# AdaSwitch: Adaptive Switching between Small and Large Agents for Effective Cloud-Local Collaborative Learning

**Hao Sun[1,2], Jiayi Wu[3], Hengyi Cai[4], Xiaochi Wei[5]**
**Yue Feng[7], Bo Wang[6], Shuaiqiang Wang[5], Yan Zhang[1,2], Dawei Yin[5]**
[1]State Key Laboratory of General Artificial Intelligence, Peking University, Beijing, China
[2]School of Intelligence Science and Technology, Peking University
[3]East China Normal University, [4]Chinese Academy of Sciences
[5]Baidu Inc, [6]Beijing Institute of Technology, [7]University of Birmingham
sunhao@stu.pku.edu.cn

## Abstract

Recent advancements in large language models (LLMs) have been remarkable. Users face a choice between using cloud-based LLMs for generation quality and deploying local-based LLMs for lower computational cost. The former option is typically costly and inefficient, while the latter usually fails to deliver satisfactory performance for reasoning steps requiring deliberate thought processes. In this work, we propose a novel LLM utilization paradigm that facilitates the collaborative operation of large cloud-based LLMs and smaller local-deployed LLMs. Our framework comprises two primary modules: the local agent instantiated with a relatively smaller LLM, handling less complex reasoning steps, and the cloud agent equipped with a larger LLM, managing more intricate reasoning steps. This collaborative processing is enabled through an adaptive mechanism where the local agent introspectively identifies errors and proactively seeks assistance from the cloud agent, thereby effectively integrating the strengths of both locally-deployed and cloud-based LLMs, resulting in significant enhancements in task completion performance and efficiency. We evaluate AdaSwitch across 7 benchmarks, ranging from mathematical reasoning and complex question answering, using various types of LLMs to instantiate the local and cloud agents. The empirical results show that AdaSwitch effectively improves the performance of the local agent, and sometimes achieves competitive results compared to the cloud agent while utilizing much less computational overhead.

## 1 Introduction

Recently, the advent of large language models (LLMs) has garnered substantial attention from the public, industry, and academia, attributed to their advanced language comprehension and generation capabilities. These LLMs, such as OpenAI's GPT-4 (Achiam et al., 2023) and Google's PALM (Anil et al., 2023), are characterized by their massive scale, both in terms of the colossal number of parameters and the substantial volume of data utilized during their training process. Due to their large number of parameters, LLMs are typically deployed on cloud servers. However, the reliance on cloud computing makes LLM utilization considerably costly and inefficient, owing to the substantial bandwidth consumption, considerable strain on the network architecture and the need for extensive computational resources.

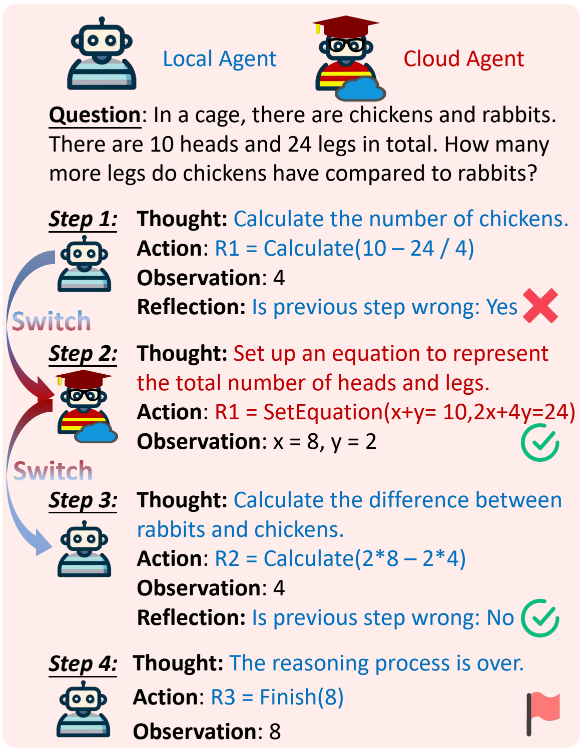

Local Agent
Cloud Agent
Question: In a cage, there are chickens and rabbits. There are 10 heads and 24 legs in total. How many more legs do chickens have compared to rabbits?
Step 1: Thought: Calculate the number of chickens.
Action: R1 = Calculate(10 – 24 / 4)
Observation: 4
Reflection: Is previous step wrong: Yes
Switch
Step 2: Thought: Set up an equation to represent the total number of heads and legs.
Action: R1 = SetEquation(x+y= 10,2x+4y=24)
Observation: x = 8, y = 2
Switch
Step 3: Thought: Calculate the difference between rabbits and chickens.
Action: R2 = Calculate(2*8 – 2*4)
Observation: 4
Reflection: Is previous step wrong: No
Step 4: Thought: The reasoning process is over.
Action: R3 = Finish(8)
Observation: 8

Figure 1: A brief illustration of AdaSwitch framework, in which local agent and cloud agent alternate to collaboratively fulfill the given question.

A promising solution is to reduce the high computational demand of LLMs through techniques such as knowledge distillation (Liang et al., 2020; Gu et al., 2023) or model quantization (Frantar et al., 2022; Lin et al., 2023; Xiao et al., 2023), and to deploy LLMs directly on local devices. Though effective, small-sized LLMs are prone to severe performance degradation when confronted with complex and demanding situations, and usually fail to deliver satisfactory performance for reasoning steps requiring deliberate thought processes.

To harness the strong capabilities of large-sized LLMs with the convenience of small-sized LLMs, we propose ADASWITCH, a novel framework enabling these two types of LLMs to collaboratively solve complex open-world tasks. This framework is inspired by human behavior in similar scenarios: when faced with complex tasks, people often seek assistance from more knowledgeable individuals for challenging components and learn from their guidance to complete the tasks. This ability to proactively seek assistance and apply acquired knowledge is a critical aspect of human intelligence. Similarly, ADASWITCH comprises two primary modules: the local agent and the cloud agent. The local agent, instantiated with a relatively smaller LLM, is capable of handling less complex reasoning steps. In contrast, the cloud agent, responsible for more deliberate reasoning, utilizes larger LLMs, such as `Llama-30B`. Our proposed approach is designed to enable both efficient inference with local smaller LLMs and resource-intensive cloud LLM executions for task steps requiring higher cognitive capabilities. It effectively integrates the strengths of both locally-deployed and cloud-based LLMs, resulting in significant enhancements in task completion performance and efficiency.

