Do the LLMs Know What/Where and Why They Lack?

Marina Igitkhanian

Supervisor: Dr. Erik Arakelyan

Submitted for the degree of

BSc in Data Science
College of Science and Engineering
American University of Armenia

2025

Abstract

Recent advances in Large Language Models (LLMs) have changed the modern world, affecting a wide range of industries. Despite their transformative power, LLMs still make mistakes. One proposed solution is to have the LLM generate feedback on its own outputs and then use it for self-correction. This approach relies on the idea that LLMs possess the ability to self-reflect. However, those capabilities are questioned in the AI community, and more validation is needed. In this study, we evaluate instruction-tuned LLMs on arithmetic reasoning benchmarks ASDiv and GSM8K by (1) collecting initial answers, (2) prompting the same model to generate hints for its incorrect responses (given the ground truth), and (3) measuring improvement after re-inference with hints sentences. As a result of our research, injection of those hint sentences corrected only 20% of initially wrong responses. This shows that current LLMs still struggle to pinpoint what they got wrong, where, and why, limiting the effectiveness of self-feedback as a standalone refinement strategy.

Acknowledgments

I would like to extend sincere thanks to my supervisor, Mr. Erik Arakelyan, for invaluable support and guidance throughout this capstone project. It was a privilege to collaborate with such a dedicated professional and to learn from his breadth of knowledge. Not only did I successfully complete my capstone project, but thanks to him, I was also introduced to the world of Natural Language Processing - a field that sparked my interest and where I am eager to continue exploring.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This section provides a brief overview of the problem setting and analyzes underlying motivations.

1.1 Context and Motivation

Large Language Models (LLMs) are advanced AI systems specialized in natural language processing, understanding, and generation. They have achieved remarkable performance, compared to the previous state-of-the-art (SOTA) models in multiple complex tasks, requiring multi-step reasoning.

However, LLMs are not perfectly accurate and still produce factual or logical mistakes leading to incorrect outputs. There are several approaches to mitigate that issue, one being the concept of self-refine proposed by Madaan et al. [1]. It is an iterative algorithm that aims to refine the LLM's initial answer based on feedback given by the same LLM. This method relies entirely on the model's existing knowledge and doesn't require supervision or fine-tuning, making it an efficient way of improving LLMs' performance. On the other hand, some researchers propose self-correction methods for LLMs in which the generation of feedback sentences relies on high-quality external information [2]. Both approaches suggest that LLMs demonstrate the ability to identify what and where they lack, since the models must self-reflect on their initial mistakes to generate sentences that can refine their original answers.

1.2 Problem Statement

The effectiveness of self-refinement as a method for improving model performance remains debated within the AI research community. According to Huang et al. [3], LLMs struggle to self-correct their responses without external feedback. Moreover, the performance of the model after its revision degrades.

Based on this background, we formulate our core research question: Do the LLMs know what, where, and why they lack? If such awareness exists, then at the very least, when the model initially predicts wrong information and then is provided with the correct answer, it should be able to generate effective feedback or hint sentences that help it answer the question correctly in the first place. Evaluation of LLM performance before and after injecting its own feedback sentences would reveal insights into LLMs' self-refinement capability.

1.3 Research Objectives

This paper investigates the self-refinement capabilities of LLMs in the context of arithmetic reasoning. Our approach doesn't rely solely on the LLM's knowledge but uses a supervised approach, by providing the model with oracle labels (correct answers) when requesting feedback.

The objective is to create an automated pipeline which evaluates changes in accuracy after applying the three-step self-refine method [1] with a modification in step two, where the model is only given the questions it answered incorrectly, along with the correct answers and is asked to generate feedback. This feedback, framed as an "answer-free" (corrective suggestion that omits the true answer itself) hint, is then appended to the original prompt as additional context, and the model is given a second opportunity to solve the same question.

By testing two instruction-tuned models, Gemma-2-2b-it and Phi-4-mini-instruct, on the ASDiv and GSM8K benchmarks, we aim to measure the proportion of answers which were correctly answered after injection of the hint sentences. Through rigorous evaluation, we seek to answer whether LLMs are capable of recognizing and responding to their own limitations.

