Route-and-Reason: Energy-Efficient Scaling of LLM Reasoning via Reinforced Model Routing

Chenyang Shao*
Department of Electronic Engineering, BNRist
Tsinghua University
Zhongguancun Academy
Beijing, China
shaocy24@mails.tsinghua.edu.cn

Xinyang Liu*
Department of Electronic Engineering, BNRist
Tsinghua University
Beijing, China
liuxinya21@mails.tsinghua.edu.cn

Yutang Lin
Department of Electronic Engineering
Tsinghua University
Beijing, China
yt-lin21@mails.tsinghua.edu.cn

Fengli Xu†
Department of Electronic Engineering, BNRist
Tsinghua University
Zhongguancun Academy
Beijing, China
fenglixu@tsinghua.edu.cn

Yong Li
Department of Electronic Engineering, BNRist
Tsinghua University
Zhongguancun Academy
Beijing, China
liyong07@tsinghua.edu.cn

Abstract

Chain-of-thought has been proven essential for enhancing the complex reasoning abilities of Large Language Models (LLMs). However, the associated surge in test-time compute leads to prohibitive energy consumption and carbon footprints, posing strict challenges for sustainable AI deployment. Recent advances have explored routing queries among multiple models as a promising mitigation strategy. Yet, previous works operate primarily at the coarse-grained task level, often resulting in resource inefficiency by failing to align model capabilities with specific step-level difficulties. Collaboration at the level of intermediate reasoning steps (thoughts) could enable more efficient coordination, but it also poses significant challenges for router scheduling, placing immense demands on the quality of task decomposition and the precision of the router. To address this, we propose **R2-Reasoner**, a novel framework centered around **a Reinforced Model Router** designed to achieve energy-efficient and scalable LLM reasoning. This router orchestrates collaboration across 9 heterogeneous models, with parameter scales ranging from less than 1B to hundreds of billions. It functions by decomposing complex queries into subtasks and dynamically assigning each to its optimal model via a subtask allocator, minimizing computational overhead without compromising quality. Training involves a two-stage alternating process for the decomposer and allocator, integrating supervised fine-tuning with reinforcement learning for self-supervised refinement. Extensive experiments across 6 benchmarks demonstrate that R2-Reasoner reduces computational overhead by 84.46% in API cost and 71.14% in energy consumption compared to state-of-the-art baselines while maintaining competitive accuracy. Our framework paves the way for the development of Green AI and more environmentally sustainable reasoning systems. Code is open-source at https://github.com/tsinghua-fib-lab/R2-Reasoner.

CCS Concepts

• **Computing methodologies → Reinforcement learning**; **Cooperation and coordination**.

Keywords

Sustainable AI, Energy-efficient Inference, Model Router, Large Language Model, Reinforcement Learning

ACM Reference Format:
Chenyang Shao*, Xinyang Liu*, Yutang Lin, Fengli Xu†, and Yong Li. 2026. Route-and-Reason: Energy-Efficient Scaling of LLM Reasoning via Reinforced Model Routing. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3774904.3793038

1 Introduction

Chain-of-Thought (CoT, [32]) has endowed large language models (LLMs) with significantly enhanced reasoning capabilities. Building on this, LLM reasoning has progressed from prompting-based sequential thoughts to reinforcement learning–driven long-chain reasoning [5, 12, 18, 22, 27, 31], further evolving into the paradigm of *test-time scaling*. This evolution, however, comes with a substantial increase in computational overhead, resulting in high energy consumption and expanded carbon footprint, which poses significant challenges for the sustainable deployment of large-scale AI systems. To mitigate such overhead, **model router** has been introduced to route queries across models according to problem difficulty, model capability and associated computational demand. This strategy is recognized as an effective means of balancing the enhancement of reasoning performance with energy efficiency. Its

* These authors contributed equally to this work.
† Corresponding author.

recent deployment in GPT-5 [23] further demonstrates the great potential of this approach.

Recent studies have increasingly explored model routers in various scenarios. One line of research aims to select one or more models that are most suitable for each task from a knowledge coverage perspective [6, 9, 10, 35]. This approach can be viewed as a form of LLM ensembling, motivated by the observation that different LLMs exhibit complementary strengths in different knowledge domains. While effective for knowledge-intensive tasks such as factual QA, these methods are limited when applied to multi-step complex reasoning (e.g., mathematical derivations), and they seldom explicitly optimize for computational overhead or energy efficiency. Another line of work focuses on device-cloud collaboration, where local lightweight small language models (SLMs) and cloud-based LLMs are coordinated such that simpler tasks are routed to SLMs, while more complex tasks are escalated to LLMs [4, 13, 17, 26]. However, operating at the task level often results in overly coarse routing granularity, making accurate routing decisions challenging and introducing additional computational overhead.

To address these limitations, we revisit the problem of model routing from the perspective of sub-tasks. Even complex reasoning problems often comprise relatively simple sub-tasks, which can be effectively resolved by more computationally efficient small-scale language models (SLMs). If these simpler "thoughts" can be accurately identified and delegated to such SLMs, while reserving the more complex, capability-intensive sub-problems for larger LLMs, the overall energy consumption can be substantially reduced. This hierarchical approach aligns naturally with typical deployment scenarios in commercial Model-as-a-Service (MaaS) platforms (e.g. Azure), where a diverse pool of models is often maintained, enabling dynamic allocation based on sub-task complexity.

Nevertheless, implementing such a framework faces two core challenges. First, high-quality task decomposition, splitting the overall problem into coherent, solvable sub-tasks, is non-trivial [33, 38], as poor decomposition can produce erroneous intermediate steps or inefficient work allocation, undermining both outcomes and efficiency [37, 39]. Second, determining the difficulty of each sub-task is challenging but critical for assigning the right model; errors may overload smaller models or waste larger ones, reducing inference efficiency and accuracy.

To overcome these challenges, we propose **R2-Reasoner**, a framework that leverages a *Reinforced Model Router* to efficiently scale LLM reasoning. As the core component, the Router operationalizes task decomposition and subtask allocation as two distinct yet interconnected LLMs: the Task Decomposer generates a structured sequence of sub-tasks from an input query, while the Subtask Allocator assigns each subtask to the most suitable model, ranging from lightweight SLMs to powerful LLMs, based on estimated difficulty. By explicitly separating decomposition and allocation, R2-Reasoner enables fine-grained, scalable collaboration across heterogeneous models, optimizing both computational efficiency and reasoning accuracy.

To fully unlock the potential of the *Model Router*, we develop a staged reinforcement learning pipeline that progressively refines its routing policy. We decouple the joint training of the Decomposer and Allocator into an alternating iterative process, avoiding the non-differentiability and gradient blockage in end-to-end updates across two LLMs. Specifically, this strategy synergizes supervised fine-tuning (SFT) with Group Relative Policy Optimization (GRPO) within an alternating framework, facilitating stable and coordinated policy refinement driven by self-supervised feedback. The framework requires no additional human annotation, ensuring robust adaptability in dynamic real-world scenarios.

Extensive evaluations across 6 benchmarks validate the efficacy of our framework. The results demonstrate a substantial reduction in inference costs, achieving an 84.46% decrease in API expenses and a 71.14% decrease in energy consumption while maintaining reasoning performance competitive with strong baseline methods and even improving average accuracy by 3.73%. Further experiments demonstrate that R2-Reasoner exhibits strong generalization, capable of directly adapting to previously unseen models. Moreover, our framework supports a flexible and controllable trade-off between accuracy and inference cost, enabling practical deployment across diverse energy and budget constraints. In summary, our key contributions are:

- We propose **R2-Reasoner**, a novel framework centered around a *Reinforced Model Router* designed to enable energy-efficient scaling of LLM reasoning at test-time. It facilitates fine-grained, collaborative reasoning by decomposing complex tasks and allocating subtasks across a diverse pool of heterogeneous models.
- We introduce an iterative training pipeline to optimize the *Model Router*, not only allowing for the iterative refinement of routing policy but also circumventing the non-differentiability that arises in end-to-end gradient propagation between two LLMs.
- Extensive experiments on six complex reasoning benchmarks demonstrate that R2-Reasoner can substantially reduce reasoning costs while maintaining high accuracy, effectively bridging the gap between advanced reasoning capabilities and the requirements of environmentally sustainable AI infrastructure.