As depicted in Figure 1, given the question "how many more legs do chickens have compared to rabbits?", ADASWITCH interleaves its generation by first composing the sub-solution "calculate the number of chickens", then reflecting on the prior failing step, and offloading this challenging step to the cloud agent. The local agent thus is able to form an improved action to finally accomplish the task. Using the cloud agent as an assistant allows the local agent to make effective use of a larger knowledge base and focus its efforts on learning the task steps appropriate for the model's current competence.

The main idea behind ADASWITCH is to allow the local agent to adaptively activate the cloud agent when it introspectively judges the current step as incorrect. To this end, we enhance the local agent's self-checking capabilities by meticulously collecting inaccurate reasoning paths to construct the mistake-checking dataset. Specifically, we ask the local agent to undertake an exam during which the cloud agent dynamically corrects the local agent's mistakes, incentivizing the local agent to learn from mistakes, determine when to ask for assistance, and how to utilize feedback to correct the mistake. Finally, the resultant local agent can introspectively judge the running steps, proactively seek assistance, and apply acquired feedback to improve its subsequent actions.

We conduct experiments on mathematical reasoning and complex reasoning benchmarks. ADASWITCH consistently improves the performance across various LLMs and tasks. For instance, the performance of the local model instantiated with the `DeepSeek-Coder-1.3B` can be improved from 29.3% to 53.9%, requiring 3x fewer computational costs for LLM inference than competitor system while achieving similar results. Notably, our proposed framework even enables `StarCoder2-3B` to achieve comparable performance against `Llama-30B`, with 5x less computational overhead for LLM inference. The effectiveness of the proposed method is also verified by ablation experiments and analytical experiments.

## 2 Methodology

### 2.1 Preliminary

In the agent framework, the agents usually follow the interaction paradigm, where the agent predicts a thought and an action, and the environment gives feedback. Specifically, the backbone of the agent is an LLM denoted as $\mathcal{M}$. In the $t$-th step, the LLM $\mathcal{M}$ generates a thought $s_t$ and an action $a_t$ based on the instruction and the current state of the system:

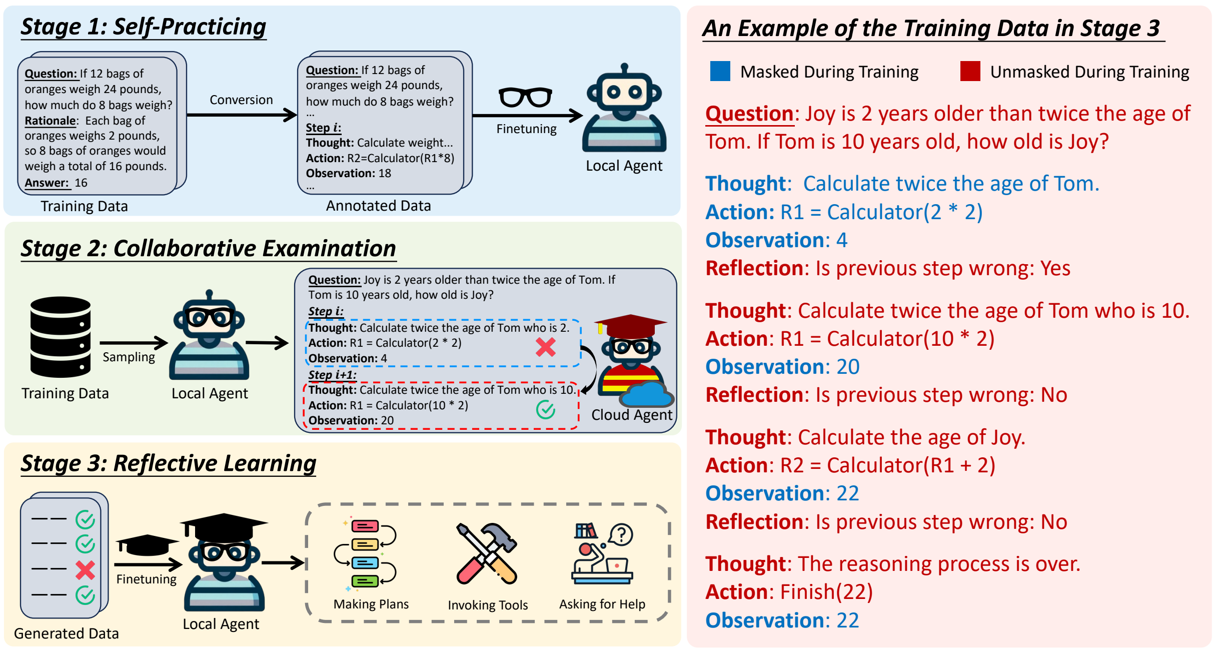
$$s_t, a_t = \mathcal{M}(\tau_{t-1}), \tag{1}$$

$$o_t = \text{Execution}(a_t) \tag{2}$$

where $\tau_{t-1} = \{s_1, a_1, o_1, ..., s_{t-1}, a_{t-1}, o_{t-1}\}$ denotes the previous interaction trajectory. Here, $o_t$ denotes the observation returned by tools when the action $a_t$ is executed. The tool list used in this paper is shown in Table 6.

