

---

# Mid-Term Project Report of LLM

## Leveraging Small Language Models (SLMs) to Mitigate Hallucinations in Large Language Models (LLMs)

Ihab Tabbara<sup>a</sup>, Johnny Huang<sup>b</sup>, Yiwen Lu<sup>c</sup>

<sup>a</sup> i.k.tabbara@wustl.edu

<sup>b</sup> h.johnny@wustl.edu <sup>c</sup> lyiwen@wustl.edu

## 1 Introduction

### 1.1 Why Hallucinations is Important

Language generation, reasoning, and dialogue systems are just a few natural language processing (NLP) applications in which large language models (LLMs) have shown impressive performance. However, LLMs often suffer from a critical issue known as hallucination, where the model generates information that is plausible-sounding but factually incorrect. According to a survey by [3], there are several underlying reasons why NLP patients have hallucinations. Heuristic data collection is one important element. In this approach, models are trained on sizable but incomplete datasets that contain biased or noisy data, which results in incorrect generalizations. The limitation of representation learning is another issue that may contribute, since LLMs may find it difficult to adequately capture the true underlying relationships within the data. Mistakes in decoding techniques, such as beam search, might worsen the issue by prioritizing outputs that are syntactically valid but factually inaccurate. Nonetheless, exposure bias, which arises from discrepancies between training and inference phases, can cause the model to deviate from the truth during generation. Lastly, the model's strong reliance on internalized knowledge, or parametric knowledge bias, may result in the generation of erroneous or obsolete data. Every one of these elements adds to the difficulty of treating hallucinations in LLMs.

### 1.2 Why Use SLMs to Train LLMs

Recent studies highlight the limitations of larger language models, which tend to produce less truthful responses as their size increases. The "TruthfulQA" [5] study, for instance, shows that scaling up models can reduce output reliability, indicating that larger models and more data do not necessarily improve truthfulness. Because of this restriction, questions have been raised concerning the validity of LLMs in delicate fields like legal advice, healthcare, and education. In this approach, it is feasible to train LLMs utilizing Small Language Models (SLMs). Hallucinations can be lessened by using SLMs to assist in the evaluation and guidance of larger language models to solve more complex problems in a more optimized and coherent manner. For instance, recent attempts have included techniques such as Retrieval-Augmented Generation (RAG) and prompt tuning [7]. Additionally, using SLMs to train LLMs has shown promise in similar works. One promising approach is to segment the reasoning process of LLMs into a structured sequence of intermediate steps, often referred to as a "Chain of Thought" (CoT) [10], which allows for better monitoring and correction of errors. Inspired by the work in the paper "Small Language Models Fine-tuned to Coordinate Larger Language Models Improve Complex Reasoning" [4], this proposal seeks to investigate how SLMs can be used to detect and correct hallucinations by segmenting the LLM's output into discrete "thoughts" and identifying where hallucination

---

occurs. Moreover, researchers [8] suggest that in certain cases, LLMs cannot find reasoning errors, but can correct them given the error location. With the utilization of SLMs, we not only seek to improve performance but also provide a more effective way to enhance LLMs outputs without relying on massive computational resources.

## 2 Research Objectives

This work aims to investigate how, by segmenting the outputs of large language models into different thoughts, akin to the Chain of Thought paradigm, customized Small Language Models can be used to detect and reduce hallucinations in large language models. By segmenting the data, we will be able to identify the precise points in the LLM’s thinking process where they occur and offer remedial input.

Specific Objectives:

1. Develop a method to segment the LLM’s output into distinct, interpretable thoughts or sub-claims.
2. Fine-tune Small Language Models to analyze segmented outputs, detecting inconsistencies between adjacent thoughts or parts that are factually incorrect.
3. Implement a feedback mechanism where the SLMs flag potential sub-claims as hallucinations and suggest corrections to the LLM.
4. Apply the method to reasoning tasks such as word sorting, tracking shuffled objects, logical deduction, and multistep arithmetic.
5. Evaluate the performance of this method in terms of reduction in hallucination and improvement in reasoning quality.
6. Accumulate a dataset of hallucinated thoughts and their corresponding correct labels and test whether fine tuning the LLM on this dataset will improve its ability to reason and solve math problems.