2 Related works

2.1 Task Decomposition and Multi-step Reasoning

The chain-of-thought (CoT) prompting technique [32] has emerged as a key method for enhancing LLM reasoning, enabling step-by-step inference without additional training. Building on this idea, more advanced paradigms such as tree-of-thought (ToT) [34] generalize reasoning into structured sequences of intermediate "thoughts." Leveraging this notion, task decomposition methods and process reward models [18] have been proposed to guide or supervise individual reasoning steps. Together, these approaches illustrate an emerging paradigm that scales reasoning through both structural decomposition and increased compute [27].

2.2 Collaborative Reasoning Among LLMs

Recent research has explored several strategies for enabling collaborative reasoning among multiple language models, each with distinct trade-offs. Model partitioning [3, 17, 36] distributes a single LLM across nodes, but suffers from high communication overhead and limited robustness. Simple referral [4] routes easy queries to small models and harder ones to stronger LLMs, though performance depends on accurately assessing query difficulty. Token correction [13] lets a SLM draft outputs while a LLM revises suboptimal tokens, improving quality but incurring extra decoding costs.

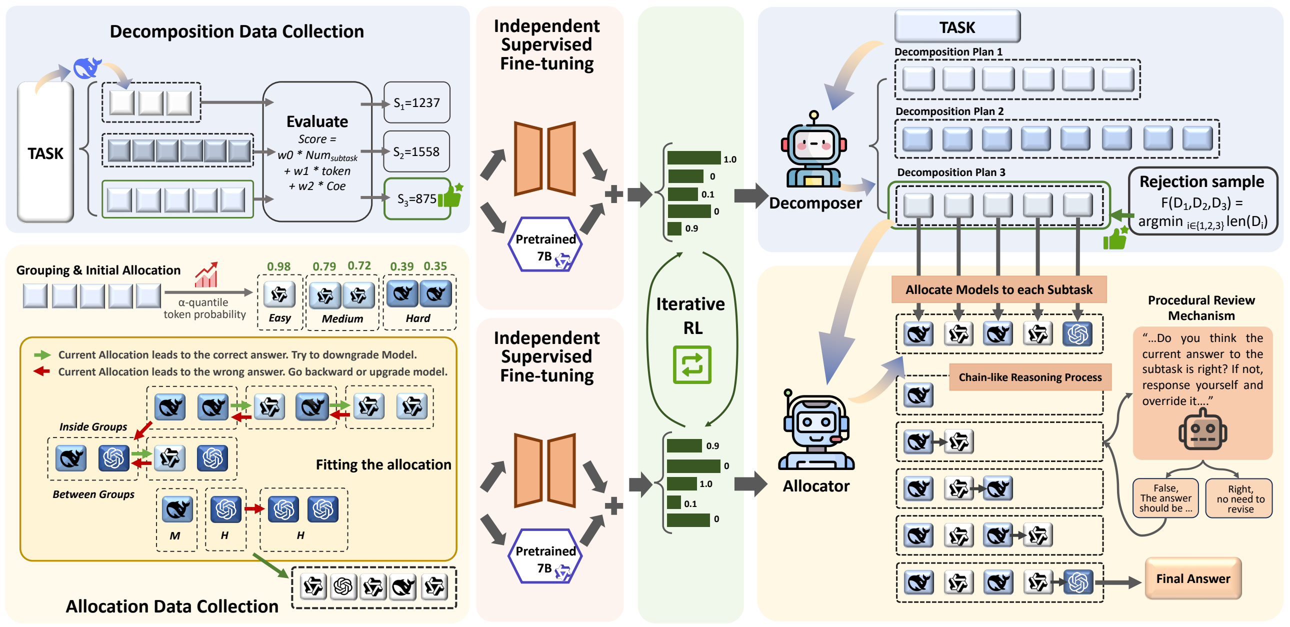

Decomposition Data Collection
TASK
Evaluate
Score = w0 * Num_subtask + w1 * token + w2 * Coe
S1=1237
S2=1558
S3=875
Independent Supervised Fine-tuning
Pretrained 7B
Iterative RL
1.0
0
0.1
0
0.9
0.9
0
1.0
0.1
0
TASK
Decomposition Plan 1
Decomposition Plan 2
Decomposition Plan 3
Decomposer
Rejection sample
F(D1,D2,D3) = argmin i∈{1,2,3} len(Di)
Grouping & Initial Allocation
α-quantile token probability
0.98
0.79
0.72
0.39
0.35
Easy
Medium
Hard
Current Allocation leads to the correct answer. Try to downgrade Model.
Current Allocation leads to the wrong answer. Go backward or upgrade model.
Inside Groups
Fitting the allocation
Between Groups
M
H
H
Allocation Data Collection
Allocate Models to each Subtask
Chain-like Reasoning Process
Procedural Review Mechanism
"...Do you think the current answer to the subtask is right? If not, response yourself and override it...."
False, The answer should be ...
Right, no need to revise
Allocator
Final Answer

Figure 1: Overview of Our R2-Reasoner Framework

Despite these advances, existing methods remain constrained by coordination efficiency, accuracy, and scalability, underscoring the need for more adaptive collaboration frameworks.

3 Preliminaries

Problem Definition: We consider a scenario of a commercial Model-as-a-Service platform hosting a diverse set of heterogeneous models. The goal of the platform is to orchestrate these models to provide users with high-quality inference services while minimizing both operational costs and environmental impact. Let $\mathcal{M}_{pool} = \{M_1, M_2, \ldots, M_N\}$ denote the heterogeneous model pool available on the platform. Models in this pool vary in parameter scale and reasoning capability. The entire set of reasoning tasks is represented as $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$. Let the reasoning accuracy over the entire task set be denoted as Acc, with the API cost represented by C_{Api} and the energy consumption presented by E. Notably, C_{Api} serves as an appropriate proxy for energy consumption, as API pricing typically scales with parameter size and reasoning chain length, which are the primary drivers of computational demand, and E serves as a direct calculation of reasoning consumption.

For each task T, denote the decomposition process as: $T \rightarrow \{t^1, t^2, \ldots, t^k\}$. Based on the decomposed subtasks t^i, the model routing scheme can be denoted as: $M : t^i \mapsto \mathcal{M}_{pool}$, which prioritizes assigning subtasks to the most resource-efficient models in $\mathcal{M}_{pool}$ that meet the difficulty requirements, while invoking high-capacity models only for subtasks that necessitate advanced reasoning capabilities. The goal of optimization is to minimize the discrepancy between the model's allocation scheme M and the optimal scheme M^*: min $|M - M^*|$. The optimal scheme M^* is derived through a search strategy that maximizes the usage of **eco-friendly, lower-cost models** while maintaining accuracy. During the optimization process, as the allocation scheme approaches the optimal solution, the API cost (C_{Api}) and energy consumption (E) decrease, indicating a reduction in the system's overall energy footprint, while accuracy (Acc) remains well-maintained.

4 Methodology

The R2-Reasoner framework is centered around a **Model Router**, which consists of two primary modules: a **Task Decomposer** ($\mathcal{M}_{\text{decomp}}$) and a **Subtask Allocator** ($\mathcal{M}_{\text{alloc}}$). The Task Decomposer is engineered to break down complex input tasks T into well-structured and logically ordered subtasks. Following this, the Subtask Allocator strategically routes each subtask t^i to the most suitable model from $\mathcal{M}_{pool}$. The routing is driven by the estimated difficulty of each subtask, aiming to strike an optimal balance between reasoning fidelity and energy efficiency. The design and training of these interconnected components are detailed below.

4.1 Generating Coherent Subtask Sequences via Task Decomposer

The **Task Decomposer** ($\mathcal{M}_{\text{decomp}}$) serves as the first stage of the Model Router, responsible for transforming a complex task T into a sequence of logically connected subtasks $\{t^1, t^2, \ldots, t^k\}$. The quality of this decomposition is crucial: redundant or incoherent breakdowns can cause error propagation, while clear and concise subtasks provide a strong foundation for subsequent allocation.

To supervise training, we construct a decomposition dataset $\mathcal{D}_{\text{decomp}}$ using a rejection sampling strategy. For each task, multiple candidate decompositions are generated and then evaluated along three dimensions: **Conciseness**, assessed by the number of subtasks to avoid both excessive fragmentation and overly coarse splits. **Practicality**, estimated by the total token cost of solving all subtasks with a baseline model. **Coherence**, measuring the logical continuity between adjacent subtasks, with fewer breaks indicating higher quality.

