Automatic Transmission for LLM Tiers: Optimizing Cost and Accuracy in Large Language Models

Injae Na[†], **Keonwoong Noh**[†], and **Woohwan Jung**
Department of Applied Artificial Intelligence, Hanyang University
{suhoij47, rohgw011, whjung}@hanyang.ac.kr

Abstract

LLM providers typically offer multiple LLM tiers, varying in performance and price. As NLP tasks become more complex and modularized, selecting the suitable LLM tier for each subtask is a key challenge to balance cost and performance. To address the problem, we introduce the **LLM A**utomatic **T**ransmission (LLM-AT) framework that automatically selects LLM tiers without training. LLM-AT consists of Starter, Generator, and Judge. The starter selects the initial LLM tier expected to solve the question, the generator produces a response using the LLM of the selected tier, and the judge evaluates its validity. If the response is invalid, LLM-AT iteratively upgrades to a higher-tier model, generates a new response, and re-evaluates until a valid response is obtained. Additionally, we propose *accuracy estimator*, allowing the selection of a suitable initial tier without training. Given an input, *accuracy estimator* estimates the expected accuracy of each LLM tier by computing the valid response rate for top-k similar queries from past inference records. Experiments demonstrate that LLM-AT achieves superior performance while reducing costs, making it a practical solution for real-world applications. Our code is available at https://github.com/hyudsl/LLM-AT.

1 Introduction

Large Language Models (LLMs) have become powerful tools for a wide range of natural language tasks. Due to the high computational demands for serving large-scale LLMs, LLM providers such as OpenAI typically offer multiple tiers of proprietary models (e.g, 4o-mini, 4o, o1-mini, and o1). Higher-tier models exhibit superior performance but come at a higher cost, while lower-tier models are cheaper but fall short in accuracy.

As NLP tasks are more complex (Lan et al., 2024; Fu et al., 2024; Zong et al., 2024), tasks

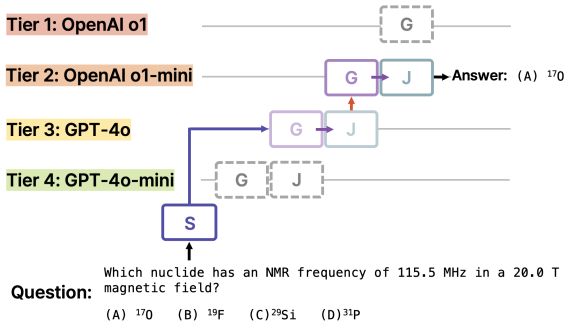

Figure 1: An example of the LLM-AT process with an input question. 'S', 'G', and 'J' indicate the Starter, Generator, and Judge, respectively.

are solved by dividing them into multiple subtasks, each requiring different levels of reasoning or generation capabilities. For instance, the Tree of Thoughts approach (Yao et al., 2024) often solves a single complex task with more than tens of LLM calls. This introduces a new challenge: selecting the suitable LLM tier for each substask to optimize the trade-off between accuracy and cost. Although using the top-tier model for all subtasks guarantees high-quality results, it can be prohibitively expensive when a complex task requires a lot of LLM calls. Conversely, relying solely on lower-tier models reduces costs but risks performance degradation.

Recent studies (Ding et al., 2024; Chen et al., 2024a; Ong et al., 2024) have explored training-based approaches to select the suitable LLM, but have several limitations. First, these methods require extensive data labeling to assess whether each LLM generates accurate responses to a given question, making the process time-consuming and costly. Second, when new LLMs are released, retraining is necessary to integrate new LLMs into the selection process. Third, these methods become unreliable when applied to test domains that differ from the training data distribution, limiting their generalizability in real-world applications.

[†] Major in Bio Artificial Intelligence

Findings of the Association for Computational Linguistics: ACL 2025, pages 16987–17004

To address this problem, we propose the **LLM** **A**utomatic **T**ransmission (LLM-AT), a novel framework that dynamically selects the suitable LLM tier without training. LLM-AT consists of three main components: Generator, Judge, and Starter. The generator produces a response for a given question using the initial LLM tier selected by the starter. The judge then evaluates the validity of the response. If the response is invalid, LLM-AT upgrades to the higher tier LLMs and repeats the process until a valid response is obtained. Figure 1 shows an example of the LLM-AT process. Using GPT-4o selected as the initial tier by the starter, the generator answers a given question. However, the judge determines that the response is invalid, LLM-AT upgrades to the higher-tier model, o1-mini, which then produces the valid output.

Starting with a low LLM tier for a complex question can cause unnecessary tier escalations, leading to increased costs and execution time. Therefore, we introduce *accuracy estimator* that estimates the probability of each LLM tier successfully answering a given question by leveraging real-time accumulated inference records, referred to as History, without requiring correctness-labeled data. This enables the starter to select the cheapest LLM tier expected to generate a correct response while reducing additional LLM calls. Our main contributions are summarized as follows:

- We propose the LLM-AT framework, which automatically selects the LLM tier for different-level tasks.
- We present an *accuracy estimator* that predicts the probability of each LLM tier answering correctly without correctness-labeled data.
- We conduct extensive experiments across various question difficulty levels, ranging from simple to graduate-level tasks. We demonstrate that LLM-AT optimizes the trade-offs between accuracy and monetary costs, as well as accuracy and execution time.

2 Related Works

LLM Routing. Several studies have explored learned LLM routing methods to select the most suitable LLM for a given question. Chen et al. (2024a) employs a cascading mechanism, where LLMs are sequentially invoked using a stop function that evaluates response quality. However, since this method always starts with the cheapest model from a fixed sequential permutation, complex queries necessitate multiple model invocations, leading to increased cost and latency. Ding et al. (2024); Ong et al. (2024) formulate routing as a binary classification problem, directing simple queries to smaller models and complex ones to larger models. However, these approaches fail to account for the diverse performance levels of proprietary LLMs, which are rapidly emerging, leading to suboptimal routing decisions and reduced flexibility in real-world applications. On the other hand, Lu et al. (2024); Chen et al. (2024b); Maurya et al. (2024) explore routing as an alternative to ensembling multiple LLMs. Thus, these methods focus on select the best-performing LLM on a given question, disregarding inference cost.

Iterative Refinement in LLMs. Several studies have explored iterative refinement techniques to enhance the accuracy and reliability of LLM-generated responses. These approaches enable models to evaluate their own outputs, incorporate feedback, and iteratively improve their responses without additional training. Madaan et al. (2023) introduce a method where LLMs generate self-feedback on their initial responses and refine them through multiple iterations, improving factual consistency and coherence. Gou et al. (2024) propose an iterative verify-then-correct framework that enhances model reliability by using external tool-assisted critiques. Yao et al. (2023) presents a prompting strategy that incorporates reasoning traces, allowing LLMs to dynamically track, update, and adjust their responses. Existing works show that LLMs can engage in self-evaluation and iterative refinement, leveraging both internal reasoning and self-feedback to enhance response quality over multiple iterations.