### 2.2 ADASWITCH Framework

We propose a multi-stage learning paradigm that enables the local agent to introspectively judge the running steps, proactively seek assistance, and apply acquired feedback to improve its subsequent actions. Specifically, as shown in Figure 2, the learning of our framework can be divided into three


Stage 1: Self-Practicing
Question: If 12 bags of oranges weigh 24 pounds, how much do 8 bags weigh?
Rationale: Each bag of oranges weighs 2 pounds, so 8 bags of oranges would weigh a total of 16 pounds.
Answer: 16
Training Data
Conversion
Question: If 12 bags of oranges weigh 24 pounds, how much do 8 bags weigh?
Step i:
Thought: Calculate weight...
Action: R2=Calculator(R1*8)
Observation: 18
Annotated Data
Finetuning
Local Agent
Stage 2: Collaborative Examination
Training Data
Sampling
Local Agent
Question: Joy is 2 years older than twice the age of Tom. If Tom is 10 years old, how old is Joy?
Step i:
Thought: Calculate twice the age of Tom who is 2.
Action: R1 = Calculator(2 * 2)
Observation: 4
Step i+1:
Thought: Calculate twice the age of Tom who is 10.
Action: R1 = Calculator(10 * 2)
Observation: 20
Cloud Agent
Stage 3: Reflective Learning
Generated Data
Finetuning
Local Agent
Making Plans
Invoking Tools
Asking for Help
An Example of the Training Data in Stage 3
Masked During Training
Unmasked During Training

🟦 Masked During Training 🟥 Unmasked During Training

Question: Joy is 2 years older than twice the age of Tom. If Tom is 10 years old, how old is Joy?

Thought: Calculate twice the age of Tom.
Action: R1 = Calculator(2 * 2)
Observation: 4
Reflection: Is previous step wrong: Yes

Thought: Calculate twice the age of Tom who is 10.
Action: R1 = Calculator(10 * 2)
Observation: 20
Reflection: Is previous step wrong: No

Thought: Calculate the age of Joy.
Action: R2 = Calculator(R1 + 2)
Observation: 22
Reflection: Is previous step wrong: No

Thought: The reasoning process is over.
Action: Finish(22)
Observation: 22

Figure 2: The illustration of ADASWITCH. 1) Self-Practicing: The local agent practices on the training dataset to brain the basic reasoning ability. 2) Collaborative Examination: The local agent undertakes an exam to expose its weakness, during which the cloud agent will be utilized to correct the mistakes. 3) Reflective Learning: The local agent is trained on the mistake-correction trajectories generated in the second stage.

stages: (1) firstly, the local agent is trained on the training set to build a basic reasoning ability; (2) then, the local agent undertakes an exam to expose its weakness in accomplishing the challenging steps, during which the cloud agents are utilized to correct the local agents' mistakes; (3) finally, the local agents are trained on the mistake-checking and mistake-correction trajectories generated in the second stage.

**Self-Practicing** Given a training dataset $D_{train} = \{\langle x_i, r_i, y_i \rangle\}_{i=1}^{k}$, where $x_i$ denotes the question, $r_i$ represents the ground-truth intermediate reasoning steps and $y_i$ is the answer, we follow Yin et al. (2023) and employ LLM to transform the reasoning steps $r$ into interaction trajectories. Specifically, we provide the LLM with the question, ground-truth intermediate reasoning steps, and defined action space, then the LLM is able to generate high-level thoughts and corresponding actions accordingly. Subsequently, we can obtain the interaction trajectories in the format of $\tau = \{s_1, a_1, o_1, ..., s_n, a_n, o_n\}$. To enable the local agent to focus on the reasoning part, we feed the entire interaction trajectories to the LLMs while merely calculating the decoding loss on the tokens of the subgoals and action by applying binary masking on observation tokens.

**Collaborative Examination** After training on the annotated dataset, the local agent can solve the question interactively. To detect the weakness of the local agent, we ask the local agent to undertake an exam on the training set $D_{train}$. For each question, we collect up to four interaction trajectories through decoding with the Top-K sampling strategy. During the interaction, we follow Li et al. (2022) and adopt a rule-based method to dynamically examine whether each reasoning step is wrong. For mathematical reasoning tasks, we gather intermediate results from the correct trajectories and verify whether each step's execution result matches any of the intermediate results in the correct trajectories. If there is a match, the step is deemed correct; otherwise, it is considered incorrect. In the case of textual reasoning tasks, we can similarly verify the correctness of each step by utilizing `roberta-large-mnli` (Liu et al., 2019) to check whether the thought and action of each step are semantically equivalent to any of the reasoning steps in the correct trajectories.

When a mistake is detected, the cloud agent will be activated to correct the mistake by erasing the wrong step in the prompt and regenerating the step. Then the local agent will continue based on the cloud agent's corrected step. This process will continue until the local agent finally reaches the answer. If the answer is true, we incorporate the interaction trajectory into the training set.

**Reflective Learning** After obtaining the mistake-checking and mistake-correction data, we ask the local agent to train on the revised trajectories. It is worth mentioning that the revised trajectories contain wrong steps that have wrong thoughts and actions. Therefore, we mask these parts during loss calculations so that the local agent will not be confused by these steps.

**Collaborative Inference** After training on the mistake-checking and mistake-correction dataset, the local agent can detect and correct errors. We introduce two modes for task inference. The first mode is SELF-REFLECTION, where the local agent will rely on itself to correct the mistake when it finds an error. However, due to the limited ability of the local agent, it might not be able to correct the mistake accurately. Therefore, we further introduce the second mode where the cloud agent will be activated to correct the mistake, which we call ADASWITCH. Specifically, the local agent predicts a probability that the previous step is wrong, and the cloud agent will be activated when the probability is larger than an activation threshold $p$.

## 3 Experiment

### 3.1 Tasks & Datasets

**Mathematical Task** We adopt five math word problem datasets to evaluate the mathematical reasoning ability. GSM8K is a primary school-level mathematical dataset (Cobbe et al., 2021). G_Hard is a harder version of GSM8K (Gao et al., 2022). MultiArith is a multi-step arithmetic reasoning dataset (Roy and Roth, 2016). SVAMP is created by applying chosen variations over examples sampled from existing datasets (Patel et al., 2021). ASDIV is a math word problem dataset that contains examples with diverse language patterns and problem types (Miao et al., 2021).

**Complex QA Task** We use two open-domain question-answering datasets to evaluate the complex reasoning ability. HotpotQA dataset (Yang et al., 2018) is a multi-hop question-answering dataset. MuSiQue dataset (Trivedi et al., 2022) is a multi-hop reasoning dataset.