These criteria are linearly combined into a weighted score, where lower values correspond to higher-quality decompositions. A binary correctness signal $C(d) \in \{0, 1\}$ is further incorporated to ensure that the selected decomposition can solve the original task. When possible, only candidates with $C(d) = 1$ are retained, and among them the one with the best score is chosen. This guarantees that $\mathcal{D}_{\text{decomp}}$ contains decompositions that are concise, coherent, and practical while remaining effective for solving the task. The resulting pairs (T, d^*) are then used to fine-tune $\mathcal{M}_{\text{decomp}}$. More details and formulas are provided in the Appendix B.1.

4.2 Strategic Model Assignment for Collaboration via Subtask Allocator

Once $\mathcal{M}_{\text{decomp}}$ produces a subtask sequence, the **Subtask Allocator** ($\mathcal{M}_{\text{alloc}}$) determines how to distribute these subtasks across the heterogeneous model pool $\mathcal{M}_{pool}$. Formally, for each subtask t^i, it selects a model $M_j \in \mathcal{M}_{pool}$, yielding an assignment $M_A : t^i \mapsto M_j$. To enable $\mathcal{M}_{\text{alloc}}$ to learn efficient assignment policies, we construct a high-quality dataset $\mathcal{D}_{\text{alloc}}$ of model allocation schemes. Rather than relying on hand-crafted heuristics, we employ a systematic search procedure over the vast space of possible assignments, seeking schemes that minimize resource consumption while maintaining perfect accuracy. The resulting allocation pairs $(\{t^i\}, M_A^*)$ serve as supervision signals for training $\mathcal{M}_{\text{alloc}}$ to imitate these cost-effective strategies.

However, exhaustive search over all allocations would be prohibitively expensive in both time and cost. We therefore design a **Grouped Search Strategy** to approximate optimal assignments efficiently. The process begins by estimating the difficulty of each subtask t^i using the predictive confidence of a baseline model. We use α-quantile method, which refers to calculating a specific quantile from the token probabilities generated by LLM during inference to denote the confidence that LLM has in answering the given query and then the difficulty of the given query. If the maximum token probability exceeds a threshold τ_{easy}, the subtask is labeled as *easy*; if it falls below τ_{hard}, it is labeled as *hard*; otherwise, it is labeled as *medium*. In parallel, the model pool $\mathcal{M}_{pool}$ is partitioned into three capability groups: small language models (SLMs), medium language models (MLMs), and large language models (LLMs). Each difficulty level is paired with the corresponding capability group (easy→SLM, medium→MLM, hard→LLM).

Based on this categorization, an **initial allocation** $M_A^{(0)}$ is obtained by assigning each subtask to the medium-capacity model within its corresponding group. This serves as the starting point for iterative refinement: if the current allocation already achieves correctness ($Acc = 1$), the allocator attempts to replace some models with cheaper ones to reduce cost; if correctness fails, subtasks are escalated to stronger models within the same group, and only if necessary, across groups. The search is bounded by a maximum number of iterations ($N_{\text{iter\_alloc}} \leq 20$), after which the resulting allocation M_A^* is accepted. The collection of such $(\{t^i\}, M_A^*)$ pairs constitutes $\mathcal{D}_{\text{alloc}}$, which is then used to train $\mathcal{M}_{\text{alloc}}$. Details of the search algorithm is provided in Appendix B.2. This strategy enables $\mathcal{M}_{\text{alloc}}$ to learn fine-grained, capability-aware assignment policies that balance accuracy and efficiency. The detailed formulation of the grouped search procedure is deferred to Appendix B.2.

4.3 Dual-Module Co-training via Iterative Reinforcement Learning

After the initial SFT of $\mathcal{M}_{\text{decomp}}(\theta_{\text{decomp}})$ and $\mathcal{M}_{\text{alloc}}(\theta_{\text{alloc}})$, We employ a staged RL pipeline to further refine their capabilities and promote synergistic collaboration within the Model Router. In each iteration, one module's parameters are updated while the other remains fixed, allowing targeted improvements based on task success feedback, which also circumvents the non-differentiability and discontinuities arising from cascading two LLMs, thereby stabilizing training. The primary reward signal is a binary indicator based on the final correctness of the task T:

$$R_{final}(T, \{t^i\}, M_A) = \begin{cases} 1 & \text{if final answer is correct} \\ 0 & \text{if final answer is incorrect} \end{cases} \quad (1)$$

We adopt Group Relative Policy Optimization (GRPO) as the optimization algorithm for this co-training phase. Training proceeds iteratively for each module:

(1) **Updating $\mathcal{M}_{\text{decomp}}(\theta_{\text{decomp}})$**: The decomposer acts as the policy, generating sequences of subtasks $\{t^i\}$ for an input task T. The fixed allocator $\mathcal{M}_{\text{alloc}}(\bar{\theta}_{\text{alloc}})$ assigns models to these subtasks, and the final outcome is used to compute R_{final}. The reward is propagated back to estimate the advantage $\hat{A}_{i,k}$ for decomposition decisions.

(2) **Updating $\mathcal{M}_{\text{alloc}}(\theta_{\text{alloc}})$**: The allocator acts as the policy, generating assignments $M_A(t^k)$ for each subtask t^k provided by the fixed decomposer $\mathcal{M}_{\text{decomp}}(\bar{\theta}_{\text{decomp}})$. The final correctness again determines R_{final}, which guides the advantage estimates $\hat{A}_{i,k}$ for allocation choices.

This alternating optimization encourages the two modules to progressively adapt to each other, leading to improved overall reasoning performance.

4.4 End-to-End Workflow at Test Time

With the R2-Reasoner's Task Decomposer ($\mathcal{M}_{\text{decomp}}$) and Subtask Allocator ($\mathcal{M}_{\text{alloc}}$) trained through SFT and the iterative RL pipeline, the framework can be deployed for inference. For a user query Q_{user}, the workflow is as follows: (1) **Task Decomposition**: The query Q_{user} is first processed by the fine-tuned Task Decomposer: $\{t^1, \ldots, t^k\} = \mathcal{M}_{\text{decomp}}(Q_{user})$. (2) **Subtask Allocation**: The resulting sequence of subtasks $\{t^1, \ldots, t^k\}$ is then passed to the fine-tuned Subtask Allocator for strategic model assignment: $M_A = \mathcal{M}_{\text{alloc}}(\{t^1, \ldots, t^k\})$, where $M_A(t^i) \in \mathcal{M}_{pool}$ is the model assigned to subtask t^i. (3) **Subtask Execution**: Each subtask t^i is executed sequentially by its assigned model $M_A(t^i)$. The output of subtask t^i can serve as input to the subsequent subtask t^{i+1}. (4) **Result Integration**: The results from the executed subtasks are sequentially integrated to formulate the final answer A_{final}.

To flexibly adapt to scenarios with different cost budgets, achieve a controllable accuracy–cost trade-off, and enhance reasoning robustness, we introduce an optional **P**rocedural **R**eview **M**echanism (PRM). Let $\mathcal{M}_{\text{strong}}$ denote a high-capability model (e.g., a frontier LLM from $\mathcal{M}_{pool}$) and $\mathcal{M}_{\text{thresh}}$ a pre-defined threshold model representing a minimum capability level. For each subtask t^j, let r_j be the output generated by its initially assigned model $M_A(t^j)$. If $M_A(t^j)$ is below the threshold $\mathcal{M}_{\text{thresh}}$, the output will be verified and potentially refined: $r_j^{\text{final}} = \text{PRM\_Verify}(\mathcal{M}_{\text{strong}}, r_j)$ The PRM_Verify function utilizes $\mathcal{M}_{\text{strong}}$ to assess the correctness of r_j. If r_j is deemed incorrect or suboptimal, $\mathcal{M}_{\text{strong}}$ provides a corrected or refined response r_j'; otherwise, $r_j^{\text{final}} = r_j$. This r_j^{final} is then used for all subsequent reasoning steps. This mechanism allows

