PLAN-TUNING: Post-Training Language Models to Learn Step-by-Step Planning for Complex Problem Solving

Mihir Parmar[1,2] **Palash Goyal**[1] **Xin Liu**[1] **Yiwen Song**[1] **Mingyang Ling**[1]
Chitta Baral[2] **Hamid Palangi**[1*] **Tomas Pfister**[1*]

[1]Google [2]Arizona State University

Abstract

Recently, decomposing complex problems into simple subtasks–a crucial part of human-like natural planning–to solve the given problem has significantly boosted the performance of large language models (LLMs). However, leveraging such planning structures during post-training to boost the performance of smaller open-source LLMs remains underexplored. Motivated by this, we introduce PLAN-TUNING, a unified post-training framework that (i) distills synthetic task decompositions (termed "planning trajectories") from large-scale LLMs and (ii) fine-tunes smaller models via supervised and reinforcement-learning objectives designed to mimic these planning processes to improve complex reasoning. On GSM8k and the MATH benchmarks, plan-tuned models outperform strong baselines by an average $\sim 7\%$. Furthermore, plan-tuned models show better generalization capabilities on out-of-domain datasets, with average $\sim 10\%$ and $\sim 12\%$ performance improvements on OlympiadBench and AIME 2024, respectively. Our detailed analysis demonstrates how planning trajectories improves complex reasoning capabilities, showing that PLAN-TUNING is an effective strategy for improving task-specific performance of smaller LLMs.

1 Introduction

Natural planning aligns more with real-world tasks such as trip or meeting planning (Zheng et al., 2024). Decomposing complex problems into simpler subtasks is a key to human-like natural planning (Jiao et al., 2024; Parmar et al., 2025). For instance, when prompted with *"How to plan a trip to . . . "*, recent LLMs naturally generate subtasks such as selecting a mode of transport, estimating a budget, and determining the trip duration

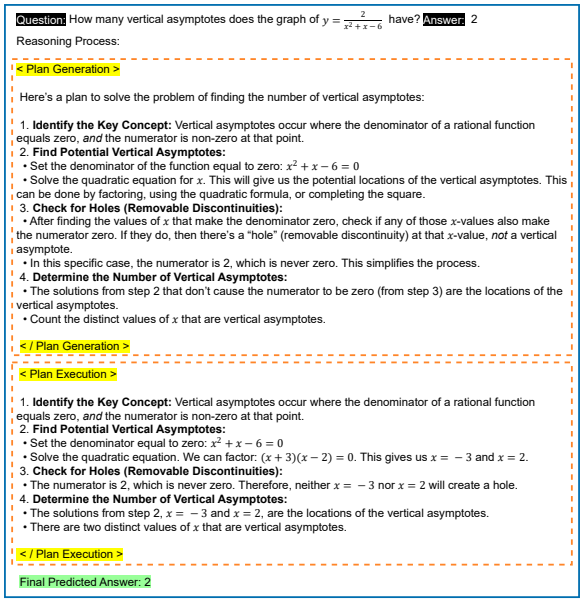

Figure 1: An example planning trajectory for a problem adapted from the MATH (Hendrycks et al., 2021).

(see App. A for examples). Recent large-scale LLMs such as o4 (Zhong et al., 2024), Gemini-2.0-Pro (Team et al., 2023), and Deepseek-v3 (Liu et al., 2024) have demonstrated this ability, and it significantly boosts their performance on complex reasoning tasks (Rein et al., 2024; He et al., 2024). Figure 1 illustrates a synthetic planning trajectory distilled from Gemini-2.0-Pro: a complex problem is first decomposed into intermediate subgoals (e.g., "identify relevant quantities," "formulate equations," "solve subexpressions"), which guide the model through a structured solution path.

However, these capabilities have largely been explored at inference time and on large-scale proprietary models (Wang et al., 2024a; Parmar et al., 2025); smaller open-source LLMs still struggle to leverage the decomposition step from natural planning effectively, limiting their performance on complex tasks. Thus, we address the research question: "**Can we improve the perfor-**

[†]Please correspond with {mihirparmar,hamidpalangi}@google.com
[*]Joint last authors

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 21431–21445
November 4-9, 2025 ©2025 Association for Computational Linguistics