### 3.2 Experimental Setup

For mathematical tasks, we use GSM8K as the training dataset. For the complex QA task, we use MuSiQue as the training dataset. The detailed statistics of these datasets are shown in Table 5. Moreover, the cloud agent undergoes only the self-practicing stage to build basic reasoning ability before deployment, while the local agent undergoes the full learning stages.

### 3.3 Models

The candidate local agents include `DeepSeek-Coder-1.3B`[1] and `StarCoder2-3B`[2]. And the candidate cloud agents include `CodeLlama-13B` [3], `Llama-30B` [4], `Qwen1.5-32B` [5], and `Llama-2-70B` [6].

### 3.4 Main Results

In this section, we conduct experiments on seven challenging reasoning tasks utilizing the 1.3B and 3B local agents and the 30B cloud agent. The result is shown in Table 1. Based on the result, several observations can be made:

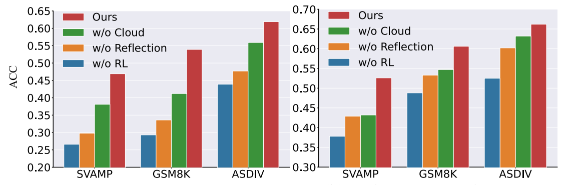
First, **our method, ADASWITCH, greatly improves the performance of the local agent**, achieving up to 86.9% relative improvement using the 1.3B model as the local agent and up to 39.1% relative improvement using the 3B model as the local agent. This is primarily because, after collaborative learning, ADASWITCH enables the local agent to seek help from the cloud agent when it detects potential mistakes. By doing so, the local agent can handle the easier steps independently while leveraging the cloud agent for more difficult steps, thereby enhancing its overall performance.

Second, **the improvement is more pronounced on difficult datasets**, as evidenced by an 86.9% relative improvement on G_Hard dataset and a 22.3% relative improvement on MultiArith dataset. This is mainly because the local agent learns to request assistance when necessary, and when faced with more challenging datasets, the local agent will call for help more frequently, leading to significant performance gains. However, the cost associated with these difficult datasets will also increase. To balance between cost and effectiveness, we can adjust the predefined activation threshold $p$, which is discussed in Table 2.

[1] https://huggingface.co/deepseek-ai/deepseek-coder-1.3b-instruct
[2] https://huggingface.co/bigcode/starcoder2-3b
[3] https://huggingface.co/codellama/CodeLlama-13b-hf
[4] https://huggingface.co/huggyllama/llama-30b
[5] https://huggingface.co/Qwen/Qwen1.5-32B-Chat
[6] https://huggingface.co/meta-llama/Llama-2-70b-hf

| Method | # Para | Mathematical Reasoning |  |  |  |  | Complex QA Reasoning |  |
|---|---|---|---|---|---|---|---|---|
|  |  | GSM8K | G_Hard | SVAMP | ASDIV | MultiArith | MuSiQue | HotpotQA |
| **Using 1.3B Local Agent** |  |  |  |  |  |  |  |  |
| **Local Agent** | 1.3B | 29.30 | 25.20 | 26.60 | 43.90 | 77.22 | 29.80 | 25.80 |
| +ADASWITCH | 1.3B | 53.90 (+24.6) | 47.10 (+21.9) | 46.90 (+20.3) | 61.90 (+18.0) | 94.44 (+17.2) | 36.80 (+7.0) | 32.50 (+6.7) |
| **Using 3B Local Agent** |  |  |  |  |  |  |  |  |
| **Local Agent** | 3B | 48.80 | 40.10 | 37.80 | 52.50 | 87.22 | 31.50 | 29.50 |
| +ADASWITCH | 3B | 60.60 (+11.8) | 50.60 (+10.5) | 52.60 (+14.8) | 66.20 (+13.7) | 96.11 (+8.9) | 37.80 (+6.3) | 31.80 (+2.3) |
| **Cloud Agent** | 30B | 63.20 | 55.00 | 52.10 | 63.80 | 98.89 | 41.80 | 35.50 |

Table 1: Our main experimental results (%) on five mathematical reasoning tasks and two complex question-answering tasks. Local agent refers to the agent after the self-practicing while ADASWITCH refers to undergoing the full learning paradigm and then collaborating with the cloud agent during inference.



(a) Using 1.3B Local Agent (b) Using 3B Local Agent

Figure 3: We conduct an ablation study by removing the cloud agent, self-reflection, and reflective learning.

## 3.5 Ablation Study

In this section, we assess the performance of different inference modes using `DeepSeek-Coder-1.3B` and `StarCoder2-3B` as local agents, which is shown in Figure 3. Specifically, w/o cloud agent refers to utilizing the local agent to self-correct the mistakes without asking for help from the cloud agent. w/o reflection refers to continuing the inference processes without identifying or correcting errors in previous steps. w/o RL refers to removing the reflective learning.

As we can see, after removing the cloud agent. the performance degrades dramatically, which is reasonable because the local agent cannot solve the difficult steps due to its limited reasoning ability. However, by conducting self-reflection, the performance of w/o cloud is still higher than the performance of w/o reflection, which demonstrates that the local agent can correctly detect and correct mistakes on its own. Moreover, the performance of w/o reflection is higher than w/o RL, demonstrating that after reflective learning, the local agent's reasoning ability gets improved.

## 3.6 Hyper-parameter Analysis

In our method, ADASWITCH, the decision for the local agent to seek assistance from the cloud agent is controlled by an activation threshold, denoted as $p$. To evaluate the impact of different threshold values, we conducted experiments using $p$ values of $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ with a 1.3B local agent and a 70B cloud agent on the subset of three mathematical reasoning datasets.

| | GSM8K | | SVAMP | | ASDIV | |
|---|---|---|---|---|---|---|
| $p$ | Acc | Cost | Acc | Cost | Acc | Cost |
| **0.1** | 57.60 | 121.80 | 62.40 | 75.53 | 69.30 | 49.39 |
| **0.3** | 57.30 | 62.95 | 58.40 | 53.21 | 67.40 | 36.39 |
| **0.5** | 57.40 | 77.61 | 55.40 | 46.48 | 66.60 | 31.59 |
| **0.7** | 53.10 | 49.18 | 51.60 | 29.93 | 65.50 | 26.70 |
| **0.9** | 48.50 | 37.90 | 45.20 | 21.07 | 63.40 | 23.49 |

Table 2: Results (%) of ADASWITCH using different activation threshold. As the threshold increases, ADASWITCH activates the cloud agent more frequently, leading to improved performance but at a higher inference cost. The unit of the cost is FLOPs.