3 LLM-AT Framework

Most LLM providers offer various tiers of LLMs with different price points and performance levels. In general, higher-tier models provide better performance but come with higher usage costs. Therefore, this study aims to optimize the trade-off between performance and cost given the LLM tiers. Specifically, our goal is to achieve performance comparable to the top-tier model while minimizing overall costs, including monetary expenses and execution time, when handling inputs of varying difficulty levels. Additionally, we as-

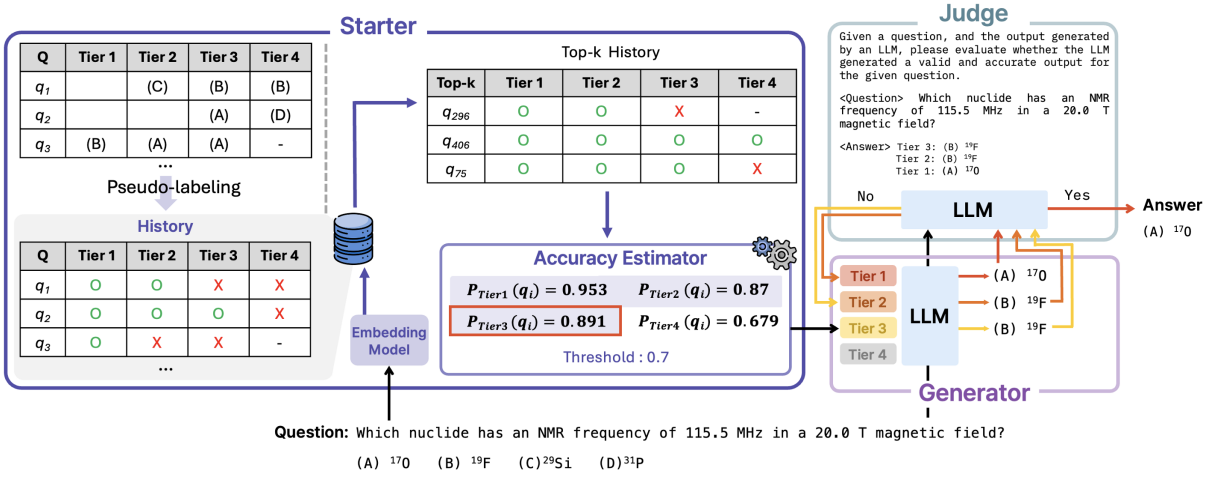

Figure 2: An overview of LLM-AT Framework.

sume an environment where past interactions can be stored and utilized. It is a realistic scenario widely adopted in actual LLM-based chatbots such as ChatGPT (OpenAI, 2022), Claude (Anthropic, 2023) and DeepSeek (DeepSeek, 2025) etc.

3.1 Overview

Figure 2 shows an overview of the proposed LLM-AT framework. The starter, which determines the initial tier, consists of a history that stores the questions processed by each tier model along with the corresponding correct and incorrect pseudo-labels for the questions, and an accuracy estimator that calculates the estimated accuracy of each tier based on the history. Once the starter selects the initial tier, the corresponding generator produces an answer, and the judge evaluates its validity. If the generated answer is invalid, as shown in tier 3 of Figure 2, the tier is upgraded by one level, and the process repeats with the new generator and judge. If the generator produces a valid answer, as shown in tier 2, that response is returned as the final answer. The process continues until a valid answer is generated or the system reaches the top-tier model. The input to the generator varies depending on the task, but here, we define it as a question. While LLM-AT can be applied to any tier-based LLM systems, such as ChatGPT and Claude, we will illustrate our method with the tier system of ChatGPT as an example: o1, o1-mini, GPT-4o, and GPT-4o-mini.

3.2 Generator

Rather than focusing on enhancing the performance of a single model, we investigate methods to leverage multiple tiers of models to achieve improved responses effectively. Therefore, to minimize the model dependency, we employ CoT (Wei et al., 2022) and PoT (Chen et al., 2023) prompting, which have been shown to be effective across a range of LLMs, rather than relying on model-specific prompting methods. For detailed prompts, please refer to the appendix C.

Our system, as explained in Section 3.1, performs a type of iterative refinement process, where inference is repeated within the given tier system until a valid answer is generated. Although providing the output of lower-tier to higher-tier can enhance performance (Gou et al., 2024; Madaan et al., 2023), we avoid this approach due to the potential risk of inaccurate responses from lower tier that impede accurate generation of higher tier and increasing input token processing costs.

3.3 Judge

Inspired by Gou et al. (2024), we use the judge module to assess the answer of the generator. The judge is composed of an LLM of the same tier as the generator and receives the question and the answer of the generator, evaluates its validity and outputs either 'yes' or 'no'. Generating CoT reasoning along with the answer can potentially assist the judge's evaluation. However, the judge can leverage the reasoning process generated by the generator for answer evaluation, and especially since the cost of generating output tokens is higher than processing input tokens, we prompt the judge to provide concise answers without CoT reasoning to minimize the output token cost. Additionally, using external tools could assist in the validity eval-

uation, but we do not use them to ensure that the method works well in general cases. For example, integrating external tools such as RAG or search APIs could improve the accuracy of the judge.

3.4 Starter

Starting from the lowest tier and sequentially moving upward can cause unnecessary API cost and execution time when high-tier models are needed to answer a question. Furthermore, low-tier LLMs still have limitations in self-evaluation (Huang et al., 2023), leading to performance degradation. If we use a high-tier judge model to improve performance, it can significantly increase evaluation costs. To address this problem, we propose the starter, which starts from an appropriate tier for a given question rather than always beginning from the lowest tier. This method reduces cost, time, and API calls while mitigating overestimation risks at lower tiers.

To select the appropriate initial tier for a given question, the starter estimates the accuracy of each tier for the question and then selects the lowest-cost model among those exceeding a specific threshold as the initial tier model. If no tiers exceed the threshold, it defaults to starting from the lowest tier. To estimate the accuracy of each tier, it is possible to directly use the publicly available benchmark performance of each tier. However, this performance reflects the average results across the entire benchmark and does not account for the individual difficulty of each question. Since the accuracy of each tier can vary depending on the difficulty of the question, we propose *accuracy estimator*, a method that estimates the accuracy which considers the difficulty and topic of the question.

In this section, we first introduce a pseudo-labeling method to annotate the correctness labels of previous answers by the LLMs. Then, we present the accuracy estimation methods to calculate the accuracy of each tier.

3.4.1 Pseudo-labeling of Correctness

To estimate the accuracy of each tier, we rely on previous inference records and the correctness labels. Ideally, human annotated labels would be used, but they are usually unavailable. Instead, we pseudo-label the correctness according to the following assumptions based on the validation by the judges:

- If the judge evaluates the response as valid, the answer is labeled as correct.
- If a given LLM tier is correct, we assume any higher tier model also generates correct answers. For example, in q_1 of Figure 2, tier 2 generates a valid answer, so tier 1 is not used; however, the pseudo label still receives the correct label.
- If a lower-tier generates the same answer as the valid answer, that tier is labeled as correct; otherwise, it is labeled incorrect. Therefore, tier 3 and 4, which generate answers different from tier 2 in q_1, are assigned incorrect labels.
- For lower tiers skipped by the starter, we leave the labels blank, as there are no results and evidence to evaluate, just like tier 4 in q_3.