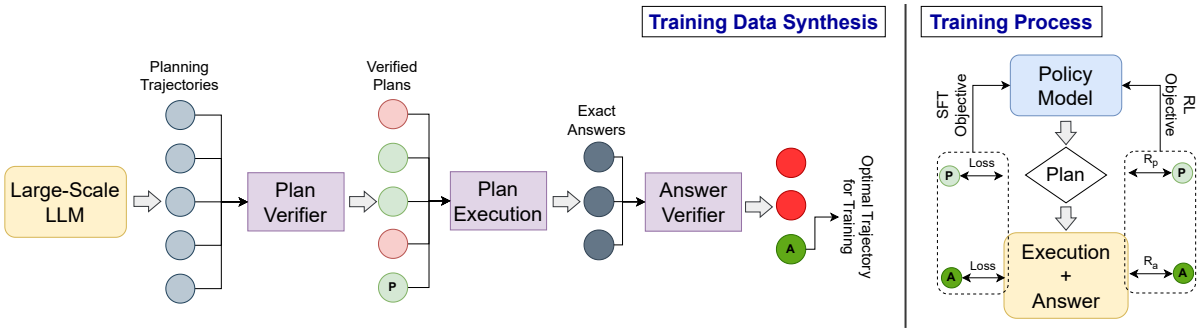

Figure 2: **Left:** Large-scale LLM generates multiple planning candidates (green = high-quality, red = low-quality). A *Plan Verifier* scores plans, and an *Answer Verifier* confirms the final answer; only trajectories passing both become the training corpus. **Right:** These trajectories train the policy model via SFT and RL objectives. P and A denote gold synthetic planning and answer trajectories, while R_p and R_a are their respective rewards.

mance of smaller LLMs on complex tasks by incorporating such capabilities through post-training, rather than relying solely on inference-time prompts?" To this end, we propose PLAN-TUNING, a post-training method that uses synthetic planning trajectories—sequences of natural decomposition steps—to teach a model how to plan as part of its parameterized knowledge. PLAN-TUNING incorporates planning trajectories in supervised and Reinforcement Learning (RL) settings, with customized objectives (loss) and reward functions to improve planning capabilities. Our plan-tuned smaller LLMs show improved reasoning skills, focused on mathematical reasoning.

For distilling high-quality planning trajectories from large-scale LLM, we leverage the Best-of-$\mathcal{N}$ approach from the recent PlanGEN framework proposed by Parmar et al. (2025). For the MATH and GSM8k (Cobbe et al., 2021) training sets, we generate five candidate plans per problem. Each plan is then evaluated by the verification agent from PlanGEN, and only those exceeding a predefined quality threshold are retained. Because gold final answers are available for training data, we next execute every retained plan and verify that it produces the correct final answer. At last, we include only those trajectories in PLAN-TUNING that both pass the agent-based scoring threshold and yield the correct solution; all others are discarded. The whole process is illustrated in Figure 2.

In PLAN-TUNING, we explore two post-training paradigms, supervised fine-tuning (SFT) and RL to incorporate planning in smaller LLMs. In SFT, we examine (1) an end-to-end setting where the model learns to map problem statements directly to a step-by-step plan and final answer, and (2) a two-stage pipeline that first generates only the plan and then executes it to derive the solution. In RL, we introduce Group Relative Policy Optimization (GRPO) (Shao et al., 2024), in which we augment the RL objective with planning-specific rewards to incorporate high-quality plan generation. Both Proximal Policy Optimization (PPO) (Ouyang et al., 2022) and Direct Preference Optimization (DPO) (Rafailov et al., 2023) depend on human-annotated or model-annotated preference data to learn a reward signal, which requires a separate data-collection or synthesis pipeline, but our work do not focus on synthesizing data for preference optimization. By contrast, GRPO leverages our existing LLM-based plan verifier to generate rewards without extra annotation effort. Furthermore, recent studies (Shao et al., 2024) have demonstrated that GRPO outperforms PPO for mathematical reasoning tasks. For these reasons, we selected the SOTA method, GRPO, in our preliminary study. We evaluate PLAN-TUNING against two strong baselines: an SFT model trained using reasoning chains and answers, and a vanilla GRPO model that optimizes preferences without any planning-based reward.

We evaluate four mathematical reasoning benchmarks—two in-domain (GSM8k and MATH) for both training and evaluation, and two out-of-domain (OlympiadBench (He et al., 2024) and AIME (Sun et al., 2025)) to assess generalization. For SFT, we use Gemma-3-12B-it (Team et al., 2025) and Qwen3-8B (Yang et al., 2024), and for RL, we use Gemma-3-1B-it, and Qwen3-4B. Across both in-domain tasks, plan-tuned models consistently outperform these baselines, yielding $\sim 7\% \uparrow$ on GSM8k and $\sim 20\% \uparrow$ on MATH. They also demonstrate stronger reasoning-based generalization, achieving $\sim 10\% \uparrow$ and $\sim 12\% \uparrow$ on

OlympiadBench and AIME, respectively. Moreover, our detailed analysis results in several interesting findings, such as mixing GSM8k and MATH during plan-tuning actually degrades performance, and plan-tuned models substantially improve performance on longer reasoning chains. In summary, our work proposes PLAN-TUNING, a method to distill planning trajectories from large-scale LLMs and fine-tune smaller LLMs to incorporate this ability to solve complex problems. We believe that PLAN-TUNING can be an effective method for improving complex reasoning in smaller LLMs.