Our contributions are summarized as:

- We propose an approach to test self-refinment capabilities of LLMs that uses an

extrinsic approach by using oracle labels during hint sentences generation and an **intrinsic** approach by re-feeding the model original question with its generated feedback as additional context.

- We develop an end-to-end, fully automated and reproducible self-refinement evaluation pipeline and release publicly available code and prompt templates to support further research across diverse datasets and alternative LLMs.
- We provide empirical evidence highlighting the limitations of LLMs to generate useful feedback that could guide them toward performance improvements.

1.4 Structure of the paper

In this chapter, we briefly reviewed the core challenge examined in the paper by introducing the context and problem statement. In Chapter 2, we review the foundational concepts required to fully understand this study. Chapter 3 then outlines our overall research approach and methodological framework. Afterwards, Chapter 4 presents the experiments and results. Finally, Chapter 5 summarizes our findings, draws conclusions, and outlines directions for future research.

Chapter 2

Background

2.1 Transformers

The Transformer architecture, introduced by Vaswani et al. [4], is the foundational mechanism for modern LLMs. Its key characteristic is that it processes the textual data without relying on recurrence or convolution. Instead, it utilizes a mechanism called self-attention, which models relationships between tokens in a sequence in parallel.

2.1.1 Model Architecture

The model follows an encoder–decoder structure. The encoder maps an input sequence $(x_1, \ldots, x_n)$ to a sequence of continuous representations $z = (z_1, \ldots, z_n)$. Then, the decoder takes as input z and generates an output sequence $(y_1, \ldots, y_m)$ autoregressively, i.e. additionally taking as input all its previously generated tokens to predict the next one.

Encoder: The encoder block consists of N identical layers each containing two sublayers – a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.

Decoder: The decoder is composed of a stack of N identical layers as well. Each layer includes three sublayers, two of them being the same as those in the encoder. The additional third sublayer performs the multi-head attention over the output of the encoder stack. The self-attention sublayer in the decoder stack is modified so that no subsequent positions are attended, ensuring that the prediction at position i depends only on the known outputs at positions less than i.

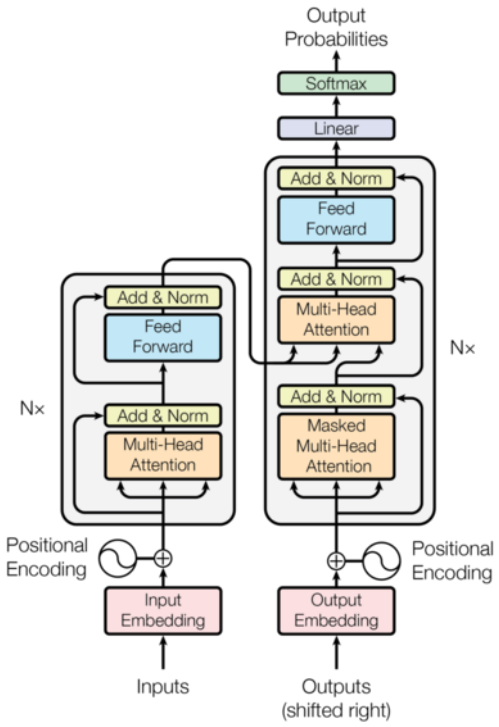

Figure 2.1: The encoder–decoder structure of the Transformer architecture.

Each sub-layer in the encoder and decoder has a residual connection around it followed by layer normalization. This is expressed as:

$$\text{LayerNorm}\big(x + \text{Sublayer}(x)\big), \tag{2.1}$$

In order to provide information about the position of each token in the sequence, the concept of Positional Encoding is used in the Transformer's architecture, which is defined for position pos and dimension i as:

$$PE_{(pos,\,2i)} = \sin\Big(\frac{pos}{10000^{2i/d_{\text{model}}}}\Big), \tag{2.2}$$

$$PE_{(pos,\,2i+1)} = \cos\Big(\frac{pos}{10000^{2i/d_{\text{model}}}}\Big). \tag{2.3}$$

2.1.2 Self-Attention Mechanism

The Attention mechanism is the principal component of the Transformer architecture which helps models to focus on relevant parts of the input sequence when predicting new tokens in the output. It can be defined as a function that maps queries $\mathbf{Q}$ and keys $\mathbf{K}$ of

dimension d_k, and values $\mathbf{V}$ of dimension d_v to an output. Those matrices are computed in the following way:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V, \tag{2.4}$$

where X is the input embedding matrix, and W^Q, W^K, and W^V are the learned weight matrices for the query, key, and value projections, respectively.