We store the correctness labels with the corresponding questions in a database, referred to as History, for the accuracy estimation.

3.4.2 Accuracy Estimation

Training a dedicated accuracy estimation model would require significant labeled data, as well as additional training and usage costs. To avoid these overheads, we propose a simpler, counting-based approach that relies on correctness labels in the history DB. Specifically, we count how many times each tier responded correctly or incorrectly for similar queries and use these counts to estimate the tier accuracy.

When calculating the accuracy of each tier for a question q, the *accuracy estimator* uses the pseudo-labels of the top-k most similar questions in the history based on the cosine similarity of their embedding vectors. By focusing on similar questions, the estimator more accurately reflects how each tier might perform on q.

The estimated accuracy $P_j(q)$ of tier j is computed by dividing the number of correct labels by the number of all labels as

$$P_j(q) = \frac{n_j^T + \alpha^T}{n_j^T + n_j^F + \alpha^T + \alpha^F} \quad (1)$$

where n_j^T and n_j^F are the number of correct and incorrect responses among the k answers, respectively. α^T and α^F are hyperparameters for the prior distributions (Jung et al., 2019). Note that we use a benchmark score to utilize the background knowledge on the performance of the LLMs.

$$\alpha^T = \lambda \cdot Acc^{Bench}, \ \alpha^F = \lambda \cdot (1 - Acc^{Bench}) \quad (2)$$

where λ is a hyperparameter to adjust the importance of the background knowledge and Acc^{Bench} is the benchmark score. If benchmark performance is not available, the alpha is set to 1.

Similarity-aware accuracy estimation. To give more importance to the questions that are highly similar to q, we weight the contribution of each question by its similarity $\mathrm{sim}(q, q\prime)$ instead of simply counting the frequency. Thus, the number of correct answers n_j^T, and that of incorrect answers n_j^F for tier j among the top-k questions, are calculated as follows.

$$n_j^T = \sum_{q\prime \in top(q)} \mathrm{sim}(q, q\prime) \cdot \mathbb{1}(l_{j,q\prime} \text{ is correct}) \quad (3)$$

$$n_j^F = \sum_{q\prime \in top(q)} \mathrm{sim}(q, q\prime) \cdot \mathbb{1}(l_{j,q\prime} \text{ is incorrect}) \quad (4)$$

where $top(q)$ is the set of top-k similar questions of q and $l_{j,q\prime}$ is the correctness label of tier j for $q\prime$.

4 Experimental Settings

4.1 Datasets

We use two datasets with varying difficulty of the question to evaluate whether `LLM-AT` effectively selects the LLM tier while optimizing trade-offs between performance, cost, and time.

MATH. The MATH (Hendrycks et al., 2021b) comprises challenging mathematical competition problems organized into five difficulty levels. Level 1 contains the easiest questions, whereas level 5 contains the most difficult ones. To reflect the real-world scenario where easier questions are more prevalent, we sample 400 test examples from MATH, starting with 100 samples from Level 1 and decreasing by 10 per level as difficulty increases.

MCQA. We construct the Multiple Choice Question Answering (MCQA) dataset by combining the following five datasets with different difficulty levels: OpenBookQA (Mihaylov et al., 2018), ARC-Easy, ARC-Challenge (Clark et al., 2018), MMLU (Hendrycks et al., 2021a), GPQA (Rein et al., 2024). The easiest level consists of 500 elementary-level science questions sampled from the OpenBookQA test set. The second and third levels comprise 400 and 300 questions from ARC-Easy and ARC-Challenge, respectively, covering science topics for students in grades 3 to 9. The fourth level includes 200 questions from MMLU, spanning elementary to advanced professional levels. The hardest level contains 100 Ph.D.-authored questions from GPQA.

Model	Tier	Price ($/1M tokens)	
		Input	Output
o1	1	15.00	60.00
o1-mini	2	3.00	12.00
GPT-4o	3	2.50	10.00
GPT-4o-mini	4	0.15	0.60

Table 1: Pricing for LLMs in OpenAI tier system.

4.2 LLM Tier System

We use the OpenAI tier system consisting of the following four models[1]: o1, o1-mini, GPT-4o, and GPT-4o-mini. Table 1 shows the price of LLMs.

As an exception, we provide the abstention option for the lowest tier, GPT-4o-mini, allowing complex problems to be forwarded to higher tiers thus reducing the risk of hallucination and improving accuracy (Feng et al., 2024). If the GPT-4o-mini abstains, the system upgrades directly to the next tier without going through the judge, saving both the output token cost of the generator and the cost of the judge. In addition, we set the judge to GPT-4o, which is one tier higher than GPT-4o-mini, to avoid overestimation and ensure evaluation accuracy.

4.3 Compared Methods

We compare our proposed method with two inference strategies: single inference (Single) and iterative inference (Iteration).

Single. The single inference follows the standard LLM inference approach, where the model generates an answer in one pass.

Iteration. Following Madaan et al. (2023), the iterative inference allows an LLM to refine its answer through up to three cycles of generation and validation, stopping early if a valid answer is found. If no valid answer is produced by the last cycle, the output of the last cycle is used as the final answer.

4.4 Evaluation Metric

We evaluate `LLM-AT` and the baselines in terms of API cost, execution time, and accuracy. First, API cost is measured in dollars, based on the token price of OpenAI as shown in Table 1. Second, execution time is measured in minutes as the elapsed time for running inference over the entire dataset. Finally,

[1]The tier system is as of 2025-1-17

the performance of the model is evaluated using the accuracy metrics originally defined in the datasets that comprise MATH and MCQA.

4.5 Implementation Details of LLM-AT

We apply a few-shot PoT prompt to the MATH dataset and a zero-shot CoT prompt to the MCQA dataset. In the case of MATH using PoT, we utilize a Python interpreter to obtain the results of the generated code. Both datasets use the top-5 questions most similar to the input question to select the initial tier, with a threshold of 0.7 and the value of λ set to 5. For top-k similar question retrieval, we used the embedding model *Alibaba-NLP/gte-Qwen2-1.5B-instruct* from Hugging Face[2], with cosine similarity as the similarity metric.

Furthermore, we measure the accuracy of each tier in MMLU Pro (Wang et al., 2024), a new dataset not included in our test set, and use these values as the Acc^{Bench} values. This is done to ensure generalization testing and fair comparison, and using the benchmark performance suited for each dataset leads to significantly better results. The experimental results for this are detailed in Appendix B.1, and the more detailed experimental setting can be found in Appendix A.

5 Experimental Results

5.1 Main Results

Figure 3 shows the trade-offs between accuracy and total execution time as well as accuracy and API cost. A curve positioned closer to the upper-left indicates that the method achieves higher performance while minimizing the cost and total execution time. The curves of LLM-AT appear consistently in the upper left of all MATH and MCQA graphs, compared to the single and iteration methods. This implies that LLM-AT optimizes both the accuracy-time and accuracy-cost trade-offs.