## 3 Proposed Methodology

The following steps outline the methodology to be employed in this research:

### 3.1 Data Collection and Task Setup

The following datasets will be gathered especially for reasoning tasks that are prone to hallucination errors:

- Word sorting assignments that call for arranging things in a specific order.
- Tracking shuffled things requires the model to stay aware of the positions of the objects as they move.
- Tasks using logic that tests the model’s capacity to conclude from premises.
- Multi-step math problems where a hallucination may cause inaccurate intermediate steps and final answers.

- 
- Various datasets will also pertain to multi-step problems, which will be used to probe hallucination inconsistencies and aim to resolve them.

### 3.2 Segmenting the Chain of Thought

For these particular tasks, we will create a technique to automatically partition LLM-generated outputs into discrete ideas or steps in the reasoning process. This organized segmentation will allow us to monitor the points in the reasoning chain where hallucinations occur.

### 3.3 Fine-Tuning Small Language Models

The SLM will be fine-tuned to analyze pairs of consecutive thoughts (e.g., Thought 1 and Thought 2) and identify potential inconsistencies or logical breakdowns specific to reasoning tasks like multistep arithmetic or logical deduction. We may also incorporate tuning to the SLM to detect non-factual and incorrect statements to be flagged. The sequence will not come from a specific dataset but instead, be generated by another more powerful model in the form of a detection trajectory[1].

### 3.4 Detecting and Flagging Hallucinations

After completing their training, the SLM will be expected to assess every pair of ideas in the LLM’s logic chain. Additionally, we can employ the SLM as agents to use external tools for hallucination detection [1]. The SLM will monitor whether object placements stay the same during the reasoning phases for tasks like tracking shuffled objects.

### 3.5 Iterative Feedback Mechanism

After a hallucination is detected, the SLM will provide feedback to the LLM, suggesting corrections to the faulty reasoning step. This process will iterate until the LLM produces a consistent, hallucination-free reasoning chain.

## 4 Hallucination Detection Methodology

In this section, we describe the current methodologies being explored to detect hallucinations in large language models (LLMs) using the GSM and AlphaMath datasets from hugging face. The following approaches are being utilized to improve the hallucination detection process:

### 4.1 Traditional Methods for Hallucination Detection

Before using language models for hallucination detection, we explored traditional methods to use as a benchmark. These tools include:

- **Calculator Tools and APIs:** Using external tools to verify mathematical operations performed by the LLM by checking the correctness of numerical results generated in its reasoning.
- **Regular Expressions:** Applying regular expressions to extract and validate specific numbers or symbols from the LLM’s output, ensuring that they match reasonable expectations or ground truth.

- 
- **Web Scraping:** For certain facts or claims generated by the LLM, we scrape data from external sources or databases to cross-check the validity of the LLM's thoughts.

```
Question: How many apples does Ramy have after buying 12 more apples if he initially had 60?
Correct Answer: 70 apples

Thought 1: Ramy starts with 60 apples.
No hallucination detected.

Thought 2: He buys 12 more apples.
No hallucination detected.

Thought 3: Now Ramy has  $60 + 12 = 70$  apples.
Potential hallucination detected!
- Invalid arithmetic operation

Thought 4: Suddenly, Ramy has 1000 apples.
Potential hallucination detected!
- Unusual quantity change detected
```

Figure 1: Example with Regex matching and calculator tool. Thought 4 also demonstrates common sense check by the SLM.

## 4.2 LLM and SLM Techniques for Hallucination Detection

In addition to traditional methods, we are utilizing Small Language Models (SLMs) alongside LLMs to help detect hallucinations. In this approach:

- We input the thought generated by the LLM, the original question or task prompt, and the correct or true label (when available) into an SLM or another LLM.
- The model then assigns a confidence score to indicate the likelihood of the thought being a hallucination. This approach is inspired by prior research, which utilized direct prompting of language models to produce confidence scores reflecting the accuracy of statements [2].
- If the thought is flagged as a hallucination, the model outputs this assessment, along with a confidence score, which can be used to adjust or correct the thought.