In the Transformer architecture, the measurement of the importance of a given token in the input relative to other tokens (including itself) is implemented as *scaled dot-product attention*. It is calculated using the dot product of the queries and keys, scaled by the square root of the key dimension to stabilize gradients. A softmax function is then applied to obtain the weights on the values:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \tag{2.5}$$

2.1.3 Multi-head Attention

Multi-head attention is an attention mechanism that projects the queries, keys, and values into h distinct subspaces using different learned linear projections, mapping them to dimensions d_k, d_k, and d_v, respectively [4].

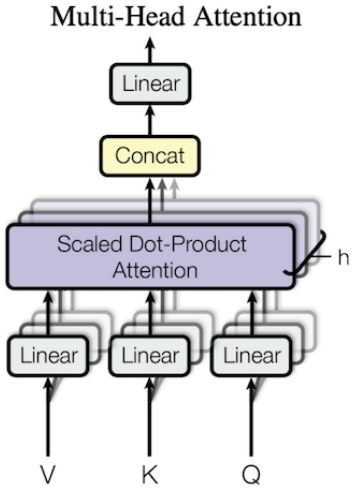

Figure 2.2: Multi-Head Attention consists of several attention layers running in parallel.

Each attention head computes a *Scaled Dot-Product Attention* independently. These

are then concatenated and projected back to the original model dimension through a final learned linear transformation, as illustrated in Figure 2.2. Formally, the mechanism is defined as:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O, \tag{2.6}$$

where each attention head head_i is computed using the scaled dot-product attention:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \tag{2.7}$$

In these equations:

- $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ are the learned projection matrices for the i-th head.
- $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ is the learned weight matrix used to project the concatenated outputs of all heads back into the original input.

2.2 Large Language Models (LLMs)

Large language models largely represent a class of Transformer networks scaled up to billions of parameters and trained on massive datasets. These models have achieved state-of-the-art results in the field of natural language processing thanks to the attention mechanism, which captures complex patterns and contextual information [4]. The architectures of such models vary.

There exist encoder-only models such as *Bidirectional Encoder Representations from Transformers* (BERT), which learn language representations in a bidirectional manner. Its main idea is to pre-train the encoder-only Transformer model on large text corpora using two tasks. The first is Masked Language Modeling (MLM), which involves predicting randomly masked words in a sentence to help BERT learn bidirectional context. The second task is Next Sentence Prediction (NSP), where the model is given pairs of sentences and must predict whether the second sentence follows the first one in the original text [5]. BERT is primarily used for tasks which require strong text understanding, such as classification and question answering.

On the contrary, there are LLMs based on a decoder-only Transformer architecture, the most prominent being *Generative Pre-trained Transformer* (GPT) models introduced by

Radford et al. [6] and later extended by Brown et al. [7] in GPT-3. Decoder-only models are optimized for autoregressive generation: they predict the next token in a sequence given all previous tokens. GPT and its successors are more suited for generative tasks, for example creative writing.

2.2.1 Selected Language Models

Gemma-2-2b-it

Gemma is an open-weight family of language models introduced by Google DeepMind in 2024. The Gemma-2-2b-it is an instruction-tuned model, which was trained using supervised examples to enhance its performance on instruction-following tasks. It contains 2.6 billion parameters and follows a decoder-only Transformer architecture [8].

Phi-4-mini-instruct

Phi-4-mini-instruct is a compact, instruction-tuned, decoder-only Transformer-based language model developed by Microsoft Research. With 3.8 billion parameters, it is optimized for strong performance. Despite its small size, the model has shown competitive results on various benchmarks, particularly in arithmetic reasoning and common-sense tasks, due to training on high-quality synthetic datasets [9].