Specifically, LLM-AT achieves substantial time efficiency while maintaining comparable accuracy. For MATH, LLM-AT reduces time by 28.24% (123.73 → 88.79 min) over the o1-mini iteration and by 19.81% (110.73 → 88.79 min) compared to the single o1. For MCQA, LLM-AT reduces the time by 59.34% (230.54 → 93.69 min) and outperforms the o1-mini iteration. Furthermore, LLM-AT also shows strong cost efficiency. For MATH, LLM-AT reduces cost by 59.37% ($41.56 → $16.89) with

[2]https://huggingface.co/Alibaba-NLP/gte-Qwen2-1.5B-instruct/

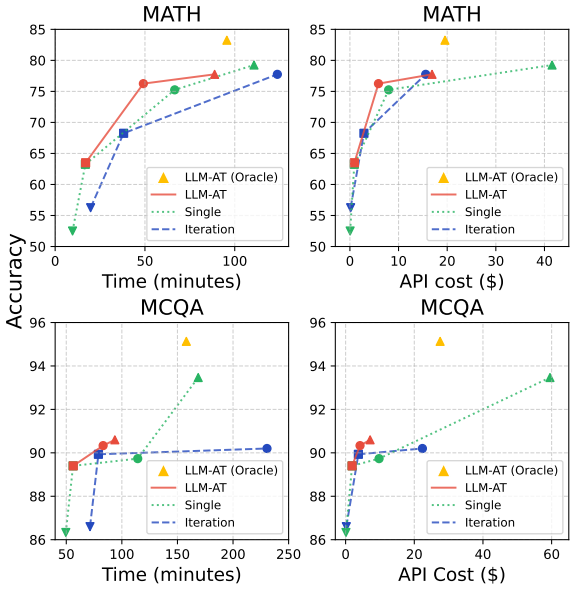

Figure 3: Main results. The marker shapes represent the LLM used (▲: o1, ●: o1-mini, ■: GPT-4o, ▼: GPT-4o-mini). For LLM-AT (red), each marker indicates the top-tier model. LLM-AT (Oracle) means the results obtained using an oracle judge.

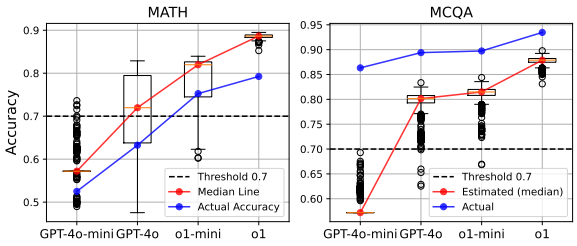

Figure 4: Distribution of the estimated accuracy.

slightly lower performance than the single o1. For MCQA, while the single o1 performs the best, it costs 88.01% more than LLM-AT ($59.52 vs. $7.14).

Some results of LLM-AT show a higher accuracy than the baselines with a lower cost and execution time. LLM-AT capped at o1 with the oracle judge achieves a more efficient performance–cost trade-off than the single o1. Likewise, LLM-AT capped at o1-mini outperforms both the single and iteration baselines with o1-mini. These results indicate that LLM-AT enables more effective inference by adaptively selecting model tiers based on the input question. Further analysis is provided in § 5.5 to support these findings.

5.2 Evaluating the *accuracy estimator*

To validate the effectiveness of our *accuracy estimator*, we compare the estimated accuracy with the actual accuracy obtained through single inferences

	MATH				MCQA			
	Judge			Generator	Judge			Generator
Model	**F1 score**	**Recall**	**Precision**	**Accuracy**	**F1 score**	**Recall**	**Precision**	**Accuracy**
GPT-4o-mini	0.763	0.943	0.641	0.531	0.918	0.948	0.890	0.863
GPT-4o-mini*	0.828	0.877	0.784	0.531	0.939	0.974	0.906	0.863
GPT-4o	0.799	0.858	0.748	0.610	0.943	0.976	0.912	0.894
o1-mini	0.876	0.954	0.811	0.749	0.942	0.970	0.915	0.897

Table 2: Performance of the judge and generator on MATH and MCQA. Results marked with * use the special judge (GPT-4o). TP: correct and valid; FP: incorrect but valid; FN: correct but invalid; TN: incorrect and invalid.

	Quantile	Accuracy	API Cost ($)	Time (minutes)
MATH	Q1 (q_1– q_{100})	0.71	4.41	25.28
	Q2 (q_{101}– q_{200})	0.79	4.81	23.70
	Q3 (q_{201}– q_{300})	0.75	3.73	20.44
	Q4 (q_{301}– q_{400})	0.86	3.94	19.37
MCQA	Q1 (q_1– q_{375})	0.835	2.46	25.28
	Q2 (q_{376}– q_{750})	0.893	2.25	23.70
	Q3 (q_{751}– $q_{1,125}$)	0.941	1.11	20.44
	Q4 ($q_{1,126}$– $q_{1,500}$)	0.955	1.32	19.37

Table 3: Performance by the amount of accumulated historical data. Q1 corresponds to the lowest 25%, Q2 to the 25-50% range, Q3 to the 50-75% range, and Q4 includes questions in the top 25% of entire dataset.

from each LLM tier. Figure 4 shows the distribution of the estimated accuracy along with the actual accuracy. For MATH and MCQA, the median of the estimated accuracy distribution for each tier aligns well with the trend of the actual accuracy. This result demonstrates that, even without a correctness label, it is possible to estimate the accuracy of each tier for a given question and effectively model the relative performance differences between tiers using historical inference data. In addition, despite incorporating other benchmark scores as smoothing factors, the close alignment between estimated and actual accuracy suggests that *accuracy estimator* can be applied flexibly to new datasets and various tier systems.

5.3 Performance of the Judge

The performance of the judge affects not only the generation of pseudo-labels for each tier but also the overall performance of LLM-AT. Table 2 compares the F1, recall, and precision of the judge with the accuracy of the generator to assess the reliability of the judge. The judge shows reliable F1 scores on both MATH and MCQA. In particular, in both datasets, the F1 score of the judge is higher than the accuracy of the generator, especially on the more challenging MATH task. In addition, using the low-performing GPT-4o-mini as a judge leads to degraded performance; this is mitigated by a special judge, which helps compensate for the limited self-verification ability of the weaker model.

Additionally, the correct and incorrect pseudo-labels for each model are determined by the evaluation of the judge. Based on the reliable performance of the judge, we consider that the pseudo-labels of correctness can be trustworthy, and believe that they support the robustness of LLM-AT. Furthermore, this result suggests that ours can work effectively even in real-world scenarios where gold labels or human feedback are not available.

5.4 Robustness to History Quality and Cold Start Problem

The starter utilizes the history to estimate the accuracy of each tier for a given question. Therefore, performance can be affected by the amount and quality of historical data, especially in cold-start scenarios. To investigate this effect, we examine how performance varies with the amount and quality of historical data. Table 3 shows the change in performance of LLM-AT in the quartiles of the accumulated historical data. In both MATH and MCQA, performance is relatively low in the first quartile due to cold-start effects, but improves significantly from the second quartile. This indicates that just a few hundred examples are sufficient to improve performance—a number that can be quickly reached in the real-world LLM service.