We applied the `detect_hallucination_with_score` function to detect hallucinations in each step. The small model compares the generated step with the correct answer and assigns a confidence score between 0 and 10 to evaluate its accuracy.

### 4.2.1 Problems found:

#### 1) Prompt Problem:

In the step 2, we use the prompt:

```
"Evaluate the correctness of this thought: 'thought' compared to the correct
answer: 'correct_answer'. Provide a confidence score between 0 and 10, with
clear reasoning."
```

In this step, we observed that the small model was unable to comprehend such a long prompt, resulting in output that consisted of various nonsensical, repeated statements, as shown in the Figure 1.

## 2) Small-Scale LLMs Problem:

"Is the step 'thought' correct in solving the question 'question' based on the correct answer 'correct\_answer'? Please respond with 'Yes' or 'No'."

From the results showed in Figure 2, the SLMs clearly generate hallucination in step 1, which indicated that this approach failed to detect hallucinations effectively.

This suggests that the issue may come from the limitations of the small-scale language models used. Consequently, we decided to employ a language model better suited for mathematical reasoning.

However, it is evident that providing the correct answer directly to the small model is not an effective approach, as it does not enable the model to adapt to new questions. Therefore, we will modify it by training the SLM to use simple methods to identify which steps contain hallucinations.

1. **No Direct Comparison with Correct Answer:** Instead of explicitly providing the correct answer to the SLM, we train the SLM to focus on internal consistency, coherence,

---

and logical flow within the generated thought. This allows the SLM to evaluate whether the reasoning step is hallucinated based on patterns rather than a direct answer comparison.

”Evaluate if the thought ‘thought’ follows logically from the previous steps in the reasoning. Respond with ‘Yes’ if it makes sense or ‘No’ if there is a logical gap.”

2. **Training on Labeled Hallucinations:** We create a dataset containing reasoning steps with labeled hallucinations and non-hallucinations. The SLM is trained to detect patterns of common hallucinations (e.g., irrelevant information, inconsistencies in reasoning) instead of relying on a comparison with the correct answer. The training should focus on:
  - Identifying deviations in logic.
  - Detecting unnecessary elaboration or irrelevant information.
  - Recognizing contradictions with earlier reasoning steps.
3. **Confidence-Based Evaluation:** After detecting potential hallucinations in a step, the SLM should output a confidence score indicating how likely it is that the step contains a hallucination. This allows for a more flexible detection process that doesn’t depend on having a predefined correct answer.
4. **Iterative Feedback and Refinement:** Use the confidence score generated by the SLM to refine the thought process. If the confidence score is below a certain threshold (e.g., 5/10), the LLM re-generates that specific reasoning step, incorporating feedback based on what caused the SLM to flag it.
5. **Gradual Learning Approach:** Over time, as the SLM processes more steps, it will become better at identifying common hallucination patterns, enabling it to generalize better to new, unseen tasks or questions.

### 4.3 Tuning SLMs for Function calling

One promising technique we are exploring further is the fine-tuning of Small language Models (SLMs). Specifically, we aim to fine tune the SLMs so that they can make decisions when trying to determine whether or not a QA pair is a hallucination including:

- Segmenting initial question into distinct semantic sentences.
- **Utilize different tools to check for validity of each sentence.**
- Utilize matching tool to check between facts from sentences with the final answer.
- Give final assessment of whether or not the QA pair is a hallucination.

#### 4.3.1 Tasks and Problems found

Primarily, we developed some trajectory dataset for the SLM to be tuned on to automatically call on the appropriate tools/functions to check for sentence validity. These functions are also implemented as part of our experiment which we will go more into depth with

---

later. The LLM we are attempting to detect anomalies of that we are also using the generate trajectory data for finetuning our SLM is GPT-3.5. From our first experimentation we attempted to utilize the “Baichuan-7B” model for tool calling, however, being constrained by a limited GPU RAM we were unable to load the model entirely for fine tuning. Instead, we found fine tuning the smaller “Qwen2-1.5B-Instruct” to be more effective and manageable. We fine tuned the 1.5B model with qLoRA on the query and value matrices. We tuned the model for 3 epochs on 2000 trajectory data and the process took a total of 30 minutes.