2.3 Prompting and Prompt Engineering

A prompt is the raw input received by the LLM, and prompt engineering is the process of crafting high-quality prompts to guide the model toward accurate and relevant outputs. Additionally, certain hyperparameters influence the model's generation behavior during inference.

- **Output length** specifies the maximum number of tokens the model produces as output. By changing this hyperparameter one may control both the completeness of the response and the computational cost.
- **Temperature (T)** controls randomness in token selection: higher values (e.g., 0.8) yield more diverse outputs, while lower values (e.g., 0.1) produce more deterministic results.

There are different prompting strategies which enhance the performance of the LLM with no additional data and no training.

A key strategy is the few-shot prompting popularized by Brown et al. [7]. The key idea is to provide as input not only the question but also several examples of how the task should be completed. In that way, the LLM learns to complete tasks which it never saw before during training.

Another effective prompting technique is *Chain-of-Thought* (CoT) prompting , which encourages the model to reason step-by-step before producing the final output. CoT is most commonly used in combination with few-shot prompting. The LLM is provided with demonstrations of chain-of-thought reasoning in the input to learn this pattern for further question answering. Figure 2.3 shows an example of a model producing correct answers after applying CoT prompting to solve an arithmetic problem which was initially answered incorrectly [10].

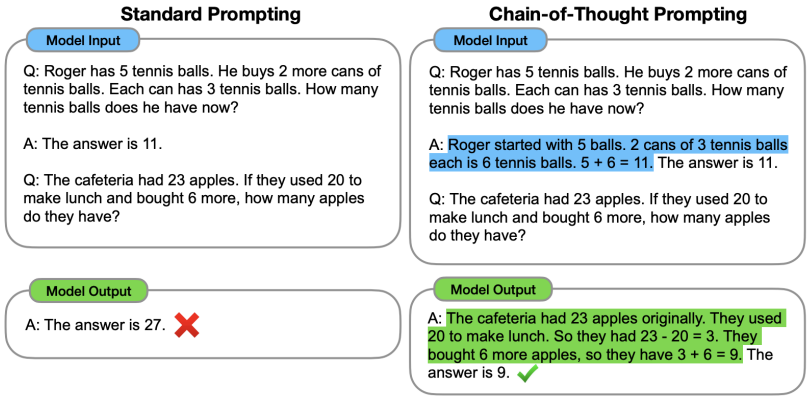

Figure 2.3: Example of Chain-of-Thought prompting: the model produces a correct answer after being guided to reason step-by-step.

This approach has been shown to significantly improve performance on tasks that require multi-step reasoning, including arithmetic word problems.

2.4 Self-Revision Paradigms

The term *self-revision* can be understood in multiple ways. In its strictest sense, it refers to LLMs independently refining their own responses [1, 11]. A broader interpretation also includes feedback from external tools or sources of knowledge [2, 12]. We define the

main types of *self-revision* and then introduce the hybrid approach proposed in our work, which combines elements of both.

2.4.1 Intrinsic Self-Revision

Intrinsic self-revision is a type of LLMs answer revision that does not include any external information or feedback and relies solely on the model's current knowledge. The process typically follows three steps [1, 2, 11, 13] and is illustrated in Figure 2.4:

1. The model is prompted to answer a question.
2. The output of step 1 is given back to the same model, and the LLM is asked to provide feedback on its own answer.
3. The model receives both the original question and the feedback as input and attempts to answer the question again, taking into account its feedback.

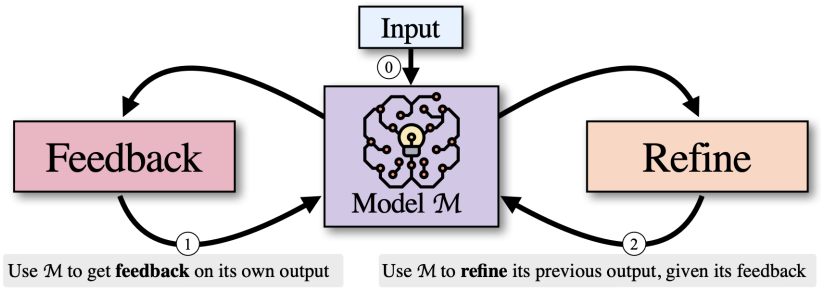

Figure 2.4: Self-Refine Framework: generate, self-feedback, and refine.