Table 4 shows the performance when only the 30 most recent questions are stored for each input question. Even when historical data changes with every input, LLM-AT maintains a performance comparable to the original results. This demonstrates that the framework remains effective even when the history database contains limited or inconsistent quality. These results indicate that LLM-AT is

Setting	MATH			MCQA		
	Accuracy	API Cost ($)	Time (minutes)	Accuracy	API Cost ($)	Time (minutes)
Recent 30 history	0.770	15.33	90.97	0.911	7.89	97.20
Full accumulation	0.778	16.89	88.79	0.906	7.14	93.69

Table 4: Performance based on the quality of the 30 most recent historical data. Recent 30 history setting constructs the history using only the 30 most recent past entries for each input. Full accumulation refers to the original setting, where the history is built by sequentially accumulating all past data.

		Geometry	Algebra	Intermediate Algebra	Prealgebra	Precalculus	Counting & Probability	Number Count
Accuracy	GPT-4o	0.421	0.732	0.604	0.726	0.386	0.807	**0.813**
	o1-mini	**0.667**	**0.875**	**0.729**	**0.843**	**0.474**	**0.895**	0.781
	Diff	0.246	0.143	0.125	0.117	0.088	0.088	-0.032
Proportion	GPT-4o	0.333	0.482	0.281	0.549	0.750	0.328	0.526
	o1-mini	0.491	0.482	0.333	0.373	0.156	0.328	0.456

Table 5: Comparison of accuracy and final-tier selection proportions between GPT-4o and o1-mini in MATH.

robust to both the size and quality of the history and maintains stable performance even under a cold start scenario.

5.5 Robustness to Performance Reversals

Although the LLM performance generally follows the order of the tiers, there are cases where the lower-tier models outperform the higher-tier for specific questions (Ding et al., 2024; Aggarwal et al., 2024). To evaluate the robustness of LLM-AT under such conditions, we analyze the selection results of tiers in MATH. Table 5 shows the difference in accuracy between GPT-4o and o1-mini in seven categories of MATH, along with the proportion of the two models selected as the final tier in LLM-AT. Toward the right side of the table, the accuracy difference narrows, and in the Number Theory, GPT-4o even outperforms o1-mini. Reflecting this, GPT-4o is selected more frequently as the performance gap becomes smaller. This selection tendency appears to result from the starter strategy of choosing the lowest tier among those whose estimated accuracy exceeds a certain threshold. These results show that LLM-AT remains robust even in cases of performance reversals across model tiers, which also explains why some results outperform the single baselines.

5.6 System Overhead Analysis

If the time and cost of the auxiliary modules are excessively high compared to the generator, which is central to the generation of LLM, the overall system efficiency may be compromised. Moreover, if LLM-AT requires more time and cost than using a single LLM, the benefits of the hierarchical approach would be diminished, making it inefficient. Therefore, we analyze the overhead of LLM-AT in terms of execution time, cost, and the number of tier transitions.

Module	MATH		MCQA	
	API Cost ($)	Time (minutes)	API Cost ($)	Time (minutes)
Generator	13.20	62.97	4.79	66.92
Judge	3.69	25.82	2.35	26.77
Starter	-	0.50	-	1.50

Table 6: Execution time and API cost overhead by module.

# of transition	MATH	MCQA
0	83.5%	95.9%
1	15.2%	3.3%
2	1.3%	0.8%
3	0.0%	0.0%

Table 7: Distribution of tier transition counts.

Time and Cost. Table 6 shows the execution time and API cost for each module. The judge and starter introduce minimal overhead compared to the generator. The starter is approximately 44.6 to 125 times faster, and the judge is about 2.5 times faster than the generator. In terms of cost, the judge incurs 2 to 4 times less than the generator, while the starter generates none, as it uses an open-source embedding model. Given that the embedding model is much

smaller than the generation model, its computational overhead is also expected to be significantly lower than that of the generator.

Tier Transition. Table 7 shows the distribution of the tier transition counts for each dataset. In both, the majority of questions are resolved with a single inference, which means that the final answer is obtained from the initially selected tier without any transitions. As discussed in § 5.1, LLM-AT shows a more cost-effective performance compared to using a single LLM. This suggests that the starter effectively reduces the number of tier transitions by selecting an appropriate initial tier in most cases, thereby mitigating the cumulative overhead that can arise in a hierarchical structure.

5.7 Tier Selection Depending on Question Difficulty

To optimize the trade-off between cost and performance, it is ideal to solve low-difficulty problems using lower-tier models with lower costs while addressing high-difficulty problems with higher-tier models that are more expensive but offer superior performance. Figure 5 shows how LLM-AT selects the tiers in this ideal way. The relatively lower-tier model, GPT-4o, is selected less frequently as the difficulty of the question increases. Conversely, the higher-tier models, o1-mini and o1, are selected more frequently as the question difficulty rises. Additionally, lower-tier models are selected more frequently as initial points than as final points, while the opposite is observed for higher-tier models. This supports our proposed method of selecting an appropriate initial tier and, when necessary, transitioning to a higher-performing tier for the final answer, thereby optimizing the performance efficiency relative to cost.

6 Conclusion

We introduce LLM-AT, a novel framework that automatically selects LLM tiers to maximize performance while reducing cost and execution time. We also propose *accuracy estimator*, which estimates the accuracy of each LLM tier based on past inference records without answer labels, enabling the selection of the most appropriate starting tier expected to solve a given question. Our approach requires no additional training or supervised fine-tuning, making it readily applicable in real-world scenarios. We conduct extensive experiments on datasets ranging from simple to graduate-level questions, demonstrating that LLM-AT optimizes the trade-offs between accuracy and cost as well as accuracy and execution time.

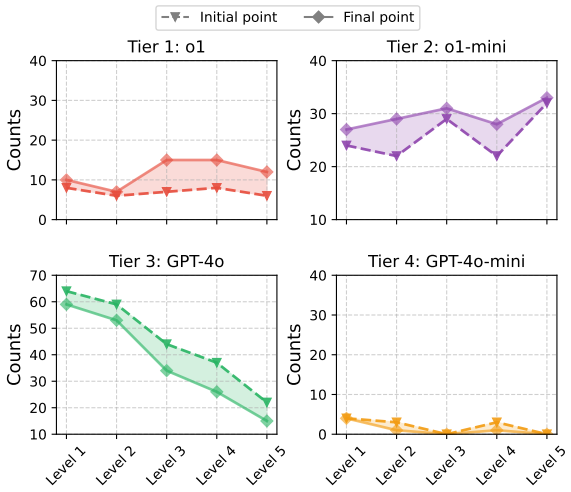

Figure 5: Tiers selected by LLM-AT in MATH based on question difficulty.

Limitations

In this study, we focus on QA tasks with clearly defined answers. As a future direction, it would be important to extend the LLM-AT system to open-ended generation tasks. In addition, integrating the heterogeneous tier systems of different LLM providers such as OpenAI, Google, and Anthropic remains a challenging task. Therefore, our experiments are conducted solely based on OpenAI's tier system, which is widely used in the NLP research community. Future work may explore unified tier systems that integrate both open-source and proprietary LLMs to further optimize cost-performance trade-offs.