2 Related Works

Decomposing complex problems into easy and smaller sub-tasks has been proven effective to improve performance of LLMs. Many prompt-level methods (Liu et al., 2023; Patel et al., 2022; Kuznia et al., 2022) has been proposed to this end. For instance, Least-to-Most (Zhou et al., 2023) prompting iteratively breaks down a complex problem into a series of simpler subproblems and then solve them in sequence. Similarly, Plan-and-Solve (Wang et al., 2023) prompting consists of two components: first, devising a plan to divide the entire task into smaller subtasks, and then carrying out the subtasks according to the plan. In contrast to these prompt-level approaches, we aim to achieve this decomposition ability via post-training of small-scale LLMs at inference-time. Thus, we present related literature in these two areas: (1) Inference-time scaling methods for LLMs, and (2) Post-training techniques for task decomposition.

Inference-time Scaling in LLMs Inference-time algorithms have recently shown a powerful way to optimize LLM output during inference, providing significant improvements in accuracy without scaling the model. Chain-of-thought (CoT) prompting and its variants (Wei et al., 2022; Kojima et al., 2022) showed that adding intermediate reasoning steps during inference-time greatly boosts the performance of LLMs. New methods have been proposed, such as self-consistency (Wang et al., 2022), which generates multiple reasoning chains from LLM and then selects the final answer based on majority voting. One very popular approach is the use of Monte Carlo Tree Search (MCTS) (Zhang et al., 2024a), which iteratively explores multiple solution paths during inference. This technique has been successfully applied to models like LLaMa-3-8B, which integrates a self-refinement mechanism that allows the model to revisit and improve its initial solutions over time. Another method, test-time optimization (Snell et al., 2024), focuses on dynamically adjusting computational resources during inference. By optimizing compute resources based on the complexity of a task, this approach strikes a balance between efficiency and accuracy, ensuring that difficult tasks receive more attention while simpler tasks are processed with fewer resources. Additionally, compute-optimal inference (Wu et al., 2024) highlights the importance of effectively distributing computational power during problem-solving tasks. Finally, repeated sampling (Brown et al., 2024) is a technique that uses multiple inference attempts to improve solution quality. Wang et al. (2024a) uses the inference time algorithms to improve LLMs planning capabilities to solve code synthesis problems. Recent works such as Parmar et al. (2025) show that better natural language planning improves downstream reasoning capabilities of underlying LLMs. In contrast to all these approaches focused on increasing compute at inference-time, our work focuses on inference-aware post-training with planning trajectories.

Post-training Methods for LLMs Past attempts have been made to improve the performance of smaller LMs using various types of training trajectories (Ouyang et al., 2022; Rafailov et al., 2023; Saeidi et al., 2024). Our work is closer to Jiao et al. (2024) where authors propose a two-stage approach: first, creating human-like, step-by-step planning trajectories, then automatically synthesizing detailed "process rewards" that score how accurately each trajectory follows its plan. In a similar direction, Song et al. (2024b) pioneers an exploration-driven paradigm, generating multiple candidate trajectories and iteratively refining them via performance feedback to improve long-horizon reasoning. Zhang et al. (2024b) proposes a pipeline that unifies demonstration data, self-play rollouts, and human preferences, demonstrating that diverse training signals yield more robust agent policies. Wang et al. (2024b) shows that incorporating explicitly negative trajectories during fine-tuning helps the model learn to avoid erroneous or unsafe actions. Other work investigates curriculum design and data curation strategies, revealing that carefully scheduled exposure to increasingly complex tasks significantly boosts final policy quality (Chen et al., 2024). Meanwhile, Song et al. (2024a)

demonstrates that scaling up trajectory volume and diversity is key to robust generalization. Unlike prior works that apply planning only at inference or via large-scale behavior cloning, PLAN-TUNING incorporates synthetic planning trajectories into small LLMs' parameters through supervised and RL post-training objectives.

3 Proposed Method

3.1 Task Formulation

Defining Task Decomposition as Part of Natural Planning In this work, we formalize natural planning—distinct from classical AI planning (which is not the focus of this work)—as the process of decomposing an input problem x into an ordered sequence of intermediate subgoals that guide its solution. Formally, we introduce a latent state space $\mathcal{S}$ of partial reasoning states, with the initial state $s_0 = x$. A planning trajectory $\tau = (\tau_1, \ldots, \tau_K)$ is then defined as a sequence of operators $\tau_k : \mathcal{S} \to \mathcal{S}$ such that

$$s_k \;=\; \tau_k(s_{k-1}) \quad \text{for } k = 1, \ldots, K, \qquad (1)$$

The final state s_K encodes sufficient information to extract the correct answer y. We model the planner as a conditional distribution as below:

$$\pi_\theta(\tau \mid x) \;=\; \prod_{k=1}^{K} \pi_\theta\big(\tau_k \mid s_{k-1}\big), \qquad (2)$$

This equation indicates that the policy model π_θ assigns high probability to trajectories that (i) follow high-quality, human-like decomposition patterns and (ii) lead to a correct final solution. For the scope of this work, we use smaller LLMs as policy models for generating planning trajectories.

