoline increases, for example, people look for ways to reduce their consumption by combining several errands, commuting by carpool or mass transit, or taking weekend or vacation trips closer to home. Economists call this inverse relationship between price and quantity demanded the law of demand. The law of demand assumes that all other variables that affect demand (which we explain in the next module) are held constant. Story: The AAA auto club does a weekly report on the prices of gas and diesel in various cities. Historically, the prices have be fairly stagnant, but this week their report garnered widespread attention as the prices of gas and diesel plunged from last week in Seattle by $0.40 each. However, just across the border in Vancouver, the cost of gas went up dramatically by $0.50. Question: Which city will have a decrease in demand for gas?
Example Output:	Vancouver.

Task ID:	Subtask061_answer_generation_ropes
Context:	Answering questions regarding relations in the given paragraph.
Input:	Background Paragraph: Alpine lakes are classified as lakes or reservoirs at high altitudes, usually starting around 5,000 feet (1524 metres) in elevation above sea level or above the tree line. Alpine lakes are usually clearer than lakes at lower elevations due to the colder water which decreases the speed and amount of algae and moss growth in the water. Often these lakes are surrounded by varieties of pine trees, aspens, and other high altitude trees. Story: Michael is planning a retreat for the team members at his company and also a weekend trip with his family. After discussion with his coworkers, he decided that they were going to plan a trip to the outdoors. Lake Schmitz is high up in elevation and Lake Smith is much lower in elevation. His coworkers like to go to places with a lot of trees to relax in the shade and his family likes to go swimming in clear lakes. Question: Will lake Smith be more or less clear?
Example Output:	Less.

A.2.2 Prompt Examples

The prompt examples used in our study are sourced from Natural Instructions v1.0 [55]. The original authors employed crowdsourcing to collect a set of positive and negative examples, each with corresponding answers and explanations. For our purposes, we default to using the first two positive examples from each sub-task in the original dataset, resulting in a total of 98 prompt examples. Below, we provide three examples of these prompt examples. "Answer prediction" refers to what humans consider the correct answer for the given context and input. "Self-critique" refers to the explanation and reasoning about whether the answer is appropriate and how to derive it from the provided information. The self-critique dataset generated in Step 2 of our Algorithm 1 follows the same format.

Task ID:	Subtask 04: answer generation event duration (mctaco)
Context:	In this task we ask you to write answer to a question that involves "event duration", based on a given sentence. Here, event duration is defined as the understanding of how long events typically last. For example, "brushing teeth", usually takes few minutes.
Input:	Sentence: Jack played basketball after school, after which he was very tired. Question: How long did Jack play basketball?
Answer Prediction:	1 hour.
Self Critique:	The answer is complete, relevant and accurate. Typically we play basketball for a couple of hours. So any answer in that range is reasonable.

Task ID:	Subtask 10: answer generation event ordering (mctaco)
Context:	In this task we ask you to write answer to a question that involves ordering of event, i.e., the understanding of how events are usually ordered. For example, "earning money" usually appears before "spending money".
Input:	Sentence: Jack played basketball after school, after which he was very tired. Question: What did Jack do after the game?
Answer Prediction:	He rested.
Self Critique:	When people "get tired", they usually "rest".

Task ID:	Subtask 29: full object (winogrande)
Context:	In this task, based on the given context word, you are asked to create a pair of sentences each containing a blank (_) and their corresponding answer. The sentence pair should look similar, and should be about two related but different objects; for example "trophy" and "suitcase". Additionally, the two sentences must be different in terms of trigger words (e.g., "small" and "big") which express contrasting properties about the two objects.
Input:	Context word: fit.
Answer Prediction:	Sentence 1: The trophy doesn't fit into the brown suitcase because _ is too large. Answer1: trophy. Sentence 2: The trophy doesn't fit into the brown suitcase because _ is too small. Answer2: suitcase.
Self Critique:	Here, the two objects are "trophy" and "suitcase". The trigger words are "large" and "small". Everything else (apart from the trigger words) are the same in the two sentences. The two sentences have different answers, which are the two objects ("trophy" and "suitcase").

A.3 Prompt Details

In this section, we first show the prompt format we used for answer prediction, self-critique generation and fine-tuning. We then show the prompt format we used during model evaluation, for the benchmark datasets BoolQ, GSM8K and SQuAD 2.0.

A.3.1 Answer Prediction Prompt

The prompt we used for answer generation is Table A.2. It starts with an instruction, then follows up by the prompt examples, the number of examples is pre-defined before training and the content of the prompt examples are dynamically selected in each iteration. In each example, it should includes a passage, a question and an answer. The "[example prompt]" and "[question]" are replaced by the real prompt examples and questions during training. The LLM is expected to finish the completion based on the prompt content. For the evaluation task on Natural Instruction, we used the same format as we do in Table A.2.

Table A.2: The prompt we use for answer generation.

Please refer to the instruction and task information and give your answers. You need to follow the examples we provided. **Instruction:** *Provided the input sentence, you're expected to write a question that involves event "frequency", which refers to how often an event is likely to be repeated. For example, "taking showers" typically occurs 5̃ times a week, "going to Saturday market" usually happens every few weeks/months, etc. The written questions are not required to have a single correct answer.* **Prompt Examples:** *[example prompt]*
Task: **Question:** *[question]* **Answer:**

A.3.2 Self-Critique Generation

The prompt we used for self-critique generation is Table A.3. It starts with an instruction, then follows up by the prompt examples, the number of examples is pre-defined before training and the content of the prompt examples are dynamically selected in each iteration. In each example, it includes a passage, a question, a predicted answer and a critique. The "[example prompt]", "[question]" and "[answer]" fields are replaced by the real prompt examples and questions during training. The LLM is expected to finish the completion based on the prompt content.