As noted in Chapter 1, this methodology has been questioned by several researchers and is not yet widely accepted as a standard technique for improving LLM performance.

2.4.2 Extrinsic Self-Revision

Extrinsic self-revision is a type of LLM refinement that involves the use of external information to guide the model's revision process. The workflow of the self-revision is the same as in the intrinsic approach, with the only difference being that at step 2, the model is fed not only with the initial answer but also with additional information, making the approach supervised. The main limitation of this method is that high-quality data is not

always available for most tasks. There are several types of information used for feedback generation, and some examples include:

- Oracle Feedback / Ground-Truth Answers [2, 13]
- Fine-Tuning with Human-Labeled Feedback [14, 15]
- Retrieved External Knowledge (e.g., Web Search, Wikipedia) [16]

2.5 Arithmetic Reasoning Benchmarks

Arithmetic word problem benchmarks serve as a tool for evaluating the reasoning and problem-solving capabilities of LLMs. In our study, to evaluate the self-refinement abilities of LLMs, we consider the following two math word problem benchmarks: the ASDiv dataset [17] and the GSM8K [18].

ASDiv is a dataset of 2.3K arithmetic word problems which provides problem diversity including sources like SATs, textbooks, and standardized tests. This corpora covers a wide range of math operations and question types and is widely used for benchmarking symbolic math solvers [17].

GSM8K is a dataset of 8.5K elementary math word problems, which require 2–8 steps to be solved. They are written using diverse linguistic formulations and are generally more complex than those in ASDiv. Those problems challenge LLMs with multi-step reasoning, making them ideal for testing CoT and refinement techniques. Despite the simplicity of the underlying math, many LLMs show poor performance on GSM8K, making this dataset a good choice for evaluating LLM reasoning abilities [18].

Chapter 3

Our Methodology

In this section, we present our approach to solving the problem formulated in Chapter 1.

3.1 Project Pipeline

Our goal is to evaluate LLMs' self-refinement abilities in the context of arithmetic reasoning. Our framework consists of (1) initial inference, where models solve mathematical problems using few-shot chain-of-thought prompting; (2) hint generation, in which incorrect responses are identified and used to create guiding hint; and (3) post-hint inference, in which the model reattempts the same problems using its own feedback as additional context. A diagram illustrating this process is presented in Figure 3.1. At the end, we use the obtained results to evaluate each model's improvement after injecting hint sentences for making conclusions about LLMs' self-refinement limitations.

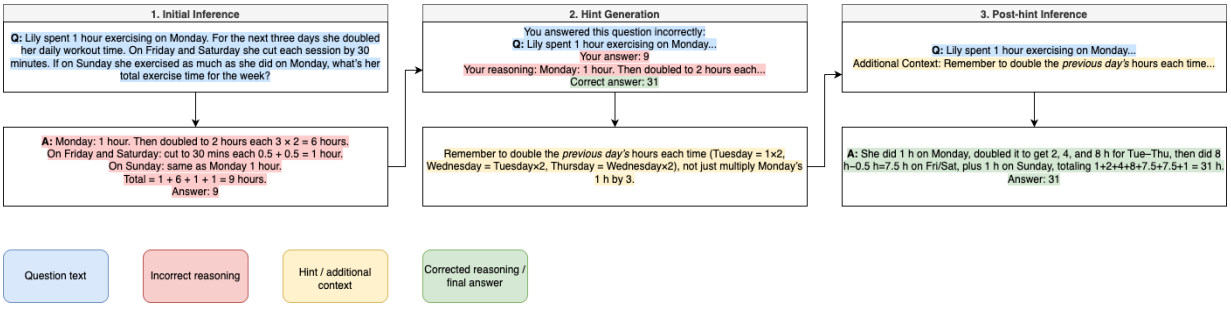

Figure 3.1: LLM Self-Refinement Workflow: (1) Initial Inference, (2) Hint Generation for incorrect responses, and (3) Post-Hint Inference using the generated hints.