### 4.3.2 Function calling Dataset

We generated a total of 2000 trajectories utilizing GPT-3.5 to guide on the SLM on being fine-tuned for function calling different tools available. The trajectories first include instructions for the SLM to follow and tells it how to act when given a user query question and LLM response. It then guides the SLM on how it’s expected to act on calling functions for each sentence, what those functions return, and asks the SLM to reason based on the returns of the functions. Finally, the SLM is asked to produce a final conclusion about whether or not the QA pair is a hallucination.

#### JSON training trajectory:

- **System:** You are an agent tasked with detecting hallucinations in reply texts using a specific framework. Below is a detailed explanation of the detection framework: Firstly, you need to determine whether to split the input reply text into a list of sentences using a sentence segmentation tool. If required, you should check each sentence individually; otherwise, the entire text should be checked as a whole. You can choose an appropriate fact-checking tool to obtain relevant information and knowledge for verification and then use the matching tool to output the judgment results or directly output the judgment results. If you do not use the match tool and directly output the judgment results, you need to output the label in your thought. There is an error output “label = 1”; there is no error output “label = 0”. After the verification is completed, you need to reflect on all detection results and output the label in your thought, then call `get.answer()` to produce the final detection result.
- **User:** QUERY: Is 7 a number that is prime? RESPONSE: 7 is a prime number because it is divisible by the number 7.  $7/7 = 1$ .
- **Assistant:** THOUGHT: need to split the provided response into individual sentences to begin the hallucination detection process. For this, I will use the sentence segmentation tool.  
ACTION: `split_text(text="7 is a prime number because it is divisible by the number 7.  $7/7 = 1$  therefore 7 is not a prime number.")`
- **User:** OBSERVATION: ["7 is a prime number because it is divisible by the number 7"," $7/7$  is 1."]
- **Assistant:** THOUGHT: Fact-check the first sentence using the search tool to find relevant factual information.  
ACTION: `web_search(sentence="7 is a prime number because it is divisible by the number 7."` THOUGHT: Check the 2nd sentence with a calculator `calculator("7/7=1")`.)
- **User:** OBSERVATION: "web\_search(sentence="7 is a prime number because it is divisible by the number 7." label = 1. Fact: "7 is a prime number as it is only divisible by itself and the number 1." OBSERVATION: "calculator("7/7=1"), label = 0"
- **User:** OBSERVATION: Yes. { "sentence": "7 is a prime number because it is divisible by the number 7.", "fact": "7 is a prime number as it is only divisible by itself and the number 1." }

\*An example trajectory used to fine-tune the SLM. We have the Question and a hallucinated answer, which is then passed to the SLM that tests the QA pair for hallucinations. The final judgement is “Yes,” indicating a hallucination; “No” would mean no hallucination is detected.

### 4.3.3 Function Calling

We implemented a variety of tools our fine-tuned agent to be able to call in order to check for the accuracy of our QA claim. In the following section we will go over all of the tools that’s implemented for the agent to be able to use.

- **Document Splitting:** The document split function takes in a structured response from the LLM in the form of a string and split it into distinct sentences for further processing. The resulting sentences is then stored in a list and returned by the function. The document split function utilizes the spaCy api to do this simply and effectively.