As illustrated in Table 2, lowering the threshold $p$ results in the local agent requesting help more frequently, which generally leads to enhanced model performance. However, this increased reliance on the cloud agent also incurs higher computational costs. For instance, on the GSM8K dataset, the cost escalates from 37.90 FLOPs to 121.80 FLOPs as the threshold decreases from 0.9 to 0.1. Moreover, we observe a saturation effect in performance improvement as the threshold value is further reduced. While a decrease in threshold from 0.9 to 0.7 results in a 9.4% increase in accuracy, a further reduction from 0.3 to 0.1 yields only a 0.5% improvement. This diminishing return is primarily because, after a certain point, the local agent's performance closely approximates that of the cloud agent. Consequently, further reducing the threshold may result in the local agent seeking assistance on steps where the cloud agent is also prone to errors, thereby minimally impacting overall accuracy.

## 3.7 Analysis

**Switching Analysis** To demonstrate the effectiveness of various switching strategies, we compare it with the following variants: Random Switch: The local agent switches to the cloud agent with a probability of $p$. Sequential Switch: The local agent switches to the cloud agent every consecutive $k$ steps. Confidence Switch: The local agent switches to the cloud agent when the probability of its generated tokens is lower than a predefined threshold $p$. To ensure a fair comparison, we tune these corresponding hyperparameters to keep all methods at a similar cost level.

As shown in Table 4, all methods improve the performance of the local agent after collaborating with the cloud agent. Among the variants, the Confidence Switch improves the performance higher than others. This is mainly because the token probability distribution can reflect the local agent's capability in solving the question to some extent, making the cost quota be distributed properly. However, the overconfidence phenomenon (Yang et al., 2024; Groot and Valdenegro-Toro, 2024; Xiong et al., 2023) makes the probability distribution an unstable metric, leading to inferior performance compared with our method.

**Capability of Self-checking** During the collaborative examination, we assess the correctness of each step using predefined rules. We first analyze the accuracy of the labeling process and then analyze if the local agent can correctly predict label accuracy. We randomly selected 100 questions and manually checked the rule-based labels. Each step's ground truth label is positive if correct and negative otherwise. The True Positive Rate (TPR) is the proportion of correct steps identified correctly, and the True Negative Rate (TNR) is the proportion of incorrect steps identified correctly. Our analysis showed that the rule-based method achieved a TPR of 92% and a TNR of 61%. The high TPR indicates the rule labeling process effectively identifies correct steps, while the low TNR suggests some steps are wrong though the result of the step has appeared in the correct trajectories. Such corner cases need human intervention to further verify the correctness of each step.

To verify if the local agent can learn to predict the correctness of each step, we randomly selected 100 inference trajectories of AdaSwitch and Confidence Switch. Our analysis showed that AdaSwitch achieved a TPR of 82% and a TNR of 52%, while Confidence Switch achieved a TPR of 73% and a TNR of 27%. These results confirm the effectiveness of our method. Notably, AdaSwitch's TPR and TNR closely match the label data, showing it effectively utilizes training data to improve its ability to detect mistakes. Higher performance could be achieved with better training data, such as using LLMs as data labelers, which is suggested for future work.

| Method | GSM8K | G_Hard | SVAMP | MultiArith |
|---|---|---|---|---|
| **13B** | 61.70 | 57.10 | 49.80 | 98.89 |
| **32B** | 70.40 | 59.80 | 67.90 | 100.00 |
| **70B** | 74.90 | 64.00 | 74.70 | 97.78 |
| **Using 1.3B Local Agent** | | | | |
| **Local** | 29.30 | 25.20 | 26.60 | 77.22 |
| **+ 13B** | 54.24 | 48.89 | 44.58 | 97.51 |
| **+ 32B** | 58.80 | 51.60 | 52.67 | 96.65 |
| **+ 70B** | 58.96 | 52.51 | 55.81 | 97.92 |
| **Using 3B Local Agent** | | | | |
| **Local** | 48.80 | 40.10 | 37.80 | 87.22 |
| **+ 13B** | 61.23 | 49.95 | 50.30 | 97.56 |
| **+ 32B** | 64.49 | 52.40 | 59.14 | 98.67 |
| **+ 70B** | 65.43 | 53.66 | 61.36 | 98.28 |

Table 3: Results (%) using different cloud agent models, where +13B refers to collaborating the local agent with the 13B cloud agent.

**Generalization Ability** In this section, we aim to analyze whether the local agent can collaborate with different cloud agents without further retraining. Specifically, we utilize both the 1.3B and 3B local agents trained under the supervision of the 30B cloud agent and ask the local agent to ask for help from other cloud agents when it thinks it has made a mistake. We choose cloud agents of different parameter sizes, which include 13B, 32B, and 70B. Based on the result shown in Table 3, we can have the following conclusions:

First, when switching to unknown cloud agents, the local agent's performance can still be improved significantly, which demonstrates the generalization ability of our method. Moreover, the improvement ratio is higher when switching to larger cloud agents with better model capability. For example, the relative improvement of AdaSwitch achieves surprising 101% and 109% on GSM8K and SVAMP when the 1.3B local agent collaborates with the 70B cloud agent. This is important because we can only deploy one local agent on the local device while deploying multiple cloud agents remotely. During inference, the users can dynam-

| Method | # Para | Mathematical Reasoning | | | | | Complex QA Reasoning | |
|---|---|---|---|---|---|---|---|---|
| | | GSM8K | G_Hard | SVAMP | ASDIV | MultiArith | MuSiQue | HotpotQA |
| **Using 1.3B Local Agent** | | | | | | | | |
| Random Switch | 1.3B | 36.50 | 29.90 | 35.60 | 50.60 | 85.56 | 31.00 | 24.80 |
| Sequential Switch | 1.3B | 36.00 | 29.90 | 31.80 | 47.90 | 82.78 | 26.80 | 28.80 |
| Confidence Switch | 1.3B | 38.70 | 33.40 | 35.60 | 53.40 | 89.44 | 29.80 | 29.50 |
| ADASWITCH | 1.3B | 53.90 | 47.10 | 46.90 | 61.90 | 94.44 | 36.80 | 32.50 |
| **Using 3B Local Agent** | | | | | | | | |
| Random Switch | 3B | 54.40 | 49.80 | 45.40 | 60.40 | 94.44 | 35.50 | 26.80 |
| Sequential Switch | 3B | 52.70 | 48.80 | 43.00 | 60.40 | 93.33 | 34.50 | 30.50 |
| Confidence Switch | 3B | 54.70 | 48.90 | 47.60 | 63.20 | 95.00 | 35.80 | 31.00 |
| ADASWITCH | 3B | 60.60 | 50.60 | 52.60 | 66.20 | 96.11 | 37.80 | 31.80 |

Table 4: Results (%) of different switching strategies. We conduct the experiment by using 1.3B and 3B as the local agents to dynamically activate the 30B cloud agent.

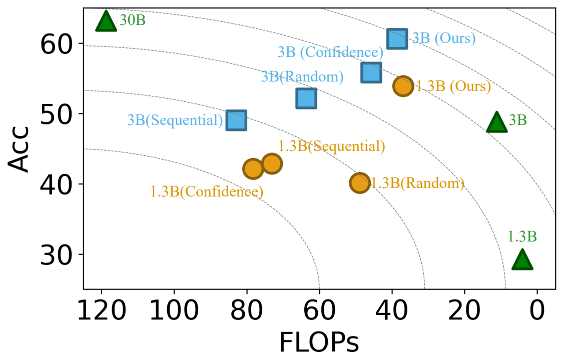


Figure 4: Cost-Effectiveness Analysis. We conduct experiments on the GSM8K dataset. From left to right, the cost of the methods gradually increases. From the bottom to the top, the accuracy of the method increases.

ically decide which cloud agents to use based on the quota of computational resources.

**Cost-Effectiveness Analysis** In this section, we analyze the cost and effectiveness of existing methods. Specifically, we take GSM8K as our test bed and calculate the average FLOPs cost per query utilizing different inference modes, such as Random Switch, Sequential Switch, and Confidence Switch.

As shown in Figure 4, the 30B local agent achieves the highest performance with the largest computational cost, and the 1.3B local agent achieves the lowest performance with the lowest cost. As a comparison, ADASWITCH balances cost and effectiveness and achieves a similar cost with 30B agent with 3x fewer computational costs.

### 3.8 Case Study

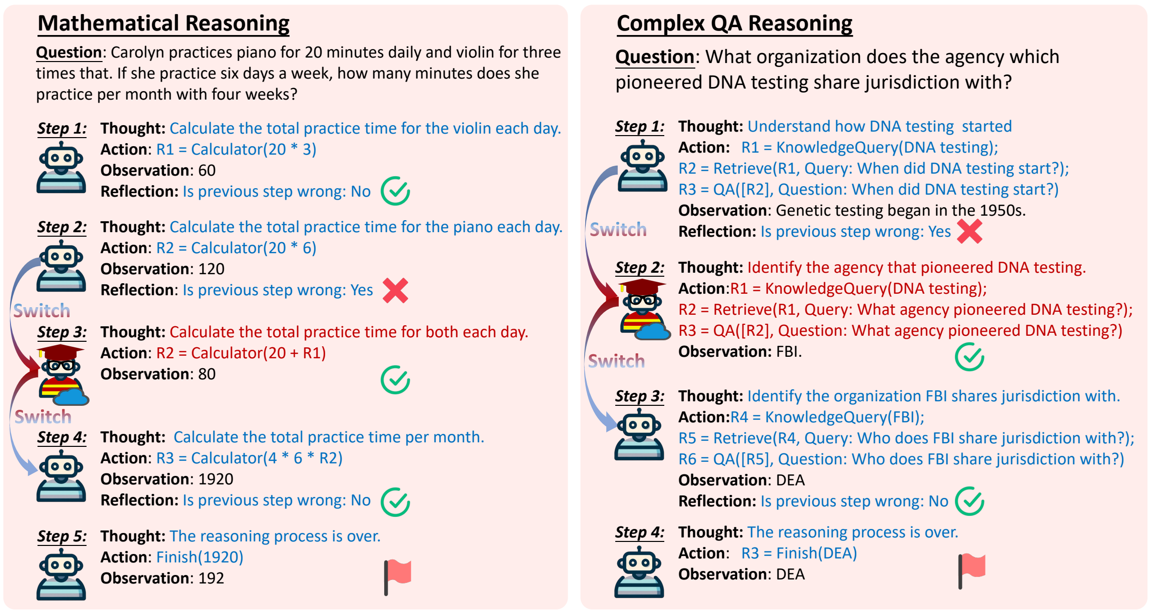
In this section, we examine examples from Mathematical Reasoning and Complex QA Reasoning tasks to illustrate the performance of ADASWITCH.

For instance, in a mathematical reasoning task involving a question about piano practice time, the local agent correctly calculates the total daily practice time for the violin but fails to compute the total practice time for the piano. However, due to its strong reflection capability, the local agent recognizes the error. At this point, a robust cloud agent intervenes and successfully corrects the mistake. The local agent then resumes the generation process based on the corrected input. Similarly, in a complex QA reasoning task, the local agent initially fails to decompose the questions correctly but soon realizes the mistake. The cloud agent intervenes, accurately predicting the next step, and allowing the local agent to continue the inference process until it reaches the correct answer. Through these examples, we can find that the local agent can successfully finish most easy steps while needing intervention from the cloud agent in rare cases, making ADASWITCH meaningful and necessary.