Problem Statement We frame mathematical reasoning with planning as learning a conditional model that, given a problem statement $x \in \mathcal{X}$, jointly generates a planning trajectory $\tau = (\tau_1, \ldots, \tau_K) \in \mathcal{T}$ and a final answer $y \in \mathcal{Y}$, as shown in the below equation.

$$\pi_\theta(\tau, y \mid x) \;=\; \pi_\theta(y \mid \tau, x)\, \pi_\theta(\tau \mid x) \qquad (3)$$

The above equation is the policy of our plan-tuned LLM, parameterized by θ. We collect a training corpus $\mathcal{D} = \{(x_i, \tau_i, y_i)\}_{i=1}^{N}$ of synthetic, high-quality trajectories paired with gold answers. For this purpose, we use large-scale LLMs such as Gemini-2.0-Flash along with a filtering module to synthesize high-quality data. Our learning process thus train model to first decompose problem into solvable subproblems through natural planning, and leverage two different objectives: supervised fine-tuning, which aligns $\pi_\theta(\tau, y \mid x)$ with the observed (τ_i, y_i) pairs, and reinforcement learning, which maximizes expected rewards to encourage better generation, execution and correct answers.

3.2 Data Generation

Figure 1 and Figure 3 show illustrative examples of planning trajectories and their execution to get the final answer for the given problem[1]. Here, we provide a detailed discussion about the distillation of these high-quality planning trajectories using Gemini-2.0-Flash. For this purpose, we utilize the training sets of GSM-8k and MATH.

Data Synthesis Let us denote the training set as $\{x_i\}_{i=1}^{N}$. Motivated by Parmar et al. (2025), for each problem x_i, we employ a method similar to the PlanGEN (Best of $\mathcal{N}$) (with $\mathcal{N} = 5$ and temperature to 0.7 for the underlying LLM $\mathcal{M}$ for diversity) to synthesize five distinct natural-language planning trajectories:

$$\{\tau_i^{(1)}, \ldots, \tau_i^{(5)}\} \;\sim\; \mathcal{M}(\cdot \mid x_i). \qquad (4)$$

Each trajectory $\tau_i^{(n)}$ is passed through constraint-based verification agent—adapted from Parmar et al. (2025)—to compute a plan quality score.

$$s_i^{(n)} = R_{\text{ver}}(\tau_i^{(n)}), \qquad (5)$$

This score assesses coherence, logical soundness, and alignment with human-like decomposition patterns. Simultaneously, we execute each $\tau_i^{(n)}$ by feeding it into our execution module (another underlying LLM) to obtain an answer $y_i^{(n)}$. This yields a candidate set as below, capturing both plan quality and solution correctness.

$$\big\{\, x_i,\ \tau_i^{(n)},\ y_i^{(n)},\ s_i^{(n)} \big\}_{\substack{i=1,\ldots,N,\\ n=1,\ldots,5}}$$

The whole process of doing this data synthesis is presented in Figure 2, and a detailed example is presented in Figure 3. Also, prompts used for this data synthesis method are provided in App. B.