- **Calculator:** The calculator function simply takes in a proposed equation and calculates it. For the purpose of simple equation checking, we only coded the calculator to be able to recognize the signs of +, -, \*, /. We primarily used REGEX to carry out this logic. The function first checks if the equation is valid or solvable, and if so it calculates the equation and returns a floating point number. If not, it raises an exception for invalid calculation.
- **Web Search:** The web search function takes in a sentence and searches the web for relevant information. For every piece of relevant information found, we append it to a fact string and return the single string at the end. If no information is found on the subject, then the function returns “No Results Found.” and there is also built in error handling. The amount of facts retrieved for our input is capped at 5. We utilize the google API to do the searches easily.
- **Matching tool:** The matching tool is a critical function that takes in two sentences and matches whether or not they support each other. The first sentence represents the LLM generated sentence in the original QA pair where as the second sentence represents the results from the web search tool, calculator tool, etc. The return of this function is a single label of values 1 or 0, with 1 representing there being a mismatch being found and 0 being concurrency. To determine whether or not the two sentences match each other, we utilize GPT-3.5 to check between the sentences and provide us with a label in its response.

```

[{'role': 'User',
  'content': 'You are an agent tasked with detecting hallucinations in reply texts using a specific framework. Below is a detailed explanation of the detection framework:\nFirstly, you need to determine whether to split the input reply text into a list of sentences using a sentence segmentation tool. If required, you should check each sentence individually; otherwise, the entire text should be checked as a whole. You can choose an appropriate fact-checking tool to obtain relevant information and knowledge for verification and then use the matching tool to output the judgment results or directly output the judgment results. If you do not use the match tool and directly output the judgment results, you need to output the label in your thought. There is an error output "label = 1"; there is no error output "label = 0". After the verification is completed, you need to reflect on all detection results and output the label in your thought, then call get\\_answer() to produce the final detection result. \n\nSentence Splitting Tool: \\nsplit_text(text: str) + sentence_list\nThis function splits the text into a list of sentences.\n\nFact-Checking Tools: \\nweb_search(sentence: str) + fact\nThis function uses a search engine to find information related to the sentence. After using web_search, you must use the match tool to determine if the reply matches the retrieved information.\n\ncalculator(sentence: str, formula: str) + result, label\nThis function uses a calculator to obtain the result of a formula and checks if the result matches the sentence. If they match, the label is 0; otherwise, it is 1. Valid operators include +, -, *, /, and parentheses. For instance, a valid input could be "(1 + 2) * 3". If the input is an equation, it needs to be converted to a formula without unknowns.\n\nword_count(length: int, text: str) + count, label\nThis function calculates the word count of a text and outputs the count. If the word count does not meet the specified length, the label is 1; otherwise, it is 0.\n\ncode_interpreter() + label\nThis function checks whether the code can execute correctly. If it executes correctly, the output label is 0; otherwise, it is 1.\n\nMatching Tool: \\nmatch(sentence: str, context: str) + label\nThis function checks a sentence against its context, which might include content from questions and replies around the detected sentence. It looks for irrelevant or contradictory answers. If any are found, the label is 1; otherwise, it is 0. If you think the output of match is wrong, you can correct the label in thought. For example, if you think the "label = 0" output by match is wrong, you can correct the answer and output "label = 1" in thought.\n\nEach time it's your turn to respond, you must strictly follow this format to present your thoughts and actions: "Thought: Your thought process.\n\\nACTION: Tool call, e.g., match(sentence=..., context=...)". After each tool use, I will provide the output as follows: "Observation: Tool's output result". QUERY: The heights of the five basketball players in Class 1 of Grade 3 are respectively 149 cm, 151 cm, 147 cm, 150 cm, and 148 cm. What is their average height? RESPONSE: (149+151+147+150+148)/5 = 147.8\n\nSo, their average height is 147.8 cm.'},
 {'role': 'Assistant',
  'content': 'Thought: To find the average height of the basketball players, I will use the calculator tool to perform the calculation.\n\\nACTION: calculator(sentence="(149+151+147+150+148)/5", formula="(149+151+147+150+148)"),
 {'role': 'User',
  'content': 'OBSERVATION: (149+151+147+150+148)/5 = 149, label = 1'},
 {'role': 'Assistant',
  'content': 'Thought: The calculator tool returned a result of 149, which does not match the correct average height of 147.8 cm. The label indicates an error. I will correct this and output the correct average height.\n\\nACTION: get_answer()'}]

```

Figure 4: Example generation with final return label of 1 (hallucination detected)

- **Get Answer:** This is a niche function implemented for the SLM to call at the end of the hallucination detection process that ties everything together. The sizes of the “sentences, labels, facts” lists are all equal to each other. The function takes in all of the sentences from the original segmented LLM response, the information gathered about each of the sentences utilize different tools, as well as the label for whether or not each of the sentences match with each of the pieces of information gathered. The label will be 1 for pairs that do not match, and 0 for pairs that do. If all of the labels are 0, then we return “No” as our final hallucination detection verdict. Otherwise, we pick out all the pairs of the three lists that has a label of 1, append each of them to a final json object with the “claim” from sentences and “correction” from facts and return all instances of violations along with the final verdict “Yes,” there indeed has been a hallucination.