## 4 Related Work

### 4.1 Multi-Agent Colloboration

Enabling multiple models with different characteristics to collaborate can lead to stronger performance or lower computational costs than a single model. $\alpha$-UMi (Shen et al., 2024b) divides a tool learning task that is challenging for a single small LLM into three subtasks, which are delegated to three small LLMs to collaborate on. In some scenarios, choosing the appropriate agent presents a challenge, and Co-LLM (Shen et al., 2024a) achieves token-level agent selection during inference by learning latent variables. AutoMix


**Mathematical Reasoning**

**Question**: Carolyn practices piano for 20 minutes daily and violin for three times that. If she practice six days a week, how many minutes does she practice per month with four weeks?

*Step 1:* **Thought:** Calculate the total practice time for the violin each day.
**Action:** R1 = Calculator(20 * 3)
**Observation:** 60
**Reflection:** Is previous step wrong: No ✓

*Step 2:* **Thought:** Calculate the total practice time for the piano each day.
**Action:** R2 = Calculator(20 * 6)
**Observation:** 120
**Reflection:** Is previous step wrong: Yes ✗

Switch

*Step 3:* **Thought:** Calculate the total practice time for both each day.
**Action:** R2 = Calculator(20 + R1)
**Observation:** 80 ✓

Switch

*Step 4:* **Thought:** Calculate the total practice time per month.
**Action:** R3 = Calculator(4 * 6 * R2)
**Observation:** 1920
**Reflection:** Is previous step wrong: No ✓

*Step 5:* **Thought:** The reasoning process is over.
**Action:** Finish(1920)
**Observation:** 192

**Complex QA Reasoning**

**Question**: What organization does the agency which pioneered DNA testing share jurisdiction with?

*Step 1:* **Thought:** Understand how DNA testing started
**Action:** R1 = KnowledgeQuery(DNA testing);
R2 = Retrieve(R1, Query: When did DNA testing start?);
R3 = QA([R2], Question: When did DNA testing start?)
**Observation**: Genetic testing began in the 1950s.
**Reflection:** Is previous step wrong: Yes ✗

Switch

*Step 2:* **Thought:** Identify the agency that pioneered DNA testing.
**Action:**R1 = KnowledgeQuery(DNA testing);
R2 = Retrieve(R1, Query: What agency pioneered DNA testing?);
R3 = QA([R2], Question: What agency pioneered DNA testing?)
**Observation**: FBI. ✓

Switch

*Step 3:* **Thought:** Identify the organization FBI shares jurisdiction with.
**Action:**R4 = KnowledgeQuery(FBI);
R5 = Retrieve(R4, Query: Who does FBI share jurisdiction with?);
R6 = QA([R5], Question: Who does FBI share jurisdiction with?)
**Observation**: DEA
**Reflection:** Is previous step wrong: No ✓

*Step 4:* **Thought:** The reasoning process is over.
**Action:** R3 = Finish(DEA)
**Observation:** DEA

Figure 5: Case studies of solving Mathematical Reasoning and Complex QA Reasoning problems. Blue text indicates the generation from the local agent while Red text indicates the generation from the cloud agent

(Madaan et al., 2023) uses an external meta-verifier to assess the correctness of small model outputs and decides whether to route to a large model, achieving query-level agent selection. SwiftSage (Lin et al., 2024) makes rule-based judgments based on feedback from the external environment to determine whether to switch from Fast mode to Slow mode, achieving action-level agent selection. Corex (Sun et al., 2023) introduces a suite of strategies designed to enhance the capabilities of LLMs in complex task-solving, with a pivotal focus on advancing multi-model collaboration.

In AdaSwitch, through self-reflection of local agents and spontaneous collaboration with cloud agents, achieves a better balance between cost and performance.

### 4.2 Learning from Mistakes

The methods of learning from mistakes are mainly categorized into two types: prompt-based and finetune-based. For prompt-based methods, TRAN (Yang et al., 2023) summarizes the reasons for past errors of the LLM as rules, forming a set of rules. During the inference stage, the model retrieves rules from the rule set as part of the prompt to assist in model reasoning. LEAP (Zhang et al., 2024) improves the few-shot prompt by intentionally making the model make mistakes when solving few-shot examples, allowing the model to reflect on errors and acquire task-specific principles, which help prevent making similar mistakes in the future. RICP (Sun et al., 2024) proposes to retrieve the relevant insights from previous mistakes and apply hierarchical clustering to the reasons and insights.

For finetune-based methods, LEMA (An et al., 2023) corrects model errors with GPT-4 and uses the correction process as a new dataset for the model to learn self-correction. By adding positive and negative prefixes to correct and incorrect rationals in the training data, mistake tuning (Tong et al., 2024) and NAT (Wang et al., 2024) can enhance model performance in the inference stage using positive prefixes. Wang et al. (2023) proposed method enhances the model's reasoning ability by allowing the model to learn from self-reflection and customized feedback.

In AdaSwitch, we utilize a finetune-based approach to enable the model to self-assess errors, thereby allowing for self-correction or switching to a more powerful LLM for assistance.

## 5 Conclusion

In this work, we advocate AdaSwitch, a novel multi-agent collaboration framework that effectively integrates the strengths of both locally deployed and cloud-based LLMs. The local agent is responsible for less complex reasoning steps, and the cloud agent is dedicated to intricate reasoning steps. Experimental results and in-depth analysis demonstrate that AdaSwitch is able to bring significant improvements in task performance while using much less computational overhead.

## Limitations

In this work, we evaluate our proposed framework on mathematical reasoning and complex question answering tasks, and it remains to be investigated in future work how to extend our approach to a wider range of reasoning tasks. Besides, due to constraints on computational resources and funding, we do not conduct experiments on larger scale language models (>100B). Thus the performance of larger LLMs remains undetermined. We will further explore the performance of our framework on larger scale language models in future research.

## Acknowledgement

This work is supported in part by Ucap Cloud and the State Key Laboratory of General Artificial Intelligence.

## Ethics Statement

This work was conducted in strict compliance with the ACL Ethics Policy. All datasets and large language models (LLMs) used for evaluation are publicly available. Furthermore, our work aims to explore a multi-agent collaboration framework. We do not foresee any negative ethical impacts arising from our work.