[1]More examples are provided at https://github.com/Mihir3009/plan-tuning

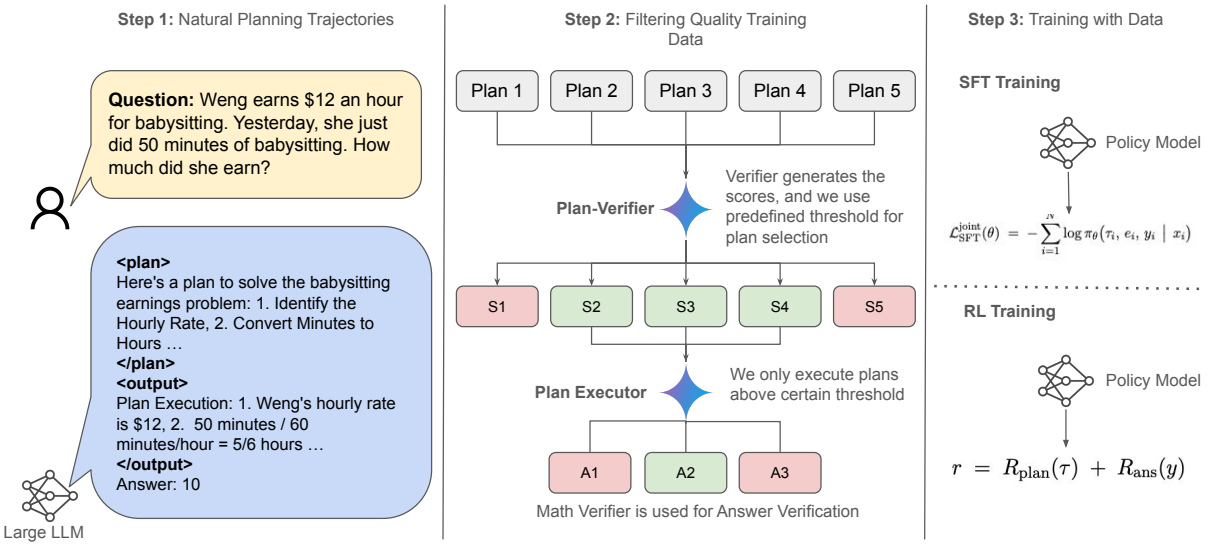

Figure 3: Overview of the PLAN-TUNING pipeline. First, a large LLM generates multiple candidate natural-language planning trajectories for each problem. Next, a Plan Verifier scores and filters these trajectories, and a Math Verifier executes and validates only those above a quality threshold. Finally, the curated plan–answer pairs are used to train the target model via both SFT and RL (GRPO) objectives.

Training Data Quality To ensure high-quality supervision, we apply a two-stage filtering process. First, we retain only trajectories whose verification score exceeds a threshold $\alpha = 80$:

$$s_i^{(n)} \geq 80, \tag{6}$$

This threshold we decided based on manual analysis presented in Parmar et al. (2025) that plans above this score have a high likelihood of yielding correct solutions. Now, for the training set, we have the gold final answer available. Hence, we validate each selected trajectory by checking execution correctness, i.e., $y_i^{(n)} = y_i^*$, where y_i^* is the gold answer. Only those $(x_i, \tau_i^{(n)}, y_i^{(n)})$ triples satisfying both criteria are included in the final training corpus; all remaining candidates are discarded. This selection yields a dataset of `<problem, plan, plan execution, final answer>` that balances plan quality with solution accuracy, creating high-quality training data.

3.3 PLAN-TUNING

We utilize two post-training methods: (i) supervised fine-tuning (SFT), and (ii) reinforcement learning via Gradient-based Reward Policy Optimization (GRPO)—each aimed to incorporate planning abilities in underlying LLMs.

3.3.1 PLAN-TUNING: SFT Training

In SFT, we compare two approaches: (1) joint plan and answer Generation teaches models to produce complete solutions (plans, step-by-step execution, and final answers) from problems, while (2) plan-only generation focuses exclusively on creating high-quality plans. These methods use different loss functions to optimize model parameters, with the joint approach minimizing negative log-likelihood across all solution components and the plan-only approach focusing only on plan quality.

Method 1 The model learns to map each problem x_i to the concatenated sequence (τ_i, e_i, y_i), where τ_i is the plan, e_i is the step-by-step execution of that plan, and y_i is the final answer. We minimize the negative log-likelihood using below:

$$\mathcal{L}_{\text{SFT}}^{\text{joint}}(\theta) = -\sum_{i=1}^{N} \log \pi_\theta(\tau_i, e_i, y_i \mid x_i) \tag{7}$$

Method 2 The model focuses exclusively on generating high-quality plans τ_i. The objective we use to optimize is given below:

$$\mathcal{L}_{\text{SFT}}^{\text{plan}}(\theta) = -\sum_{i=1}^{N} \log \pi_\theta(\tau_i \mid x_i) \tag{8}$$

Once we have generated a high-quality plan τ_i, we use the same off-the-shelf base LLM to execute that plan and produce the final answer. No additional training is performed on this execution model. In the rest of the paper, we refer to method 1 as $\mathcal{M}_1$, and method 2 as $\mathcal{M}_2$.

3.3.2 PLAN-TUNING: GRPO Training

In this approach, we apply GRPO, a policy-gradient algorithm that directly maximizes sequence-level returns. Let q denote a problem statement and $o = (o_1, \ldots, o_{|o|})$ the model's generated output/rollouts (the plan, its execution, and the final answer). We define a combined reward for each sampled trajectory as below:

$$r(.) \;=\; R_{\text{plan}}(\tau) \;+\; R_{\text{ans}}(y), \qquad (9)$$

where $R_{\text{plan}}(\tau)$ is the plan-quality score produced by our similarity function (discussed later in the section), and $R_{\text{ans}}(y)$ is a binary correctness indicator (2 if y matches the gold answer, 0 otherwise).