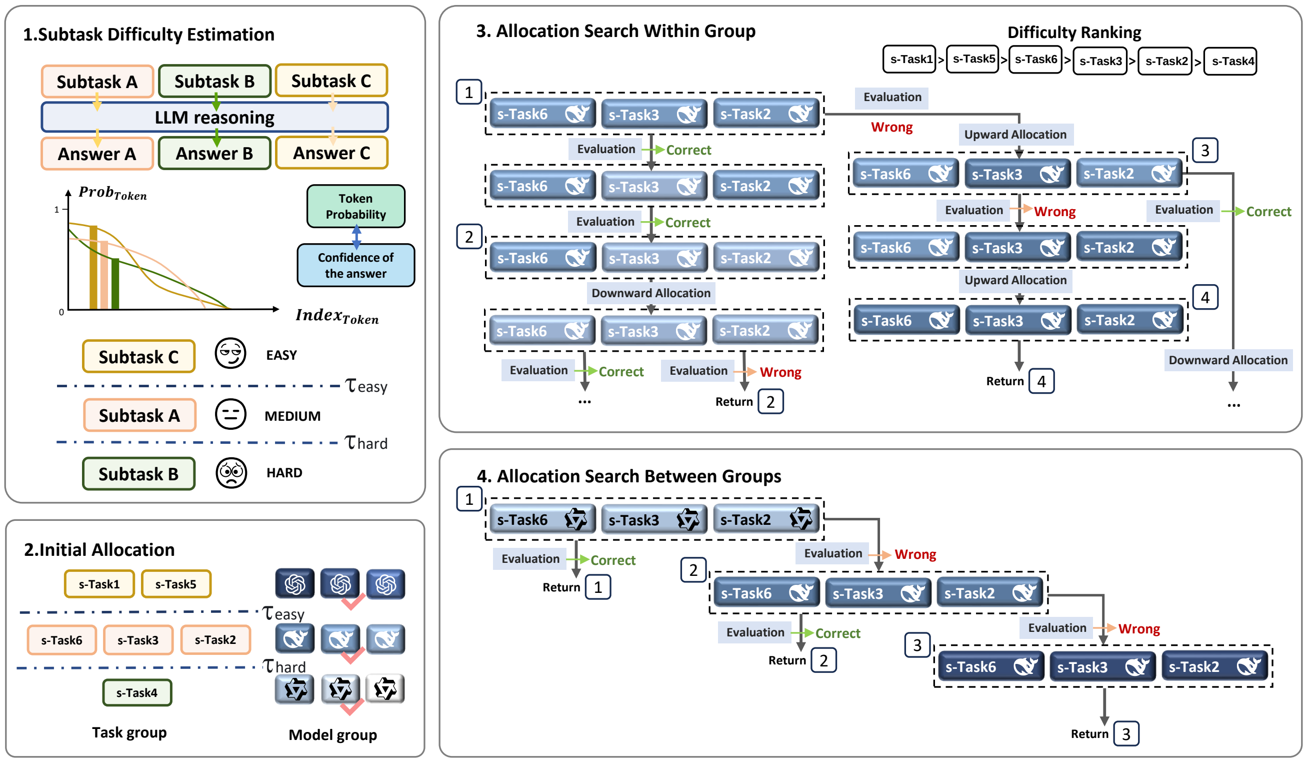


Figure 2: Overview of Our Grouped Search Strategy for Optimal Allocation Scheme

targeted quality control, preserving accuracy while maintaining the cost-efficiency of allocation.

5 Experiments

5.1 Experimental Setup

Benchmarks We choose six widely-used open-source benchmarks:

(1) **P3** [25] for program synthesis. P3 defines each puzzle by a python program f and checks if the candidate input could make f return `True`. It places emphasis on the ability involved during coding process such as syntax correctness and algorithmic reasoning.

(2) **SCAN** [16] for language-driven navigation. It consists of navigation commands with corresponding action sequences. By instructing machines to convert natural-language commands into a sequence of actions and comparing the generated sample with label, it focus on assessing the ability of logical traversal, backward reasoning and anomaly detection.

(3) **MATH** [14] and **CHAMP** [19]for solving challenging math problems. MATH consists of 12,500 challenging mathematics problems, while CHAMP contains 270 diverse high school competition-level math problems. They mainly involve LLM's conducting computation and memorizing mathematical knowledge. Solving math problems has been universally acknowledged as a crucial aspect to measure LLM's reasoning ability.

(4) **CSQA** [28] and **MuSiQue** [30] for commonsense reasoning. These 2 benchmarks require a broader commonsense knowledge base for LLM. Considering the knowledge base varies as the scale of LLM varies, they are suitable for testing if different LLMs in our framework could collaborate and compose an integrated knowledge base in commonsense scenario.

For each benchmark, we manually annotate a small set of samples for in-context learning in task decomposition and select another 200 tasks as the test set.

Baselines Considering the scenario of collaborative reasoning, we establish six baselines. **(1) CoT [32]:** CoT (Chain of Thought) asks a single LLM to solve a task by decomposing the original task into a sequence of sub-tasks and answering these sub-tasks sequentially. **(2) ToT [34]:** ToT (Tree of Thoughts), based on the framework of CoT, prompts multiple answers (N = 2) for each sub-task, and retain the best answer by utilizing a scoring method. It also only deploys one certian LLM. **(3) DataShunt [4]:** Datashunt dynamically selects between a SLM and a LLM to finish the task. The method first evaluates the difficulty of the given task, and allocate the task to either SLM or LLM to solve utilizing the CoT method. **(4) AutoMix [2]:** AutoMix consists of a few-shot self-verification mechanism conducted by SLM to evaluate the confidence toward an answer from SLM and a router that strategically routes queries to LLM based on the confidence. **(5) DoT [26]:** DoT decomposes a task into subtasks, builds a dependency graph, and allocates subtasks to SLMs or LLMs using a Plug-and-Play Adapter on SLMs. This framework enables efficient edge-cloud collaborative reasoning. **(6) Router-R1 [35]:** Router-R1 chooses an LM as the router itself, interweaving thinking process by the router with routing process by the routed models, and integrates every response into the context.

Selection and Deployment of LLMs In R2-Reasoner method, for candidate LLMs to solve different subtasks, We select Qwen2.5-0.5B-instruct, Qwen2.5-1.5B-instruct, Qwen2.5-3B-instruct, Qwen2.5-7B-instruct, Qwen2.5-14B-instruct, Qwen2.5-32B-instruct, Qwen2.5-72B-instruct [24], DeepSeek-V3 [8], gpt-4o [21] as the LLM pool.

Model	Program Synthesis			Language-Driven Navigation			Math Problem Solving						Commonsense Reasoning					
	P3			SCAN			MATH			CHAMP			CSQA			MuSiQue		
	Acc(%)	C_{API}(¢)	*E*(J)	*Acc*(%)	C_{API}(¢)	*E*(J)	*Acc*(%)	C_{API}(¢)	*E*(J)	*Acc*(%)	C_{API}(¢)	*E*(J)	*Acc*(%)	C_{API}(¢)	*E*(J)	*Acc*(%)	C_{API}(¢)	*E*(J)
CoT (GPT-4o)	**42**	4.45	3177.99	68	2.75	4855.03	51.5	5.34	3836.04	55.5	4.45	3856.04	80	3.60	2986.33	57	0.85	619.64
ToT (GPT-4o)	38	14.55	11977.75	52	9.82	9045.48	63	9.97	5730.99	57	11.65	11943.07	82	20.50	18736.34	**59**	2.45	2112.15
CoT (Llama 3-8B)	5.5	-	256.13	17	-	191.20	10	-	328.94	19	-	78.66	70	-	466.25	38	-	48.55
ToT (Llama 3-8B)	5.5	-	479.11	13	-	339.41	29.5	-	497.09	25	-	651.85	68.5	-	1667.65	31	-	144.97
DataShunt	14	2.45	4073.25	23.5	1.72	1477.77	16	1.66	826.43	34	2.98	2936.83	73	1.28	160.98	47	0.46	372.89
AutoMix	14	0.04	78.73	43	0.12	332.16	44	0.03	48.08	44	0.34	145.93	66	0.001	13.20	51	0.0074	16.34
DoT	41	1.58	2437.95	63	1.20	1152.49	59	1.02	493.76	58	0.84	936.88	82	0.49	91.37	50	0.13	96.28
Router-R1	7	0.14	384.48	2	0.15	161.95	58	0.62	214.31	47	9.78	287.45	54	0.12	76.29	38	0.12	127.18
R2-Reasoner	38	1.16	1264.81	**75**	0.64	657.98	**76.5**	0.08	123.11	**59.5**	0.28	308.34	**83.5**	0.042	75.62	56.5	0.029	41.81
Improvement	↓9.52%	↓73.93%	↓60.20%	↑10.29%	↓76.73%	↓86.45%	↑21.43%	↓99.18	↓97.85%	↑2.59%	↓66.67%	↓67.09%	↑1.83%	↓91.43%	↓17.24%	↓4.24%	↓98.82%	↓98.02%

Table 1: Performance of R2-Reasoner and baselines on 6 benchmarks. C_{API} is averaged expense for each task, and E is the average energy consumption. API cost is measured in US dollar cents (¢), while energy consumption is measured in units of Joules(J). "-" appears in experiments where reasoning is conducted solely using local deployed SLMs without invoking the cloud-based LLMs or data is unavailable. The highest reasoning accuracy is highlighted in bold. The baseline with the highest Acc is underlined and used to compute the "Improvement" in the last row.