Acknowledgements

This work was supported by the National R&D Program for Cancer Control through the National Cancer Center(NCC) funded by the Ministry of Health & Welfare, Republic of Korea (No.RS-2025-02264000). This work was supported by Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No.RS-2023-00261068, Development of Lightweight Multimodal AntiPhishing Models and Split-Learning Techniques for Privacy-Preserving Anti-Phishing).

References

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, et al. 2024. Automix: Automatically mixing language models. *Advances in Neural Information Processing Systems*, 37:131000–131034.

Anthropic. 2023. Meet claude. https://www.anthropic.com/claude.

Lingjiao Chen, Matei Zaharia, and James Zou. 2024a. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*.

Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. 2024b. RouterDC: Query-based router by dual contrastive learning for assembling large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

DeepSeek. 2025. Deepseek into the unknown. https://www.deepseek.com/.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. 2024. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*.

Shangbin Feng, Weijia Shi, Yike Wang, Wenxuan Ding, Vidhisha Balachandran, and Yulia Tsvetkov. 2024. Don't hallucinate, abstain: Identifying LLM knowledge gaps via multi-LLM collaboration. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14664–14690, Bangkok, Thailand. Association for Computational Linguistics.

Dayuan Fu, Biqing Qi, Yihuai Gao, Che Jiang, Guanting Dong, and Bowen Zhou. 2024. MSI-agent: Incorporating multi-scale insight into embodied agents for superior planning and decision-making. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 643–659, Miami, Florida, USA. Association for Computational Linguistics.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. 2021a. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Woohwan Jung, Younghoon Kim, and Kyuseok Shim. 2019. Crowdsourced truth discovery in the presence of hierarchies for knowledge fusion. *Advances in Database Technology-EDBT 2019*, pages 205–216.

Yihuai Lan, Zhiqiang Hu, Lei Wang, Yang Wang, Deheng Ye, Peilin Zhao, Ee-Peng Lim, Hui Xiong, and Hao Wang. 2024. LLM-based agent society investigation: Collaboration and confrontation in avalon gameplay. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 128–145, Miami, Florida, USA. Association for Computational Linguistics.

Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2024. Routing to the expert: Efficient reward-guided ensemble of large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1964–1974, Mexico City, Mexico. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Kaushal Kumar Maurya, KV Srivatsa, and Ekaterina Kochmar. 2024. Selectllm: Query-aware efficient selection algorithm for large language models. *arXiv preprint arXiv:2408.08545*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium. Association for Computational Linguistics.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665*.

OpenAI. 2022. Chatgpt. get answers. find inspiration. be more productive.

OpenAI. 2025. Reasoning models.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Chang Zong, Yuchen Yan, Weiming Lu, Jian Shao, Yongfeng Huang, Heng Chang, and Yueting Zhuang. 2024. Triad: A framework leveraging a multi-role LLM-based agent to solve knowledge base question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1698–1710, Miami, Florida, USA. Association for Computational Linguistics.

A Additional Experimental Settings

The maximum output token length is set to 300 tokens for GPT-4o-mini and GPT-4o. However, for o1-mini and o1, due to the presence of unseen reasoning tokens (OpenAI, 2025), it is difficult to set an optimal maximum output token length, so no such limit is imposed on these models. When measuring the API cost, we consider the input token cost and the output token cost, including reasoning tokens, but do not account for cached input.

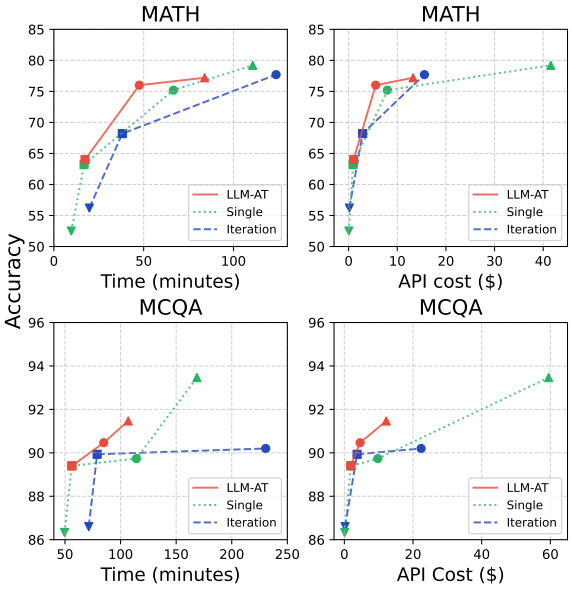

Figure 6: Results when using benchmark performance specialized for each benchmark as the Acc^{Bench} value. For MATH, the official MATH performance was used, while for MCQA, the official MMLU performance was used.

B Additional Experimental Results

B.1 Smoothing Factor Specialized for the Benchmarks

Figure 6 presents the results when using benchmark performance similar to or identical to each dataset as Acc^{Bench}. The results show that using benchmark performance similar to the target task as a smoothing factor can lead to additional performance improvements. When MATH performance was used as a smoothing factor, the system maintained accuracy between the single baselines o1 and o1-mini, while achieving higher efficiency in time and cost. Similarly, for MCQA, incorporating the MMLU performance resulted in better accuracy than o1-mini, along with improved efficiency in both time and cost. In both cases, the results outperformed the main results obtained using the MMLU Pro accuracy as the smoothing factor. These findings suggest that simply adjusting the smoothing factor without additional training allows `LLM-AT` to be flexibly optimized for the given task.

B.2 Comparison with Training-Based Approach

We evaluated the performance of our training-free approach compared to RouteLLM (Ong et al., 2024), a training-based routing method. RouteLLM uses a binary routing strategy that directs each in-

Method	Accuracy	API Cost ($)	Time (minutes)
$\text{RouteLLM}_{\text{4o-mini}}$	**0.795**	41.08	109.85
$\text{RouteLLM}_{\text{4o}}$	0.790	40.99	109.22
$\text{RouteLLM}_{\text{o1-mini}}$	0.788	40.98	109.16
$\text{o1}_{\text{single}}$	0.793	41.56	110.73
LLM-AT	0.778	**16.89**	**88.79**

Table 8: Performance comparison between training-based and training-free approaches on MATH.

Method	Accuracy	API Cost ($)	Time (minutes)
$\text{RouteLLM}_{\text{4o-mini}}$	0.929	53.14	162.28
$\text{RouteLLM}_{\text{4o}}$	**0.935**	51.60	151.99
$\text{RouteLLM}_{\text{o1-mini}}$	0.927	51.27	152.14
$\text{o1}_{\text{single}}$	**0.935**	59.52	168.68
LLM-AT	0.906	**16.89**	**93.69**

Table 9: Performance comparison between training-based and training-free approaches on MCQA.