We currently have all of the functions needed for function calling implemented and have already generated baseline trajectories and trained the SLM with the data. The next steps would be to carry out inference and collect accuracy scores from our standard testing datasets to see how well the SLM performs on detecting hallucinations using available tools.



---

## 5 Methodology for Hallucination Correction

The next phase of our work we will be focusing on in the future involves designing controlled experiments where we create a feedback loop between the LLM and the SLM to address hallucinations. We propose the following experiment configurations:

### 5.1 Feedback Loop to Correct Hallucinations

We will create a feedback loop mechanism where, upon detecting a hallucination, we provide feedback to the LLM in one of the following ways:

1. **Method 1: Feedback with Correct Label.** When a hallucination is detected, we give the LLM the wrong thought alongside the correct label or output. This way, the LLM can learn the correct reasoning and update its next generated thought accordingly.
2. **Method 2: Feedback with Confidence Score.** When a hallucination is detected, we provide the LLM with the thoughts and their associated confidence scores. By doing so, the LLM can recognize the error, adjust its reasoning process, and refine its next generated response to align more closely with correct reasoning.
3. **Method 3: Feedback Without Label.** Instead of providing the correct label, we simply tell the LLM that its thought is incorrect and ask it to retry the reasoning process.
4. **Method 4: Feedback with Examples of Previous Hallucinations** A potential method could involve providing the LLM with examples of previous hallucinations and corrections made, allowing it to self-correct based on prior mistakes. This would introduce a form of meta-cognitive learning, where the LLM learns from past errors in a more generalized way, without specific label correction.

We have already implemented method2, but the results are not ideal and may require further adjustments. Other methods will be tested in the future.

### 5.2 Problems found in Method 2:

To complete the entire process and evaluate the code, we will temporarily set aside the issue encountered in hallucination detection. Instead, we will utilize the following sources as feedback for testing the validity of the code in hallucination correction: 1) Section 4.2, 2) ChatGPT 4.0 Mini. Here is the example used:

- **Question:** Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May?
- **Correct Answer:** Natalia sold  $48/2 = 24$  clips in May. Natalia sold  $48+24 = 72$  clips altogether in April and May.
- **Thoughts:** ['Natalia sold  $48 / 2 = 24$  clips in May.', 'In April, Natalia sold  $48 + 24 = 72$  clips.', 'In May, Natalia sold  $48 / 2 = 24$  clips.', 'In April and May, Natalia sold  $72 + 24 = 108$  clips.', 'The answer: 108.']

#### 5.2.1 Feedback from Section 4.2:

**Feedback:** ['Step 1: Score = 5', 'Step 2: Score = 5', 'Step 3: Score = 5', 'Step 4: Score = 3', 'Step 5: Score = 1']

---

Updated Thoughts Based on Feedback:

Updated Step 1: In April, Natalia sold  $48 / 2 = 24$  clips.

Updated Step 2: In May, Natalia sold  $24 / 2 = 12$  clips.

Updated Step 3: Altogether, Natalia sold  $48 + 12 = 64$  clips in April and May.

Updated Step 4: The final answer: 64.

Figure 5: Updated Thoughts Based on Feedback from Section 4.2

From figure 5, we can see the LLM introduces new hallucinations, and the original hallucinations is not corrected. This may be due to the feedback from SLM is not reasonable and do not include the previous thoughts.

### 5.2.2 Feedback from ChatGPT 4.0 Mini:


- 
1. **'Natalia sold  $48 / 2 = 24$  clips in May.'**  
Correct.  
**Score:** 10/10.
  2. **'In April, Natalia sold  $48 + 24 = 72$  clips.'**  
Incorrect, this sum is for April and May, not just April.  
**Score:** 0/10.
  3. **'In May, Natalia sold  $48 / 2 = 24$  clips.'**  
Correct but redundant.  
**Score:** 10/10.
  4. **'In April and May, Natalia sold  $72 + 24 = 108$  clips.'**  
Incorrect, it should be  $48 + 24 = 72$ .  
**Score:** 0/10.
  5. **'The answer: 108.'**  
Incorrect.  
**Score:** 0/10.