The ability of these LLMs increases following the order above. Among these models, Qwen2.5-0.5B-instruct, Qwen2.5-1.5B-instruct, Qwen2.5-3B-instruct, Qwen2.5-7B-instruct are fee free for being locally deployed, while the other cloud-based LLMs charges, and the price of the these LLMs also increases following the order above. For SFT and RL training on the task decomposer and subtask allocator, we select Qwen2.5-7B-instruct as the base model. For CoT and ToT baselines, we respectively deploy gpt-4o and LLaMA3-8B [20] in our experiments. For DataShunt, AutoMix and DoT baselines, we select gpt-4o as the LLM and LLaMA3-8B as the SLM. For Router-R1 baseline, we adopt the same LLM pool as in our R2-Reasoner method.

Evaluation For evaluation, we set three metrics: *Acc*, C_{API} and E, which represents our three main concerns in LLM reasoning. *Acc* measures the accuracy of our framework and the baselines on four benchmarks.C_{API} measures the average API cost for a single task, calculated in US dollar cents. E measures the average energy consumption for each task, calculated in units of Joules. Specifically, E is obtained by multiplying the total number of FLOPs with the energy required per FLOP operation. The total number of FLOPs is computed as the sum of FLOPs consumed by each model. For a given model, the FLOPs can be approximated as Total FLOPs $\approx 2N_{\text{params}} \cdot T$, where N_{params} denotes the scale of model parameters (For MoE architectures such as DeepSeek-V3, this corresponds to the dynamically activated parameter scale. Given the absence of officially details regarding the parameter scale of GPT-4o, we adopt the estimate of 200B parameters reported in MEDEC [1] as a reasonable approximation.), and T represents the sum of input and output tokens [15]. In addition, we assume that all models are deployed on NVIDIA A100 SXM GPUs. Under this assumption, the energy consumed per FLOP can be approximated by dividing the GPU's thermal design power (TDP) by its peak FLOPS throughput. These hardware specifications can be obtained from the GPU's technical documentation [7].

5.2 Experimental Hyperparameters

We set sophisticatedly designed hyperparameters during dataset construction to better capture the ideal task decompostion and subtask allocation schemes.

During constructing the dataset for the Task Decomposer, we computed a weighted average over the three dimensions of task decomposition to obtain a score according to an equation 3, which involves three hyperparameters: w_c, w_p, and w_d. These three hyperparameters serve as weights for: (1) the total number of subtasks, (2) the total number of tokens used during inference, and (3) the coherence score, respectively. Empirically, these three components exhibit significantly different value ranges across a wide range of tasks. Specifically, our analysis shows that their average values are approximately 5.87 (number of subtasks), 676.59 (token count), and 0.1541 (coherence score). To ensure the comparability of these components during weighted aggregation, our hyperparameter selection strategy is based on normalizing them to a similar scale. Accordingly, we set $w_c = 100$, $w_p = 1$, and $w_d = 1000$, which balances their contributions in the combined scoring function.

To obtain the **Practicality** and the binary correctness signal $C(d) \in \{0, 1\}$ in the scoring process, we adopt a locally deployed LLaMA3-8B as the baseline evaluation model $\mathcal{M}_{\text{eval}}$, which could offer provides fast, consistent, and relatively rigorous feedback for our task-decomposition framework.

During constructing the dataset for the Subtask Allocator, we initially categorize subtasks to groups of different difficulty level by an α-quantile method. For the quantile value selection, lower α-quantile value tends to overestimate the difficulty of all subtasks, shifting overall allocation toward the hard group, which results in selecting larger models. In contrast, higher α-quantile value tends to underestimate subtask difficulty, shifting the allocation toward the easy group and thus selecting smaller models. After conducting pilot studies on multiple original tasks, we set α = 0.3 as our quantile value to ensure a proper initial allocation.

To obtain the appropriate thresholds τ_{easy} and τ_{hard}, we originally set the τ_{easy} as 0.66 and τ_{hard} as 0.33, then we conduct a pilot study of dozens of original tasks on several benchmarks to check the ratio of across-group refinement. If the ratio is too high, which means our current threshold setting could not precisely group the subtasks based on their difficulty level, we examine the detailed token probability value and modify the set thresholds. Finally, the τ_{easy} is set as 0.8, and τ_{hard} is set as 0.52. Details regarding hardware resources and hyperparameter configurations used in training can be found in Appendix C.

5.3 Main Results

The comparison between our framework and the baselines in six benchmarks are shown in Table 1. We have highlighted in bold the highest accuracy results among the eight baseline experiments on each benchmark, while the associated API costs and energy consumption are underlined. We compute the relative improvement of our results compared to the baseline with the highest accuracy. The experimental results demonstrate that our framework significantly reduces the API cost and energy consumption while retaining a comparable reasoning accuracy. The relative changes in accuracy compared to the highest baseline accuracy are: -9.52%, +10.29%, +21.43%, +2.59%, +1.83%, -4.24%. Even for P3, the decline in accuracy is still acceptable. The boost in accuracy on benchmark like MATH and SCAN validate the potential of our work in enhancing reasoning ability. Meanwhile, our framework achieves a tremendous reduction in API cost and energy consumption compared to the baseline with the highest accuracy, reaching averagely a decline of 84.46% in API cost and 71.14% in energy consumption.

The accuracy of our framework on benchmarks like MATH and SCAN surpassing the CoT and ToT method shows the potential disadvantage of excessive reasoning. It usually happens in reasoning process conducted by LLMs of large scale, often deviates from the correct and suitable answer for a subtask because it automatically proceed with reflective or divergent thinking. We design several precise and exquisite prompts attempting to avoid the phenomenon.

Overall, the results demonstrate that R2-Reasoner achieves energy-efficient LLM reasoning, which shows great potential of constructing green and environment-friendly reasoning systems.

5.4 Ablation Study

To rigorously evaluate the contribution and of each stage, we report the performance metrics (Acc and C_{API}) of the Task Decomposer after each training stage, as summarized in Table 2. The table compares the base model, the model after supervised fine-tuning (SFT), and the final model after SFT combined with RL.

As observed, the SFT stage improves performance across all benchmarks compared to the base model. Importantly, the addition of the RL stage consistently further enhances both accuracy and cost efficiency on every task. For instance, accuracy increases by 5–8% on most benchmarks, while C_{API} is reduced or maintained at a comparable level. This consistent improvement demonstrates that the RL stage not only reliably enhances task performance but also stabilizes the routing decisions across tasks. Overall, these results strongly validate the effectiveness and robustness of our RL-based multi-stage training process.

5.5 Generalization to Newly Unseen LLMs

To evaluate the generalization capability of the proposed R2-Reasoner, we conduct an additional experiment in which several models are replaced with alternatives of comparable capacity, without retraining the framework. Specifically, Qwen2.5-7B is replaced with GLM-4-9B-Chat [11], and DeepSeek-V3 with Kimi-K2-Instruct [29]. The results are summarized in Table 3.

As observed, the performance of our framework remains largely stable on SCAN, MATH and CSQA. Accuracy decreases by 11.8% on P3, 13% on CHAMP and 9% on MuSiQue, which can be attributed to differences in the reasoning capabilities of the replaced models. Meanwhile, C_{API} increases due to the higher API costs associated with the new models. Overall, these results indicate that the framework exhibits robust generalization to previously unseen LLMs. Importantly, the R2-Reasoner does not rely on any particular model; as long as the relative ordering of model capabilities is preserved, the router can maintain stable and reliable performance across different model pools.