Background GRPO, a PPO (Schulman et al., 2017) variant, estimates the advantage by aggregating reward scores of a group of n sampled responses to a given query q. Formally, let π_θ and $\pi_{\theta_{old}}$ be the current and old policy models respectively. Let q and o_i be the query and i^{th} response sampled from the dataset and the old policy, respectively. Let $r(.)$ be the reward function, which measures the correctness of a given response. Then, the GRPO objective is defined as follows:

$$\begin{aligned}\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}\Big[\, q \sim \mathcal{D},\ \{o_i\}_{i=1}^{n} \sim \pi_{\theta_{\text{old}}}(O \mid q)\Big] \\ \Bigg\{ \frac{1}{n}\sum_{i=1}^{n} \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \min\Big[\tfrac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,o_{i,<t})}\, \hat{A}_{i,t}, \\ \text{clip}\big(\tfrac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,o_{i,<t})}, 1-\epsilon,\, 1+\epsilon\big)\, \hat{A}_{i,t} \Big] \\ - \beta\, D_{KL}\big[\pi_\theta \,\|\, \pi_{\text{ref}}\big] \Bigg\}\end{aligned}$$

Here, the advantage is calculated as the normalized reward, i.e., $\hat{A}_{i,t} = \tilde{r}(o_i) = \frac{r(o_i) - \text{mean}(r)}{\text{std}(r)}$. This $\hat{A}_{i,t}$ centers and scales each trajectory's return r across the sampled batch. By weighting each token's log-probability gradient by $\hat{A}$, GRPO amplifies updates for outputs yielding above-average combined reward and suppresses those with below-average reward.

Our Modification (Plan GRPO) In the above formulation, our focus is to modify $r(.)$ function, and incorporate planning-based reward. So, the final reward function consists: (1) plan-quality reward $R_{\text{plan}}(\tau)$ ensures that generated trajectories closely match high-quality synthetic plans distilled from large-scale LLMs, and (2) answer correctness reward $R_{\text{ans}}(y)$ guarantees that following these plans leads to right solutions. Thus, the final reward is presented in Equation 9. By normalizing and combining these signals, GRPO fine-tunes the policy model π_θ to generate both good plans, accurate execution, and a correct final answer.

Dataset	Train	Eval
GSM-8k	6,586	1,319
MATH	10,000	500
OlympiadBench	–	674
AIME	–	933

Table 1: Statistics of the datasets used in our experiments. Training set sizes are shown for in-domain benchmarks, and evaluation set sizes for both in-domain and out-of-domain benchmarks.

Details on Planning Reward $R_{\text{plan}}(\tau)$ is computed by measuring how closely a model-generated plan τ matches its reference τ^* using our Gemini-based similarity scorer. Here, reference τ^* is the plan synthesized from above section. It is distilled from large-scale LLMs (Gemini-2.0 in our case). We first extract the plan segments. We parse each model completion to pull out only the `<plan>` portion. Then, we prompt Gemini for similarity. We construct a natural-language prompt comparing the generated plan to the gold plan and send it to Gemini-2.0-Flash, asking it to rate plan similarity on a 0–1 scale. At last, we parse the numeric score. We apply a flexible regular expression to the returned text to extract the score.

$$R_{\text{plan}}(\tau_i) \;=\; \text{Score}_{\text{Gemini}}\big(\tau_i, \tau_i^*\big) \;\in\; [0, 1].$$

These per-example rewards are then fed into our GRPO objective $\mathcal{J}_{\text{GRPO}}$, so that higher-quality plans receive proportionally larger policy-gradient updates. The prompt for calculating $R_{\text{plan}}(\tau)$ is presented in App. C.

4 Results and Analysis

4.1 Experimental Setup

Datasets We evaluate our PLAN-TUNING on four mathematical reasoning benchmarks. As shown in Table 1, GSM-8k (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) provide in-domain datasets for both training and evaluation. After filtering out lower-quality examples, the GSM-8k training set contains 6,586 problems (from an original 7,500), with 1,319 held out for evaluation. The

Models	Methods	Datasets Evaluated using $\mathcal{M}$(GSM8k)			Datasets evaluated using $\mathcal{M}$(MATH)		
		GSM8k	OlympiadBench (MATH)	AIME 2024	MATH	OlympiadBench (MATH)	AIME 2024
Gemma-3	Baseline SFT	81.43	31.45	28.62	65.40	22.11	14.04
	Vanilla GRPO	19.94	02.52	00.00			
	$\mathcal{M}_1$	86.20	32.20	32.27	74.20	27.00	20.04
	$\mathcal{M}_2$	**86.35**	**34.27**	**33.98**	**83.80**	**31.75**	**29.37**
	Plan GRPO	**28.35**	**04.30**	**03.00**			
Qwen-3	Baseline SFT	80.74	28.78	29.05	53.20	12.02	05.57
	Vanilla GRPO	84.27	24.63	**17.68**			
	$\mathcal{M}_1$	**87.11**	**30.27**	**32.05**	73.80	23.44	19.08
	$\mathcal{M}_2$	85.67	30.12	31.94	**79.40**	**31.01**	**31.51**
	Plan GRPO	**86.57**	**25.07**	15.22			