## References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. 2023. Learning from mistakes makes llm better reasoner. *arXiv preprint arXiv:2310.20689*.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Tobias Groot and Matias Valdenegro-Toro. 2024. Overconfidence is key: Verbalized uncertainty evaluation in large language and vision-language models. *arXiv preprint arXiv:2405.02917*.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Minillm: Knowledge distillation of large language models. In *The Twelfth International Conference on Learning Representations*.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*.

Kevin J Liang, Weituo Hao, Dinghan Shen, Yufan Zhou, Weizhu Chen, Changyou Chen, and Lawrence Carin. 2020. Mixkd: Towards efficient distillation of large-scale language models. *arXiv preprint arXiv:2011.00593*.

Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. 2024. Swiftsage: A generative agent with fast and slow thinking for complex interactive tasks. *Advances in Neural Information Processing Systems*, 36.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2023. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv preprint arXiv:2306.00978*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*.

Aman Madaan, Pranjal Aggarwal, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. 2023. Automix: Automatically mixing language models. *arXiv preprint arXiv:2310.12963*.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2021. A diverse corpus for evaluating and developing english math word problem solvers. *arXiv preprint arXiv:2106.15772*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Shannon Zejiang Shen, Hunter Lang, Bailin Wang, Yoon Kim, and David Sontag. 2024a. Learning to decode collaboratively with multiple language models. *arXiv preprint arXiv:2403.03870*.

Weizhou Shen, Chenliang Li, Hongzhan Chen, Ming Yan, Xiaojun Quan, Hehong Chen, Ji Zhang, and Fei Huang. 2024b. Small llms are weak tool learners: A multi-llm agent. *arXiv preprint arXiv:2401.07324*.

Hao Sun, Yong Jiang, Bo Wang, Yingyan Hou, Yan Zhang, Pengjun Xie, and Fei Huang. 2024. Retrieved in-context principles from previous mistakes. *arXiv preprint arXiv:2407.05682*.

Qiushi Sun, Zhangyue Yin, Xiang Li, Zhiyong Wu, Xipeng Qiu, and Lingpeng Kong. 2023. Corex: Pushing the boundaries of complex reasoning through multi-model collaboration. *arXiv preprint arXiv:2310.00280*.

Yongqi Tong, Dawei Li, Sizhe Wang, Yujia Wang, Fei Teng, and Jingbo Shang. 2024. Can llms learn from previous mistakes? investigating llms' errors to boost for reasoning. *arXiv preprint arXiv:2403.20046*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Renxi Wang, Haonan Li, Xudong Han, Yixuan Zhang, and Timothy Baldwin. 2024. Learning from failure: Integrating negative examples when fine-tuning large language models as agents. *arXiv preprint arXiv:2402.11651*.

Zhaoyang Wang, Shaohan Huang, Yuxuan Liu, Jiahai Wang, Minghui Song, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, et al. 2023. Democratizing reasoning ability: Tailored learning from large language model. *arXiv preprint arXiv:2310.13332*.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. 2023. Can llms express their uncertainty? an empirical evaluation of confidence elicitation in llms. *arXiv preprint arXiv:2306.13063*.

Haoyan Yang, Yixuan Wang, Xingyin Xu, Hanyuan Zhang, and Yirong Bian. 2024. Can we trust llms? mitigate overconfidence bias in llms through knowledge transfer. *arXiv preprint arXiv:2405.16856*.

Zeyuan Yang, Peng Li, and Yang Liu. 2023. Failures pave the way: Enhancing large language models through tuning-free rule accumulation. *arXiv preprint arXiv:2310.15746*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. 2023. Lumos: Learning agents with unified data, modular design, and open-source llms. *arXiv preprint arXiv:2311.05657*.

Tianjun Zhang, Aman Madaan, Luyu Gao, Steven Zheng, Swaroop Mishra, Yiming Yang, Niket Tandon, and Uri Alon. 2024. In-context principle learning from mistakes. *arXiv preprint arXiv:2402.05403*.

## A Dataset Statistics

The dataset statistics used in this paper is shown in Table 5.

## B Tool Definition

The tool definition is listed in Table 6.

| Method | Mathematical Reasoning | | | | | Complex QA Reasoning | |
|---|---|---|---|---|---|---|---|
| | **GSM8K** | **G_Hard** | **SVAMP** | **ASDIV** | **MultiArith** | **MuSiQue** | **HotpotQA** |
| Train Data | 7,500 instances from GSM8K | | | | | 10,000 instances from MuSiQue | |
| Test Data | 1,000 | 1,000 | 1,000 | 1,000 | 1,000 | 500 | 500 |

Table 5: Datasets Statistics.

| **Task Type** | **Action Types** | **Function Descriptions** | **Tools** |
|---|---|---|---|
| QA | `KnowledgeQuery(Entity) -> Knowledge` | Query the entity knowledge | Wikipedia, Google Search |
| | `ParagraphRetrieval(Knowledge, Query) -> Paragraphs` | Retrieve relevant paragraphs according to the query | `dpr-reader-multiset-base` |
| | `QA(Context, Query) -> Answer` | Answer the query based on the given context | GPT-series/open LLMs |
| | `Calculator(Expression) -> Value` | Calculate given math expressions | WolframAlpha |

(a) Actions used in Complex QA Tasks.

| **Task Type** | **Action Types** | **Function Descriptions** | **Implementation** |
|---|---|---|---|
| Math | `Calculator(Expression) -> Value` | Calculate given math expressions | WolframAlpha |
| | `SetEquation(Expression) -> Equation` | Set equations based on given expressions | |
| | `SolveEquation(Equation) -> Solutions` | Solve the set equations | |
| | `Define(Variable) -> Variable` | Define a variable | |
| | `SolveInequality(Inequality) -> Solutions` | Solve the given inequality | |
| | `Code(Function_Description) -> Code` | Generate codes for math functions | `gpt-3.5-turbo` |
| | `Count(List) -> Number` | Count the element number in a list | Python |

(b) Actions used in Mathematical Tasks.

Table 6: Action interfaces and execution module used in this paper.