Figure 6: The prompt generated from ChatGPT 4.0 mini

```

feedback = [
    'Natalia sold 48 / 2 = 24 clips in May. Correct. Score: 10/10.',
    'In April, Natalia sold 48 + 24 = 72 clips. Incorrect, this sum is for April and May, not just April. Score: 0/10.',
    'In May, Natalia sold 48 / 2 = 24 clips. Correct but redundant. Score: 10/10.',
    'In April and May, Natalia sold 72 + 24 = 108 clips. Incorrect, it should be 48 + 24 = 72. Score: 0/10.',
    'The answer: 108. Incorrect. Score: 0/10.'
]
updated_thoughts = generate_thoughts(question, feedback)

print("Updated Thoughts Based on Feedback:\n")
for i, updated_thought in enumerate(updated_thoughts):
    print(f"Updated Step {i+1}: {updated_thought}\n")

```

Updated Thoughts Based on Feedback:

Updated Step 1: In April, Natalia sold 48 + 24 = 72 clips.

Updated Step 2: In May, Natalia sold 48 / 2 = 24 clips.

Updated Step 3: In April and May, Natalia sold 72 + 24 = 108 clips.

Updated Step 4: The answer: 108.

Figure 7: Updated Thoughts Based on Feedback from ChatGPT 4.0 Mini

From figure 7, the `generate_thoughts` function may not correctly utilizing the feedback to modify the original thoughts. More adjustment is needed in this function.

### 5.2.3 Summary of Issues in Method 2:

In Method 2, although we have tested the hallucination correction process using feedback from Section 4.2 and ChatGPT 4.0 Mini, the results are still not satisfactory. Both feedback mechanisms fail to effectively address the hallucinations introduced in the thought generation process. Specifically, Section 4.2’s feedback did not provide sufficient context to rectify previous incorrect thoughts, and ChatGPT 4.0 Mini did not fully utilize the feedback for modification. Further adjustments are required in both the feedback mechanism and the thought generation function to improve the overall performance of hallucination correction in LLMs.

## 5.3 Dataset Generation for Fine-Tuning

As part of these experiments, we will accumulate a dataset of hallucinated thoughts and their corresponding correct labels. This dataset will serve a dual purpose:

- It will also enable us to fine-tune the LLM on hallucination correction, improving its reasoning and reducing the occurrence of hallucinations in future outputs.
- Refining the feedback loop techniques discussed above to reduce the likelihood of hallucination in future thoughts generated by the LLM.

## References

- [1] X. Cheng, J. Li, W. X. Zhao, H. Zhang, F. Zhang, D. Zhang, K. Gai, and J.-R. Wen. Small agent can also rock! empowering small language models as hallucination detector. *arXiv preprint arXiv:2406.11277*, 2024.
- [2] B. Hou, Y. Zhang, J. Andreas, and S. Chang. A probabilistic framework for llm hallucination detection via belief tree propagation. *arXiv preprint arXiv:2406.06950*, 2024.
- [3] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38, Mar. 2023.
- [4] G. Juneja, S. Dutta, S. Chakrabarti, S. Manchanda, and T. Chakraborty. Small language models fine-tuned to coordinate larger language models improve complex reasoning. *arXiv preprint arXiv:2310.18338*, 2023.

- 
- [5] S. Lin, J. Hilton, and O. Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.
  - [6] OpenAI. GSM8K: A dataset of grade school math word problems. <https://github.com/openai/gsm8k>, 2021. MIT License.
  - [7] S. M. T. I. Tonmoy, S. M. M. Zaman, V. Jain, A. Rani, V. Rawte, A. Chadha, and A. Das. A comprehensive survey of hallucination mitigation techniques in large language models, 2024.
  - [8] G. Tyen, H. Mansoor, P. Chen, T. Mak, and V. Cărbune. Llms cannot find reasoning errors, but can correct them! *arXiv preprint arXiv:2311.08516*, 2023.
  - [9] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.
  - [10] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.