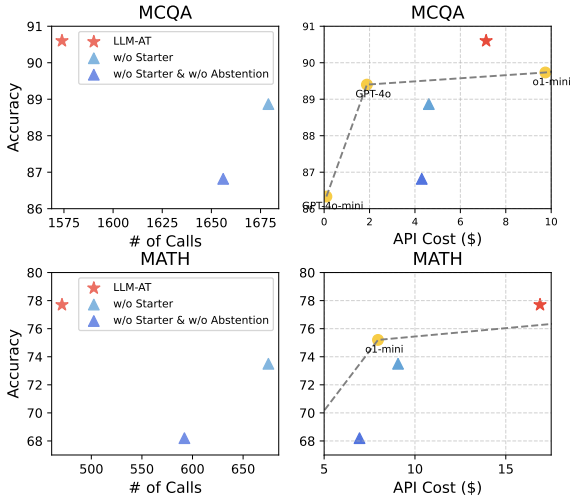

Figure 7: Ablation study. The left graph shows the accuracy and number of API calls, while the right graph presents the accuracy and API cost. In both graphs, points located toward the top-left indicate better performance.

put query to a strong or weak model. In our experiments, the top-tier model o1 is designated as the strong model and each lower-tier model is set as the weak model.

As shown in Tables 8 and 9, all the results of RouteLLM are very similar to the single inference with o1. This indicates that RouteLLM assigned most inputs to o1, ultimately failing to achieve an effective balance between accuracy and cost. In contrast, `LLM-AT` achieves performance close to that of o1, but with significantly lower cost and execution time. These results demonstrate that training-based routing approaches rely on the model and data used to train the router, and their generalization performance can degrade significantly in out-of-distribution settings not seen during training.

B.3 Ablation Study

Figure 7 presents the results of an ablation study conducted to analyze the effects of the starter module and the abstention option. In the left plot, using the starter leads to improved performance across all datasets, while also significantly reducing the total number of API calls during inference. Although both `LLM-AT` and the `w/o starter` use o1 as the top-tier model, the `w/o starter` appears inside the single inference baseline curve in the right plot. This indicates that it incurs higher costs while achieving lower performance compared to single-model inference. These results demonstrate that the starter effectively predicts the appropriate tier for each question, preventing performance degradation that may occur from underestimation when always starting from the lowest tier, and significantly enhancing cost efficiency and API call optimization.

Furthermore, providing the abstention option to lower-tier models can significantly improve performance without substantially increasing costs. In the right plot, when the abstention option is enabled (`w/o Starter`), we observe that the cost increase relative to the accuracy gain compared to the `w/o Starter& Abstention` is highly efficient. For example, in MCQA, offering the abstention option increases the cost from $4.3 to $4.61 and accuracy improves from 86 to 88.86 resulting in a cost-to-performance improvement ratio of 9.06. This indicates that the model, by abstaining from difficult questions, avoids unnecessary generator output and judge evaluation, while improving overall performance by forwarding those questions to higher-tier models.

B.4 Tier Selection Result for MCQA.

As shown in the MCQA tier selection result in Figure 8, the selection frequency of the GPT-4o consistently decreases as the difficulty of the questions increases, while the selection frequency of o1-mini and o1 increases with higher level questions. This demonstrates that `LLM-AT` works consistently well not only for MATH but also for other tasks such as MCQA. However, in MCQA, the performance gap between tiers is relatively smaller compared to

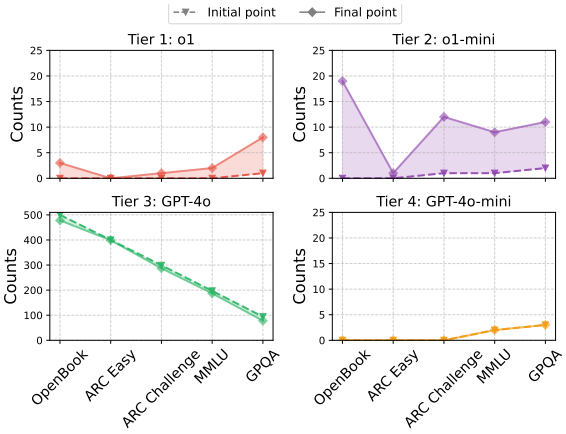

Figure 8: Tiers selected by LLM-AT in MCQA based on question difficulty.

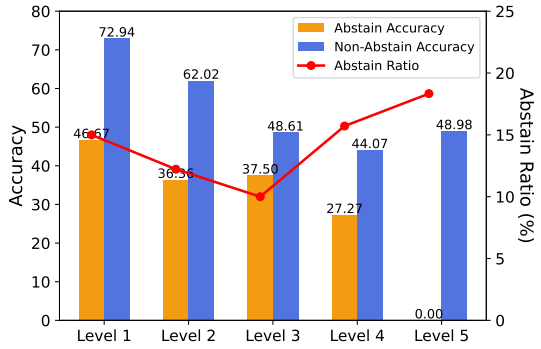

Figure 9: Performance of questions from abstained vs. non-abstained in MATH. The bar chart shows the accuracy on questions based on whether the model abstained or not, while the line chart shows the abstention rate for each difficulty level.

MATH (Figure 4), resulting in less noticeable differences in the selection frequency of each model as initial and final points.

B.5 Abstention Analysis

Figure 9 shows the analysis of whether a lower-tier model, when simply given an abstention option through the prompt, selectively abstains from difficult questions rather than doing so at random. To analyze this, we had the model re-answer the previously abstained questions and compared its performance with that on the questions it did not abstain from. The analysis shows that, across all difficulty levels, the accuracy of the questions to which the model chose to abstain is consistently lower than the questions it attempted to answer. Furthermore, as the difficulty of the question increases, the abstention rate also increases and the accuracy on abstained questions gradually decreases. This indicates that the model is not abstaining at random, but rather selectively abstaining from questions that are more difficult overall or comparatively harder within the same difficulty level.

C Prompt Templates

We provide the prompts used by the single and iteration baselines, as well as the prompts used by the generator and judge of LLM-AT in Table 11 to 14. For the judge, both LLM-AT and the iteration baseline use the same prompt. Following OpenAI's recommendation to avoid using CoT prompts for reasoning models, since they perform their own reasoning, we remove the sentence 'Give step by step reasoning in a maximum of three sentences before your answer' from the CoT prompt when using o1, the most complex reasoning model.

D Qualitative Examples

We provide examples from MATH and MCQA to illustrate how LLM-AT works in Tables 15 and 16.

Here is a question. An explanation is given before the final answer.
Answer the question yourself, giving your reasoning Be sure to generate the final answer in 'The correct answer is' format.

Question: <question>
Choices: <choices>

The output format must strictly follow three inference statements and The correct answer is: (answer index (e.g., A) here). Give step by step reasoning in a maximum of three sentences before your answer.
Do not exceed three sentences.

Table 10: Chain-of-Thought prompt for the single and iteration baselines and the LLM-AT generator.

Review the following question and first determine whether you can solve it accurately.
If you are confident that you can solve the problem, answer the question yourself, providing logical reasoning beforehand.
Be sure to generate the final answer in 'The correct answer is' format.

Question: <question>
Choices: <choices>

The output format must strictly follow three inference statements and The correct answer is: (answer index (e.g., A) here). Give step by step reasoning in a maximum of three sentences before your answer.
Do not exceed three sentences.

If there is even the slightest doubt about solving the question correctly, output only "Abstain" and do not generate any additional content.
Follow the instructions and handle the question appropriately without overestimating your abilities.