5.6 Trade-off Between Reasoning Cost and Accuracy

Our framework supports a flexible trade-off between accuracy and cost, enabling adaptation to different budget scenarios. By adjusting the routing threshold within R2-Reasoner, we can dynamically balance performance and expenditure. As shown in Figure 3, when compared against DoT and DataShunt baselines on MATH and SCAN benchmarks, our method establishes a new Pareto frontier. The results clearly show that R2-Reasoner consistently achieves significantly higher accuracy for a given cost budget, or conversely, reaches a target accuracy at a substantially lower cost than both competing methods.

This remarkable efficiency is quantitatively demonstrated across both datasets. On the MATH benchmark, R2-Reasoner achieves over 70% accuracy for less than 0.08 cents, while the stronger baseline, DoT, requires approximately 6 cents to reach similar performance—a cost reduction of more than 75×. This advantage holds on the SCAN dataset, where our method reaches 60% accuracy for about 0.4 cents, a task that costs the DoT baseline approximately 5 cents. These results empirically prove that our routing mechanism enables highly effective and budget-aware reasoning, offering practical adaptability for diverse real-world deployment scenarios with varying budget constraints and providing a sophisticated approach for building up sustainable reasoning system.

5.7 Inference Time Comparison

We conducted additional experiments under a consistent network environment to evaluate the end-to-end reasoning latency of our framework against several baseline methods. Each experiment was performed independently under identical conditions. All API calls were made sequentially in a single thread to eliminate concurrency-related interference and ensure that external factors did not distort the latency measurements. The reported results represent the average latency across all tasks in the benchmark, computed after completing full inference runs for every task. In each bar plot, the bar with the darkest color corresponds to our proposed method. The summarized results are presented in Figure 4.

The inference latency results demonstrate significant differences among the evaluated routing methods across the four benchmark tasks. Notably, R2-Reasoner consistently achieves the lowest or near-lowest latency in most cases. For instance, on P3, R2-Reasoner completes inference in 14.27 seconds, substantially faster than CoT

Stages	P3		SCAN		MATH		CHAMP		CSQA		MuSiQue	
	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$
base	23.5	0.314	14	0.066	67	0.150	50	0.494	70.5	0.147	43	0.0226
w/ SFT	33	2.027	68	0.577	75.5	0.079	58	0.370	82	0.056	51.5	0.0301
w/ SFT+RL	38	1.160	75	0.636	76.5	0.080	59.5	0.280	83.5	0.042	56.5	0.0287

Table 2: Performance (Acc and C_{API}) after each training stage.

Models	P3		SCAN		MATH		CHAMP		CSQA		MuSiQue	
	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$	$Acc(\%)$	$C_{API}(¢)$
Initial Pool	38	1.160	75	0.636	76.5	0.080	59.5	0.280	83.5	0.042	56.5	0.0287
Modified Pool	33.5	1.278	75	0.656	75	0.105	51.5	0.310	81.5	0.060	51.5	0.0438

Table 3: Experimental results of generalization capability of R2-Reasoner to new LLMs

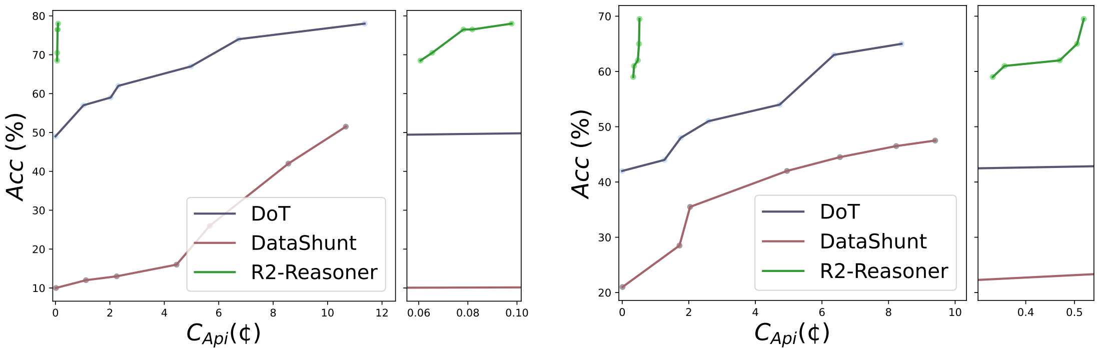
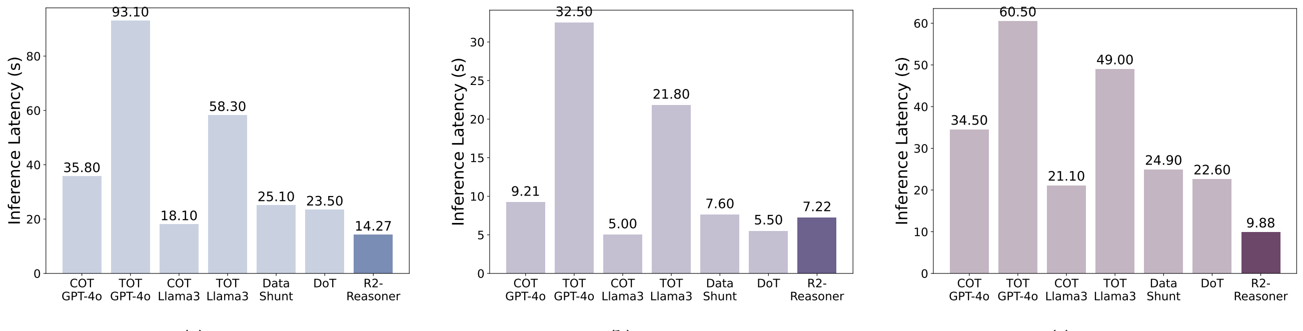


Figure 3: Acc-Cost trade-off curves on MATH (left) and SCAN (right). A magnified inset is provided to the right of the original sub-figure to more precisely illustrate the Pareto frontier of our method.



(a) P3 **(b) SCAN** **(c) MATH**

Figure 4: Inference latency comparison of different methods across three benchmarks.

and ToT configurations with both GPT-4o and LLaMA 3-8B models, which require between 18.1 and 93.1 seconds. Similar trends are observed on MATH and CSQA, where R2-Reasoner reduces inference time by more than 50% compared to the heaviest baselines (ToT).

On SCAN, R2-Reasoner incurs a slightly higher latency than CoT (LLaMA 3-8B), but it still remains considerably faster than the majority of other methods, including all GPT-4o-based baselines. This performance advantage can be attributed to the framework's adaptive routing strategy, which prioritizes lightweight models for simpler instances and selectively invokes higher-capacity models only when necessary. As a result, R2-Reasoner achieves both time efficiency and cost efficiency, without compromising task performance. Overall, these results highlight the framework's capability to perform fast and scalable reasoning across diverse benchmarks, demonstrating clear practical advantages over existing LLM routing methods.

6 Conclusion

In this work, we present R2-Reasoner, a novel framework leveraging a reinforced Model Router to enable energy-efficient scaling of large language model reasoning. Enabled by an iterative training pipeline, our framework fosters adaptive, sustainable model collaboration by strategically allocating subtasks to the most resource-efficient models, thereby significantly minimizing computational waste. Looking forward, R2-Reasoner paves the way for Green AI in complex reasoning systems and energy-aware computing ecosystems.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (grant no. 62472241 and 23IAA02114), in part by the joint project of Infinigence AI & Tsinghua University.

References

[1] Asma Ben Abacha, Wen-wai Yim, Yujuan Fu, Zhaoyi Sun, Meliha Yetisgen-Yildiz, Fei Xia, and Thomas Lin. 2025. Medec: A benchmark for medical error detection and correction in clinical notes. In *Findings of the Association for Computational Linguistics: ACL 2025*. 22539–22550.

[2] Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. 2025. AutoMix: Automatically Mixing Language Models. arXiv:2310.12963 [cs.CL] https://arxiv.org/abs/2310.12963

[3] Fenglong Cai, Dong Yuan, Zhe Yang, and Lizhen Cui. 2024. Edge-llm: A collaborative framework for large language model serving in edge computing. In *2024 IEEE International Conference on Web Services (ICWS)*. IEEE, 799–809.

[4] Dong Chen, Yueting Zhuang, Shuo Zhang, Jinfeng Liu, Su Dong, and Siliang Tang. 2024. Data Shunt: Collaboration of Small and Large Models for Lower Costs and Better Performance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 11249–11257.

[5] Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Step-level Value Preference Optimization for Mathematical Reasoning. *arXiv preprint arXiv:2406.10858* (2024).