Table 2: This table reports the accuracy (%) of the base LLM and its two plan-tuned variants (custom SFT and GRPO) on four mathematical reasoning benchmarks. Columns 1–2 show in-domain performance on GSM8K and MATH. Columns 3–4 present out-of-domain generalization on OlympiadBench and AIME 2024. These results demonstrate that leveraging synthetic planning trajectories via both SFT and RL objectives improves reasoning accuracy in smaller LLMs. $\mathcal{M}$ (GSM8k): Model trained using GSM8k dataset, $\mathcal{M}$ (MATH): Similar for MATH.

MATH training set comprises 10,000 problems (from 12,000), with 500 held-out for evaluation. To evaluate out-of-domain generalization, we use the text-only version of OlympiadBench (MATH) (674 problems) (He et al., 2024) and AIME (933 problems), for only evaluation purposes.

Models For data synthesis, we use **Gemini-2.0-Flash** (Checkpoint: April 2025). We fine-tune four pretrained HuggingFace checkpoints: **Gemma-3-12B-It** and **Qwen3-8B** for PLAN-TUNING: SFT; and **Gemma-3-1B-It** and **Qwen3-4B** for PLAN-TUNING: GRPO.

Baselines Our two baselines for these training paradigms: (i) an SFT model that learns to output conventional chain-of-thought reasoning and final answers, and (ii) a "vanilla" GRPO model optimized only on answer correctness $R_{\text{ans}}(y)$ without any planning-specific rewards $R_{\text{plan}}(\tau)$.

Proposed Experiments For PLAN-TUNING: SFT, we experiment with both methods: (i) $\mathcal{M}_1$, a joint plan-and-answer formulation, where the model maps each input x to the tuple $\langle \text{Plan}, \text{Execution}, \text{Answer} \rangle$, and (ii) $\mathcal{M}_2$, a plan-only variant in which the model simply generates $\langle \text{Plan} \rangle$. Both SFT variants use a batch size of 8, an adaptive learning rate of 5×10^{-6}, a single training epoch, and a cosine learning-rate scheduler. Second, we apply GRPO Training: a vanilla GRPO baseline that optimizes only the answer correctness reward $R_{\text{ans}}(y)$, and a planning-specific GRPO that additionally incorporates the Gemini-based planning reward $R_{\text{plan}}(\tau)$ into the objective $R_{\text{plan}} + R_{\text{ans}}$. For GRPO, we use a batch size of 32, the same learning rate and scheduler as SFT, one epoch, 4 rollouts per policy update, and a KL-coefficient of 0.04. Due to resource and time constraints, our GRPO experiments are limited to the GSM8k dataset.

Metrics We report dataset-specific accuracy on each benchmark to assess in-domain performance and out-of-domain generalization. In particular, we use micro-average accuracy for OlympiadBench similar to He et al. (2024), and Exact Match (EM) for all other datasets.

4.2 Main Results

Baseline Performance on In-Domain Tasks From Table 2, the off-the-shelf SFT model trained on GSM8K achieves 81.43% accuracy when evaluated on GSM8K and 65.4% on the MATH benchmark for the Gemma-3. In comparison, the Qwen-3 SFT baseline shows 80.74% on GSM8K and only 53.2% on MATH. This is a $\sim 15\%$ drop between the two in-domain datasets, highlighting that, without explicit natural planning, smaller LLMs struggle to generalize from more complex and constrained math word problems where the diverse, multi-step reasoning is required.

Supervised Fine-Tuning Variants Introducing supervised planning trajectories yields consistent gains across both model families. For Gemma-

3, the joint plan-and-answer SFT ($\mathcal{M}_1$) improves GSM8K accuracy to 86.2% ($+4.8\%$) and MATH to 74.2% ($+8.8\%$), while the plan-only SFT ($\mathcal{M}_2$) further boosts these to 86.35% on GSM8K and an 83.8% on MATH. Qwen-3 exhibits similar trends: joint SFT improves up to 87.11% on GSM8K ($+6.4\%$) and 73.8% on MATH ($+20.6\%$), whereas plan-only fine-tuning yields 85.67% and 79.4%, respectively. In particular, large gains on MATH suggest that guiding the model to focus purely on plan generation better incorporates the structured decomposition strategies needed for complex, multi-step reasoning tasks.