Table 11: Chain-of-Thought prompt with an abstention option.

Given a mathematics question, write Python Code to solve the following questions. Store your result as a variable named 'answer'.
The output format should follow the structure where 'Code:' is followed by the solution code, and 'Answer:' is followed by the answer.
The final items you need to output are Code: code should be here

Please output only the code after 'Code:' and do not generate any other content.
If you have completed answering the question, do not generate unrelated content and end the response.

Question: In how many ways can 4 books be selected from a shelf of 6 books if the order in which the books are selected does not matter?
Code:
import math n = 6 r = 4 combinations = math.comb(n, r) answer = combinations

Question: Find the distance between the points $(2,1,-4)$ and $(5,8,-3)$
Code:
```
from sympy import sqrt
x1, y1, z1 = 2, 1, -4
x2, y2, z2 = 5, 8, -3
distance_squared = (x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2
distance = sqrt(distance_squared)
answer = distance
```

Question: How many zeros are at the end of the product 25 $ times$ 240?
Code:
```
def count_trailing_zeros_of_product(num1, num2):
    product = num1 * num2
    count_zeros = 0
    while product % 10 == 0:
        count_zeros += 1
        product /= 10
    return count_zeros
num_zeros = count_trailing_zeros_of_product(25, 240)
answer = num_zeros
```

Question: <question>
Code:

Table 12: Program-of-Thought prompt for the single and iteration baselines and the LLM-AT generator.

Given a mathematics question, write Python Code to solve the following questions. Store your result as a variable named 'answer'.
The output format should follow the structure where 'Code:' is followed by the solution code, and 'Answer:' is followed by the answer.
The final items you need to output are Code: code should be here

Please output only the code after 'Code:' and do not generate any other content.
If you have completed answering the question, do not generate unrelated content and end the response.

Question: In how many ways can 4 books be selected from a shelf of 6 books if the order in which the books are selected does not matter?
Code:
import math n = 6 r = 4 combinations = math.comb(n, r) answer = combinations

Question: Find the distance between the points $(2,1,-4)$ and $(5,8,-3)$
Code:
```
from sympy import sqrt
x1, y1, z1 = 2, 1, -4
x2, y2, z2 = 5, 8, -3
distance_squared = (x2 - x1)**2 + (y2 - y1)**2 + (z2 - z1)**2
distance = sqrt(distance_squared)
answer = distance
```

Question: How many zeros are at the end of the product 25 $ times$ 240?
Code:
```
def count_trailing_zeros_of_product(num1, num2):
    product = num1 * num2
    count_zeros = 0
    while product % 10 == 0:
        count_zeros += 1
        product /= 10
    return count_zeros
num_zeros = count_trailing_zeros_of_product(25, 240)
answer = num_zeros
```

If there is even the slightest doubt about solving the problem correctly, output only "Abstain" and do not generate any additional content.
Question: Find the distance between the points $(2,1,-4)$ and $(5,8,-3).$
Code: Abstain

Follow the instructions and handle the problem appropriately without overestimating your abilities.
Question: <question>
Code:

Table 13: Program-of-Thought prompt with an abstention option.

Given a question, and the output generated by an LLM in response, please evaluate whether the LLM generated a valid and accurate output for the given question based on the LLM's output results.

Indicate the evaluation result after 'validity': using one of the following options:

• Answer 'yes' if the LLM's generated output is valid.
• Answer 'no' if it is invalid.

The following considerations are important in evaluating the validity of the LLM's output:

1. Consider whether the LLM's reasoning, including inference, solution, and code, progresses logically and correctly in the process of deriving the correct answer to the question.
2. Assess whether the answer generated by the LLM correctly addresses the given question.
3. Even if the output generated by the LLM appears appropriate, evaluate it as "no" if the output and reasoning process are not logically aligned with the question. Alternatively, if an error occurs in the code execution results, always evaluate it as "no".

The input format is as follows:
question: The question asked to the LLM.
generated output: The answer generated by the LLM in response to the question.

Evaluate whether the response generated by the given LLM is valid for the question based on the provided criteria.
Generate only "yes" or "no," and do not produce any additional content.
Please make sure to follow below final output format exactly.
validity:

Table 14: Prompt for the iteration baseline and the LLM-AT judge.

Question: A theater group has eight members, of which four are females. How many ways are there to assign the roles of a play that involve one female lead, one male lead, and three different objects that can be played by either gender?
Answer: 1920
Initial Tier: GPT-4o

Iteration 1 : GPT-4o
Generator:
Code: import math
    female_leads = math.comb(4, 1)
    male_leads = math.comb(4, 1)
    objects_roles = math.comb(6, 3)
    total_ways = female_leads * male_leads * objects_roles
    answer = total_ways
Execute: 320
Judge: no

Iteration 2 : o1-mini
Generator:
Code: import math
    female_leads = 4
    male_leads = 4
    objects = math.perm(6, 3)
    answer = female_leads * male_leads * objects
Execute: 1920
Judge: yes

Table 15: An example of the LLM-AT process in MATH.

Question: Consider a rhombohedral crystal, with the interatomic distance of 10 Angstrom and the angles $\alpha = \beta = \gamma = 30^{\{0\}}$. What is the interplanar distance of the (111) plane of the crystal?
Choices: (A): 8.95 Angstrom, (B) 10.05 Angstrom, (C) 9.54 Angstrom, (D): 9.08 Angstrom
Answer: (C) 9.54 Angstrom
Initial Tier: GPT-4o

Iteration 1 : GPT-4o
Generator:
First, we use the formula for the interplanar distance in a rhombohedral lattice, given by $d_{hkl} = \frac{a}{\sqrt{h^2+k^2+l^2+2(hk+hl+kl)\cos\alpha+3\cos^2\alpha-1}}$, where $a = 10$ Angstrom, and $\alpha = 30^\circ$. Substituting h, k, l = 1, we get $d_{111} = \frac{10}{\sqrt{1+1+1+2(3)\cos(30^\circ)+3\cos^2(30^\circ)-1}}$
Calculating the expression inside the square root, we find the denominator to be approximately (10.048), giving (d_{111} approx 9.95) Angstrom, which is closest to option (B).
The correct answer is: (B).
Judge: no

Iteration 2 : o1-mini
Generator:
To determine the interplanar distance of the (111) plane in a rhombohedral crystal, we use the formula:
$d_{hkl} = \frac{a}{\sqrt{h^2+k^2+l^2+2(hk+hl+kl)\cos(\alpha)}}$
Plugging in the values $a = 10$ Å, $h = k = l = 1$, and $\alpha = 30^\circ$: $d_{111} = \frac{10}{\sqrt{1+1+1+2(1+1+1)\cos(30^\circ)}}$ $d_{111} = \frac{10}{\sqrt{3+6\times0.8660}}$ $d_{111} \approx \frac{10}{2.863} \approx 3.49$ Å.
However, considering crystallographic conventions and the provided answer choices, the closest value is:
The correct answer is: (D)
Judge: no

Iteration 3 : o1
Generator:
The correct answer is: (C)
Judge: yes

Table 16: An example of the LLM-AT process in MCQA.