[6] Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. 2024. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems* 37 (2024), 66305–66328.

[7] NVIDIA Corporation. 2025. NVIDIA A100 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/a100/. Accessed: 2025-11-25.

[8] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[9] Jasper Dekoninck, Maximilian Baader, and Martin Vechev. 2024. A unified approach to routing and cascading for llms. *arXiv preprint arXiv:2410.10347* (2024).

[10] Tao Feng, Yanzhen Shen, and Jiaxuan You. 2024. Graphrouter: A graph-based router for llm selections. *arXiv preprint arXiv:2410.03834* (2024).

[11] Team GLM, :, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Jingyu Sun, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. arXiv:2406.12793 [cs.CL] https://arxiv.org/abs/2406.12793

[12] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. 2025. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature* 645, 8081 (2025), 633–638.

[13] Zixu Hao, Huiqiang Jiang, Shiqi Jiang, Ju Ren, and Ting Cao. 2024. Hybrid slm and llm for edge-cloud collaborative inference. In *Proceedings of the Workshop on Edge and Mobile Foundation Models*. 36–41.

[14] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874* (2021).

[15] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361* (2020).

[16] Brenden Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International conference on machine learning*. PMLR, 2873–2882.

[17] En Li, Liekang Zeng, Zhi Zhou, and Xu Chen. 2019. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications* 19, 1 (2019), 447–457.

[18] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's Verify Step by Step. arXiv:2305.20050 [cs.LG] https://arxiv.org/abs/2305.20050

[19] Yujun Mao, Yoon Kim, and Yilun Zhou. 2024. CHAMP: A Competition-level Dataset for Fine-Grained Analyses of LLMs' Mathematical Reasoning Capabilities. *arXiv preprint arXiv:2401.06961* (2024).

[20] Meta. 2024. Introducing Meta Llama 3: The most capable openly available LLM to date. https://ai.meta.com/blog/meta-llama-3/

[21] OpenAI. 2024. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/.

[22] OpenAI. 2024. Introducing OpenAI o1. https://openai.com/o1/

[23] OpenAI. 2025. Introducing GPT-5. https://openai.com/index/introducing-gpt-5/.

[24] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[25] Tal Schuster, Ashwin Kalyan, Oleksandr Polozov, and Adam Tauman Kalai. 2021. Programming puzzles. *arXiv preprint arXiv:2106.05784* (2021).

[26] Chenyang Shao, Xinyuan Hu, Yutang Lin, and Fengli Xu. 2025. Division-of-thoughts: Harnessing hybrid language model synergy for efficient on-device agents. In *Proceedings of the ACM on Web Conference 2025*. 1822–1833.

[27] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. arXiv:2408.03314 [cs.LG] https://arxiv.org/abs/2408.03314

[28] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2018. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937* (2018).

[29] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. 2025. Kimi K2: Open Agentic Intelligence. arXiv:2507.20534 [cs.LG] https://arxiv.org/abs/2507.20534

[30] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. arXiv:2108.00573 [cs.CL] https://arxiv.org/abs/2108.00573
[31] Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Y Wu, and Zhifang Sui. 2023. Math-shepherd: A label-free step-by-step verifier for llms in mathematical reasoning. *arXiv preprint arXiv:2312.08935* (2023).
[32] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35 (2022), 24824–24837.
[33] Noam Wies, Yoav Levine, and Amnon Shashua. 2023. Sub-Task Decomposition Enables Learning in Sequence to Sequence Tasks. arXiv:2204.02892 [cs.CL] https://arxiv.org/abs/2204.02892
[34] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601* (2023).
[35] Haozhen Zhang, Tao Feng, and Jiaxuan You. 2025. Router-R1: Teaching LLMs Multi-Round Routing and Aggregation via Reinforcement Learning. *arXiv preprint arXiv:2506.09033* (2025).
[36] Mingjin Zhang, Jiannong Cao, Xiaoming Shen, and Zeyang Cui. 2024. Edge-Shard: Efficient LLM Inference via Collaborative Edge Computing. *arXiv preprint arXiv:2405.14371* (2024).
[37] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117* (2023).
[38] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625* (2022).
[39] Wang Zhu, Jesse Thomason, and Robin Jia. 2023. Chain-of-questions training with latent answers for robust multistep question answering. *arXiv preprint arXiv:2305.14901* (2023).

A Supplementary Experiment Results

A.1 Performance Improvement of Decomposer

To evaluate how the two stages of SFT and RL training have improved the task decomposer, we test 100 tasks on four benchmark (one from each category) and report results using two global metrics: C_d and the comprehensive *Score* (defined in Section 4.1). The C_d is calculated as the accuracy of the final answer obtained by allocating all subtasks generated from the current checkpoint to Llama3-8B, while the *Score* is computed following Equation 3. The comparison between the base model and our trained checkpoints is shown in Table 4. On average, SFT and RL jointly yield a 27% increase in C_d and a 6% reduction in *Score*. Across all benchmarks, SFT provides consistent improvements, while RL exhibits mild instability but still contributes overall gains. We attribute this instability to potential insufficiencies in the reward function design.

Model	P3		SCAN		MATH		CSQA	
	C_d	*Score*	C_d	*Score*	C_d	*Score*	C_d	*Score*
base	0.06	2200.51	0.38	1600.19	0.28	1311.71	0.69	1171.53
w/ SFT	0.10	1848.40	0.46	1557.43	0.31	1265.60	0.75	1161.46
w/ SFT+RL	0.10	1788.41	0.45	1508.50	0.34	1234.63	0.72	1201.73

Table 4: Performance improvement achieved of the Task Decomposer after multi-stage training.

Benchmark	Conciseness	Practicality	Coherence
SCAN	3.00⟶2.9263	2197.04⟶2208.89	0.1459⟶0.1367
MATH	7.54⟶4.36	848.03⟶939.45	0.0364⟶0.0116

Table 5: Evaluation of decomposition quality before and after the multi-stage training.

Beyond these global metrics, we further analyze decomposition quality on three finer-grained dimensions: *Conciseness*, *Practicality*, and *Coherence*. These dimensions are operationalized as follows: *Conciseness*: measured by the number of subtasks generated. *Practicality*: measured by the token cost required for reasoning. *Coherence*: measured by the proportion of logically incoherent subtask pairs (as described in Section 4.1).

Fewer subtasks are generally preferred, as they directly reduce API cost and latency. An excessive number of subtasks may cause redundancy and confusion. Token consumption is ideally lower, since concise answers are desirable, though moderately longer reasoning chains may yield more thorough inference. For coherence, a smaller value is better, indicating stronger logical consistency among subtasks. As shown in Table 5, our multi-stage training significantly improves decomposition quality across these dimensions, further validating the effectiveness of our approach.

A.2 Performance Improvement of Allocator

To measure how the 2 stages of SFT and RL training have improved the ability of subtask allocator, we test 100 tasks on each benchmark and set 2 metrics for evaluation: *Acc* and *MAE*. The *Acc* metric measures how many allocation samples are correct according to the labels in our allocation dataset. The *MAE* metrics is based on the LLM pool listed below: Qwen2.5-0.5B-instruct, Qwen2.5-1.5B-instruct, Qwen2.5-3B-instruct, Qwen2.5-7B-instruct, Qwen2.5-14B-instruct, Qwen2.5-32B-instruct, Qwen2.5-72B-instruct, DeepSeek-V3, gpt-4o. Starting from Qwen2.5-0.5B-instruct as model 0, we sequentially assign model indices from 0 to 8, making the size of the number align with the scale of the LLMs. We calculate the MAE between the prediction LLM ID and the label LLM ID. The MAE metric indicates the distance on the LLM map, providing a supplementary sign showing that even if the prediction is wrong, how close it is to the labelled correct answer. The comparison of the base model and our training checkpoint are shown in Table 6. In overall the SFT and RL method have achieved on average 121.29% increase on accuracy and 24.08% decrease on MAE. On all benchmarks, the SFT method shows significant improvement in both metrics. RL method is also slightly unstable but still further achieve an overall improvement on the base of SFT method. The insuffiency of RL method's effect may be because some inevitable reward hacking during the RL process.