3.1.1 Initial Inference

The first step of the pipeline is to prompt the LLMs to solve mathematical questions. To solve the problems, few-shot prompting and CoT prompting techniques are applied. The input to the model includes the target question and four examples demonstrating the desired reasoning format. Each response must include the model's step-by-step reasoning inside special markers "<cot_start>" and "<cot_end>" and conclude with a clearly stated final answer.

3.1.2 Hint Generation

From the Initial Inference step (3.1.1), all incorrectly answered questions (cases where the model's predicted numerical answer does not exactly match the ground truth) are extracted for targeted refinement. For each incorrect example, the model was prompted with the original question, its prediction (the chain-of-thought and incorrect final answer), and the correct answer. The LLM is asked to generate the information and reasonings that would've helped it to answer the original question correctly without writing the correct answer explicitly. The output of this step is referred to as the hint sentences intended to improve model performance after their injection. In case of answer leakage, the hint generation step was repeated up to three times using temperature sampling (T=0.8) to encourage variation.

3.1.3 Post-Hint Inference

The problems, which were originally incorrectly solved, are revised given the hint sentences generated by the same LLM. Now the model is prompted to answer the question, given the same four shots in the prompt, but additional hint sentences are inserted as additional context. This setup tests whether the model can effectively use its own externally guided feedback to revise its reasoning and produce a correct final answer.

3.2 Evaluation Metrics

Self-refinement capabilities of LLMs are assessed based on three core metrics, each formally defined in Eqs. (3.1)–(3.4) below:

- **Baseline Accuracy** (A_{base}, Eq. (3.1)): The percentage of mathematical problems which were correctly solved during initial inference.
- **Δ Accuracy (Overall Accuracy Gain)** (ΔA, Eq. (3.4)): The net increase in overall accuracy across the full dataset after hint-based self-refinement. Since hint sentences are only generated for initially incorrect examples, the overall accuracy gain cannot decrease and depends solely on the number of problems corrected after hint injection.
- **Post-Hint Accuracy** (A_{post}, Eq. (3.2)): The percentage of questions correctly answered after hint injection, computed only for the subset of examples that were initially answered incorrectly.

Let N be the total number of problems, C_0 the number correctly solved in initial inference, $I_0 = N - C_0$ the number initially answered incorrectly, and C_1 the number of those I_0 problems solved correctly after hint injection.

Then:

$$A_{\text{base}} = \frac{C_0}{N}, \tag{3.1}$$

$$A_{\text{post}} = \frac{C_1}{I_0} = \frac{C_1}{N - C_0}, \tag{3.2}$$

$$A_{\text{final}} = \frac{C_0 + C_1}{N}, \tag{3.3}$$

$$\Delta A = A_{\text{final}} - A_{\text{base}} = \frac{C_1}{N}. \tag{3.4}$$

Chapter 4

Experiments and Results

In this chapter we summarize the experiments we conducted and discuss the results.

4.1 Experimental Setup

We evaluated two instruction-tuned LLMs—Gemma-2-2b-it and Phi-4-mini-instruct—on two standard arithmetic reasoning benchmark subsets: ASDiv (2,023 problems) and GSM8K (1,305 problems). For each model–dataset pair, we ran the three-step self-refinement pipeline described in Section 3.1 and computed the metrics from Section 3.2.

4.2 Results and Discussion

Table 4.1 reports the baseline accuracy (A_{base}), post-hint accuracy (A_{post}), final accuracy (A_{final}), and accuracy gain (ΔA) for each model–dataset pair.