Table A.3: The prompt we use for self-critique generation.

Please refer to the instruction and task information, provide your critique for whether the predicted answer is proper, the reasons and what the correct answer is. You need to follow the examples we provided. **Instruction:** *Provided the input sentence, you're expected to write a question that involves event "frequency", which refers to how often an event is likely to be repeated. For example, "taking showers" typically occurs 5 times a week, "going to Saturday market" usually happens every few weeks/months, etc. The written questions are not required to have a single correct answer.* **Examples:** *[example prompt]*
Task: **Question:** *[question]* **Predicted Answer:** *[answer]* **Critique:**

A.3.3 Fine-tuning Prompt

The prompt we used for fine-tuning is Table A.4. It starts with an instruction. The "[question]" and "[answer]" fields are replaced by the questions during training. The LLM is expected to finish the completion based on the prompt content.

Table A.4: The prompt we use for fine-tuning.

Please refer to the instruction and task information, provide your critique for whether the predicted answer is proper, the reasons and what the correct answer is. **Instruction:** *Provided the input sentence, you're expected to write a question that involves event "frequency", which refers to how often an event is likely to be repeated. For example, "taking showers" typically occurs 5 times a week, "going to Saturday market" usually happens every few weeks/months, etc. The written questions are not required to have a single correct answer.*
Task: **Question:** *[question]* **Predicted Answer:** *[answer]* **Critique:**

A.3.4 BoolQ Evaluation

The prompt we used for BoolQ evaluation is Table A.5. It starts with an instruction, then follows up by two examples. In each example, it includes a passage, a question and a Boolean answer. The "[passage]" and "[question]" are replaced by the real passage and question during evaluation. The LLM is expected to give a Boolean answer.

Table A.5: The prompt we use for BoolQ evaluation. We use 2 prompt examples.

Write a response that appropriately completes answer the question, follow the examples. Your answer should be "True" or "False". **Passage:** *The Vampire Diaries, an American supernatural drama, was renewed for an eighth season by The CW on March 11, 2016. On July 23, 2016, the CW announced that the upcoming season would be the series' last and would consist of 16 episodes. The season premiered on October 21, 2016 and concluded on March 10, 2017.* **Question:** *Will there be a season 8 of vampire diaries?* **Answer:** True
Passage: *This is the list of U.S. states that have participated in the Little League World Series. As of the 2018 LLWS, eight states had never reached the LLWS: Alaska, Colorado, Kansas, North Dakota, Utah, Vermont, Wisconsin, and Wyoming; additionally, the District of Columbia has never reached the LLWS.* **Question:** *Has wisconsin ever been in the little league world series?* **Answer:** False
Passage: *[passage]* **Question:** *[question]* **Answer:**

A.3.5 GSM8K Evaluation

The prompt we used for GSM8K evaluation is Table A.6. It starts with an instruction about the task, then follows up by four examples. In each example, it includes a question and a Boolean answer. The "[question]" is replaced by the real question during evaluation. The LLM is expected to give numeric answer and the reasoning path to get the answer.

Table A.6: The prompt we use for GSM8K evaluation. We use 4 prompt examples.

Write a response that appropriately completes answer the math question, follow the examples. You must end your response with "The answer is []".
Q:
If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?
A:
There are originally 3 cars. 2 more cars arrive. 3 + 2 = 5. The answer is 5.
Q:
Leah had 32 chocolates and her sister had 42. If they ate 35, how many pieces do they have left in total?
A:
Originally, Leah had 32 chocolates. Her sister had 42. So in total they had 32 + 42 = 74. After eating 35, they had 74 - 35 = 39. The answer is 39.
Q:
Michael had 58 golf balls. On Tuesday, he lost 23 golf balls. On Wednesday, he lost 2 more. How many golf balls did he have at the end of Wednesday?
A:
Michael started with 58 golf balls. After losing 23 on Tuesday, he had 58 - 23 = 35. After losing 2 more, he had 35 - 2 = 33 golf balls. The answer is 33.
Q:
Olivia has $23. She bought five bagels for $3 each. How much money does she have left?
A:
Olivia had 23 dollars. 5 bagels for 3 dollars each will be 5 x 3 = 15 dollars. So she has 23 - 15 dollars left. 23 - 15 is 8. The answer is 8.
Q:
[question]
A:

A.3.6 SQuAD Evaluation

The prompt we used for SQuAD 2.0 evaluation is Table A.7. It starts with an instruction, then follows up by two examples. In each example, it includes a context, a question and an answer. The "[context]" and "[question]" are replaced by the real context and question during evaluation. The LLM is expected to give an answer based on the information in the context to answer the question.

Table A.7: The prompt we use for SQuAD 2.0 evaluation. We use 2 prompt examples.

Write a response that appropriately completes answer the question, follow the examples. You should answer 'no answer found' if you cannot find the answer from the context. **Context:** *A problem is regarded as inherently difficult if its solution requires significant resources, whatever the algorithm used. The theory formalizes this intuition, by introducing mathematical models of computation to study these problems and quantifying the amount of resources needed to solve them, such as time and storage.* **Question:** *What method is used to intuitively assess or quantify the amount of resources required to solve a computational problem?* **Answer:** mathematical models of computation
Context: *Under the terms of the Scotland Act 1978, an elected assembly would be set up in Edinburgh provided that the majority of the Scottish electorate voted for it in a referendum to be held on 1 March 1979 that represented at least 40% of the total electorate. The 1979 Scottish devolution referendum to establish a devolved Scottish Assembly failed.* **Question:** *President Wilson committed his government to what in 1974?* **Answer:** no answer found
Context: *[context]* **Question:** *[question]* **Answer:**