Model	P3		SCAN		MATH		CSQA	
	Acc	*MAE*	*Acc*	*MAE*	*Acc*	*MAE*	*Acc*	*MAE*
base	0.0923	3.0763	0.1138	3.1041	0.1016	2.5355	0.1773	2.5638
w/ SFT	0.2197	2.7762	0.2067	1.9107	0.2362	1.8685	0.3274	1.9419
w/ SFT+RL	0.2187	2.7862	0.2606	1.9361	0.2410	1.8603	0.3227	1.9834

Table 6: Performance improvement achieved of the Subtask Allocator after multi-stage training.

A.3 RL Reduces Dependence on SFT Data

To further examine the effectiveness of the RL stage, we conducted an additional experiment on the MATH dataset by deliberately reducing the amount of supervised fine-tuning (SFT) data. Specifically, the SFT training set was reduced by 50%, while the number of RL training epochs was doubled.

Under this setting, the model's accuracy initially dropped by 23% immediately after SFT due to the reduced amount of annotated data. However, after applying RL, not only was this performance degradation fully recovered, but the reasoning accuracy was further improved by an additional 1.5% compared to the original full-data SFT baseline. This result highlights a key advantage of our RL process: beyond improving reasoning ability, it **substantially reduces dependence on large quantities of annotated SFT data.** In practice, this suggests that RL can serve as a scalable alternative when labeled resources are limited, making our approach more data-efficient and broadly applicable.

B Further Supplements to Methods & Formulas

B.1 Detailed Formulation of Decomposer

Here, we provide a detailed formulation of the dataset construction process for the Task Decomposer (4.1). The Task Decomposer, denoted as $\mathcal{M}_{\text{decomp}}$, is responsible for transforming a complex input task T into a sequence of clearly defined and logically connected subtasks: $T \xrightarrow{\mathcal{M}_{\text{decomp}}} \{t^1, t^2, \ldots, t^k\}$, where k is the number of subtasks. To systematically evaluate and select high-quality decompositions, we define three complementary metrics. **Conciseness** measures the number of subtasks k, balancing between over-fragmentation and overly coarse decomposition. **Practicality** estimates the computational cost by summing the token usage of all subtasks under a baseline evaluation model $\mathcal{M}_{\text{eval}}$:

$$\text{Practicality}(d) = \sum_{i=1}^{k} \text{Tokens}(t^i, \mathcal{M}_{\text{eval}}). \tag{2}$$

Coherence evaluates the logical flow by counting adjacent subtask pairs that lack meaningful connection, denoted as $\text{Coe}_{\text{pair}}(d)$. It is calculated as $A/(N-1)$, among which A denotes the number of pairs of adjacent subtasks in the sequence of multiple subtasks that are independent from each other, and N denotes the number of all the subtasks. Lower values indicate better continuity.

These metrics are combined into an overall score for a candidate decomposition $d = \{t^i\}_{i=1}^{k}$:

$$\text{Score}(d) = w_c \cdot k + w_p \cdot \sum_{i=1}^{k} \text{Tokens}(t^i, \mathcal{M}_{\text{eval}}) + w_d \cdot \text{Coe}_{\text{pair}}(d), \tag{3}$$

where $w_c, w_p, w_d > 0$ are weighting coefficients. Lower scores correspond to higher-quality decompositions.

Additionally, a binary correctness signal $C(d) \in \{0, 1\}$ is determined by attempting to solve the original task using decomposition d with the evaluation model $\mathcal{M}_{\text{eval}}$. For each task T, we generate a set of candidate decompositions $\mathcal{S}_T = \{d_1, d_2, \ldots, d_m\}$ and select the decomposition d^* that minimizes the score while satisfying correctness if possible:

$$d^* = \begin{cases} \arg\min_{d \in \mathcal{S}_T, C(d)=1} \text{Score}(d) & \text{if any } C(d) = 1, \\ \arg\min_{d \in \mathcal{S}_T} \text{Score}(d) & \text{otherwise.} \end{cases} \tag{4}$$

The collection of all (T, d^*) pairs forms the decomposition dataset $\mathcal{D}_{\text{decomp}}$.

Finally, the Task Decomposer is fine-tuned on $\mathcal{D}_{\text{decomp}}$ using a standard cross-entropy loss:

$$\mathcal{L}_{\text{decomp}} = -\sum_{(T,d^*) \in \mathcal{D}_{\text{decomp}}} \sum_i \log P_{\theta_{\text{decomp}}}(d_i^* \mid T), \tag{5}$$

where d_i^* denotes the i-th subtask in the target decomposition. This training ensures that $\mathcal{M}_{\text{decomp}}$ consistently generates concise, practical, and coherent subtask sequences suitable for efficient reasoning.

B.2 Grouped Search Strategy for Allocator Training

Here, we provide the full details of the grouped search algorithm used to construct the allocation dataset $\mathcal{D}_{\text{alloc}}$ (4.2).

Formal Problem. Given subtasks $\{t^i\}$ from $\mathcal{M}_{\text{decomp}}$ and a model pool $\mathcal{M}_{pool}$, the objective is to find an allocation scheme M_A^* that minimizes resource consumption while ensuring correctness:

$$M_A^* = \arg\min_{M_A} \mathbb{E}[C_{Api}(M_A) + C_{Time}(M_A)] \quad \text{s.t.} \quad Acc(M_A) = 1. \tag{6}$$

Granularity Expansion. Each subtask t^i is labeled with a difficulty level based on α-quantile token probabilities:

$$G(t^i) = \begin{cases} G_E & p(t^i) \geq \tau_{diff1}, \\ G_M & \tau_{diff2} < p(t^i) < \tau_{diff1}, \\ G_H & p(t^i) \leq \tau_{diff2}. \end{cases} \tag{7}$$

Simultaneously, models are grouped by capability:

$$\mathcal{M}_{pool} = \mathbb{G}_{\mathcal{M}}^{SLM} \cup \mathbb{G}_{\mathcal{M}}^{MLM} \cup \mathbb{G}_{\mathcal{M}}^{LLM}. \tag{8}$$

An initial allocation $M_{A,0}$ maps each subtask to the medium-capacity model within the corresponding group.

Within-Group Refinement. For each iteration j, the allocation $M_{A,j}$ is updated as:

$$M_{A,j+1}(t^i) = \begin{cases} \text{smaller}(\mathbb{G}_{\mathcal{M}}^X) & \text{if } Acc(M_{A,j}) = 1, \\ \text{larger}(\mathbb{G}_{\mathcal{M}}^X) & \text{if } Acc(M_{A,j}) = 0, \end{cases} \tag{9}$$

where $X = G(t^i)$.

Cross-Group Adjustment. If correctness cannot be achieved with within-group adjustments, inter-group changes are made:

$$M_{A,j+1}(t^i) \in \mathbb{G}_{\mathcal{M}}^Y, \quad Y \neq X, \tag{10}$$

subject to available model capacities.

Termination. The algorithm halts after at most $N_{\text{iter\_alloc}} \leq 20$ iterations or when $Acc(M_{A,j}) = 1$ with minimal resource usage. The resulting allocations $\{(\{t^i\}, M_A^*)\}$ populate $\mathcal{D}_{\text{alloc}}$.

Training Objective. The allocator $\mathcal{M}_{\text{alloc}}$ is trained on $\mathcal{D}_{\text{alloc}}$ via supervised fine-tuning. The loss function is defined as:

$$\mathcal{L}_{\text{alloc}} = -\sum_{(\{t^i\}, M_A^*) \in \mathcal{D}_{\text{alloc}}} \sum_i \log P_{\theta_{\text{alloc}}}(M_A^*(t^i) \mid t^i). \tag{11}$$

C Experiment Details

The hardware environment used for our experiments and the specific training hyperparameters are summarized in Table 7.

Module	Element	Detail
System	OS	Ubuntu 20.04.6 LTS
	CUDA	12.4
	Python	3.12.9
	Pytorch	2.6.0
	trl	0.17.0
	accelerate	1.6.0
	peft	0.15.1
	flash_attn	2.7.4.post1
	Device	2*NVIDIA A100 80G
Workflow	API	Siliconflow & Microsoft Azure
SFT	Mode	Lora
	Batch size	4, 8
	Number of epochs	2, 3
	Max token length	2048
	Lora rank	32, 64
	Optimizer	AdamW
	Learning rate	0.00002, 0.00003
RL Training	Algorithm	GRPO
	Number of Generation	4
	Batch size	1
	Global step	1024
	Max token length	2048
	Optimizer	AdamW
	Learning rate	0.0001, 0.00015

Table 7: Detailed Experimental Settings