Model	Dataset	A_{base} (%)	A_{final} (%)	ΔA (%)	A_{post} (%)
gemma-2-2b-it	ASDiv	75.53	80.38	4.84	19.80
gemma-2-2b-it	GSM8K	39.92	51.65	11.72	19.52
phi-4-mini-instruct	ASDiv	95.65	96.44	0.79	18.18
phi-4-mini-instruct	GSM8K	86.82	89.12	2.30	17.44

Table 4.1: Self-refinement results for each model–dataset pair

Baseline Accuracy: GSM8K features more linguistically complex mathematical prob-

lems, which is why both models perform poorly on that dataset compared to the results obtained on ASDiv. Additionally, Phi-4-mini-instruct outperforms Gemma-2-2b-it on both benchmarks. These baseline differences must be considered when drawing conclusions from the overall accuracy gain metric.

Overall Accuracy Gain: The largest difference between initial and post-hint inference occurs for Gemma-2-2b-it on GSM8K, where accuracy increases from 39.92% to 51.65%. Conversely, Phi-4-mini-instruct on ASDiv shows the smallest gain (0.79%), reflecting its high initial accuracy (95.65%). Note that, because hints were generated only for questions answered incorrectly, the delta metric depends solely on how many of those questions the model corrected after adding context.

Post-Hint Accuracy: All model–dataset combinations converge to a post-hint accuracy of roughly 20%, i.e., only about one-fifth of the initially incorrect predictions were corrected after hint injection. For the remaining items, self-generated hints failed to guide the model to the correct answer.

4.3 Qualitative Case Studies

To understand why self-generated hints often fail, we examine three representative examples from our post-hint inference logs. In each case, although the model is given a correct hint, it fails to understand how to arrive at the correct answer and does not recognize what it did incorrectly. These examples are taken from the post-hint inference logs of the Phi-4-mini-instruct model on ASDiv (ID29, ID1954) and Gemma-2-2b-it on ASDiv (ID1982).

Case 1 (ID 29): Coin-difference

Problem: "Mrs. Hilt has two pennies, two dimes, and two nickels. Jacob has four pennies, one nickel, and one dime. How much more does Mrs. Hilt have than Jacob?"

Injected Hint: Mrs. Hilt has two pennies, two dimes, and two nickels. Jacob has four pennies, one nickel, and one dime. To determine how much more Mrs. Hilt has compared to Jacob, we need to calculate the total value of each person's coins and then find the difference.

Post-hint Inference: The model recomputed $\$0.32 - \$0.19 = \$0.13$, exactly as before.

Analysis: The hint simply repeats the information already provided in the problem and restates the solution approach from the original chain of thought. It fails to understand that the model's issue is outputting the answer in dollars rather than computing the difference in item counts. As a result, the model's reasoning chain and final answer remain unchanged.

Case 2 (ID 1954): Baseball Team Members

Problem: "Faye's team won their baseball game and scored sixty-eight points total. If Faye scored twenty-eight of those points and every other player scored eight points each, how many players were on her team?"

Injected Hint: Faye's team won their baseball game and scored sixty-eight points total. Faye scored twenty-eight of the points. Everyone else scored eight points each. To find the number of players on Faye's team, we first need to determine how many points were scored by the other players. We do this by subtracting the points Faye scored from the total points: $68 - 28 = 40$. Next, we divide the points scored by the other players by the number of points each of them scored.

Outcome: The model computes the number of team members in Faye's team correctly (5) and then adds one to it, since it calculates Faye included too. This is the same answer as before hint injection.

Analysis: The hint simply repeated the original chain of thought, omitting only the final answer. The model failed to recognize that the question asks for the number of team members excluding Faye. Although the correct answer (5) was provided during hint generation, the model could not infer the proper interpretation of the problem.

Case 3 (ID 1982): Packaging Boxes

Problem: "Nancy sold 45 boxes. How many full cases of 12 boxes can she fill, and how many extra boxes will remain?"

Injected Hint: Nancy sold 45 boxes of Tagalongs. Each case contains 12 boxes. Number of full cases $= 45 \div 12 = 3$ (since $3 \times 12 = 36$). Extra boxes $= 45 - 36 = 9$.

Outcome: The model performs the same steps as originally and outputs the same incorrect answer—3.

Analysis: The model does not understand that taking only three cases of twelve boxes yields 36, which is less than the required 45 boxes. It treats the extra boxes as a supplement to reach 45, when in fact they represent the count beyond 45. Although the model was

given the correct answer (4), it failed to recognize its mistake and simply repeated its original chain of thought as the hint sentence.

As a result, these examples clearly demonstrate situations in which LLMs do not understand why their initial answers were incorrect, even when the correct answers are provided. Their ability to identify their own mistakes is limited, and they often repeat the same errors during hint generation.

Chapter 5

Summary and Conclusion

In this study we evaluated self-refinement capabilities of LLMs in arithmetic context. A hybrid approach of feedback generation was employed combining extrinsic approach at the stage of hint generation and intrinsic approach at the stage of post-hint inference.

Our methodology consisted of several key stages: initial inference, hint generation, post-hint inference and evaluation of the results. We employed advanced techniques such as a Few-shot CoT prompting and Temperature-controlled sampling. This setup allowed us to make a fair assessment of LLMs self-reflection abilities.

The experiments conducted demonstrate that the feedback sentences provided by the LLM are useful for itself in the least of the cases. Most of the time, the LLM fails to understand where its reasoning failed, despite the correct answer being provided. Our experimentation provides valuable insights into the limitations of LLMs self-reflection abilities. The LLMs may still need external verification mechanisms which will guide them toward correct answers.

In conclusion, our study results are in line with the conclusion that LLMs do not yet know what/where and why they lack. However, there is a need for further research to widen the range of considerations. Our research was centered on arithmetic reasoning, but diverse tasks and domains may be considered for the self-refinement capabilities of LLMs. Additionally, larger models should be tested to determine whether increased scale improves their self-refinement capabilities.

Bibliography

[1] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, "Self-refine: Iterative refinement with self-feedback," arXiv preprint arXiv:2303.17651, 2023.

[2] N. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems*, 2023.

[3] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou, "Large language models cannot self-correct reasoning yet," arXiv preprint arXiv:2310.01798, 2023.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, 2017, pp. 5998–6008.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*. Association for Computational Linguistics, 2019, pp. 4171–4186.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," OpenAI Blog, 2018.

[7] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse,

M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–1901.

[8] Google DeepMind, "Gemma-2-2b-it: Instruction-tuned decoder-only transformer with 2.6b parameters," Online, 2024, https://www.deepmind.com/publications/gemma-2-2b-it.

[9] Microsoft Research, "Phi-4-mini-instruct: Compact decoder-only transformer with 3.8b parameters," Online, 2024, https://www.microsoft.com/research/project/phi-4-mini-instruct.

[10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, vol. 35. Curran Associates, Inc., 2022, pp. 24 824–24 837.

[11] J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han, "Large language models can self-improve," arXiv preprint arXiv:2210.11610, 2022.

[12] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen, "CRITIC: Large language models can self-correct with tool-interactive critiquing," in *Proceedings of the 12th International Conference on Learning Representations*, 2024.

[13] G. Kim, P. Baldi, and S. McAleer, "Language models can solve computer tasks," in *Advances in Neural Information Processing Systems*, vol. 36. Curran Associates, Inc., 2023, pp. 39 648–39 677.

[14] S. Welleck, X. Lu, P. West, F. Brahman, T. Shen, D. Khashabi, and Y. Choi, "Generating sequences by learning to self-correct," in *Proceedings of the Eleventh International Conference on Learning Representations*, 2023.

[15] S. Ye, Y. Jo, D. Kim, S. Kim, H. Hwang, and M. Seo, "Selfee: Iterative self-revising llm empowered by self-feedback generation," Blog post, 2023.

[16] L. Gao, Z. Dai, P. Pasupat, A. Chen, A. T. Chaganty, Y. Fan, V. Zhao, N. Lao, H. Lee, D.-C. Juan, and K. Guu, "Rarr: Researching and revising what language models say, using language models," in *Proceedings of the 61st Annual Meeting of the*

Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, 2023, pp. 16 477–16 508.

[17] S.-Y. Miao, C.-C. Liang, and K.-Y. Su, "A diverse corpus for evaluating and developing english math word problem solvers," arXiv preprint arXiv:2106.15772, 2021.

[18] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, Łukasz Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," arXiv preprint arXiv:2110.14168, 2021.