Out-of-Domain Generalization When evaluated on Olympiad-level math benchmarks, the importance of PLAN-TUNING is even more prominent. Gemma-3 baseline SFT achieves only 31.45% on OlympiadBench and 14.04% on AIME. Joint SFT (Method 1) improves it to 32.2% and 20.04%; plan-only SFT (Method 2) improves them further to 34.27% and 29.37%. Qwen-3 follows the same trend: baseline SFT is 28.78%/5.57% (OlympiadBench/AIME), joint SFT 30.27%/19.08%, and plan-only 30.12%/31.51%. These gains, often improving twice or more AIME performance, demonstrate that high-quality planning exemplars for training are especially critical for tackling novel, complex Olympiad-level math problems.

Improvements with Plan-GRPO From Table 2, we present results for vanilla-GRPO and Plan-GRPO where the model is trained using the GSM8k dataset. From the results, we can observe that, for Gemma-3-1B-It, plan-GRPO improves performance on GSM8k to 28.35% compared to vanilla-GRPO (19.94%). A similar trend can be observed for the Qwen3-4B model in terms of GSM8k. Now, on out-of-domain datasets, for Gemma-3, plan-tuned models are achieving better generalization. However, for Qwen-3 models, we are seeing a performance drop for AIME. Training with GRPO is highly dependent on reward functions. Also, the lower performance on OlympiadBench and AIME is subject to the smaller sizes of LLMs, 1B and 4B, used for GRPO training.

Lower performance of GRPO is attributed to model-size. Our GRPO experiments were conducted on significantly smaller models (Gemma-3-1B-It and Qwen3-4B) due to resource constraints, whereas SFT experiments used larger models (Gemma-3-12B-It and Qwen3-8B). This size gap likely contributed to the performance drop, especially given that RL-based training is more sensitive to model capacity for stable policy optimization. To assess the impact of model size, we conducted an additional case study during the limited rebuttal period, where we trained Gemma-3-1B-It using SFT on GSM8K. The resulting accuracy is 5.16%, which is even lower than the results on vanilla-GRPO (19.94%). This supports that model size is a major factor in the performance gap.

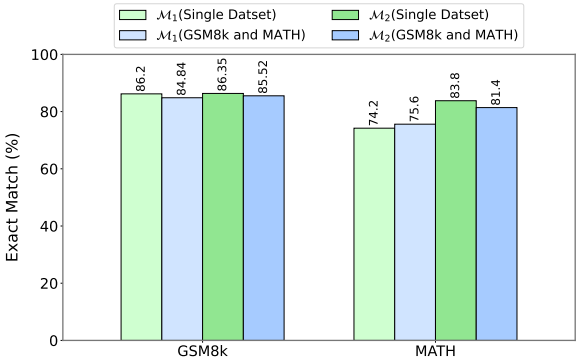

Figure 4: Comparison of plan-tuning accuracy on GSM8K and MATH when trained on each dataset individually *vs.* their combined corpus. $\mathcal{M}_i$ (Single Dataset) indicates that the respective method is trained on a given dataset (green), while other indicates that the respective method is trained on both datasets (blue).

4.3 Analysis

Synthesis and Distribution-shift Considerations Together, these results indicate that embedding explicit natural planning into smaller LLMs—first via supervised trajectories and then through an RL-based policy refinement—yields substantial improvements in both in-domain accuracy and out-of-domain robustness. The significant improvements on AIME highlight how enforcing intermediate correctness reduces arithmetic drift, while consistent gains across two model families indicate the generality of our PLAN-TUNING post-training approach in bridging distributional gaps between training and evaluation domains.

Longer Reasoning Chains on OlympiadBench The reason behind the higher performance of PLAN-TUNED models is that SFT provides clear templates for breaking down multi-step proofs, giving the model a reliable blueprint for structured decomposition. Plan-GRPO builds on this by rewarding diverse, high-quality plan, encouraging the model to flexibly combine reasoning fragments when it encounters novel or unexpected subgoals.

Method	GSM8k	AIME 2024
$\mathcal{M}$(single-stage)	83.32	22.72
$\mathcal{M}$(two-stage)	**87.11**	**32.05**

Table 3: Performance comparison of model trained with data created using single-stage *vs.* two-stage filtering methods on GSM8k and AIME 2024.

Variance in Plan Quality For short, arithmetic-focused GSM8K tasks, both methods quickly converge to high accuracy using straightforward reasoning chains. On more complex olympiad-level tasks—like geometry or combinatorics subcases—the RL objective in Plan-GRPO improves the impact of diverse, intermediate-valid plans, enabling the discovery of novel solution paths.

Mixing GSM8k and MATH during PLAN-TUNING Figure 4 compares PLAN-TUNING on GSM8K and MATH when training on each dataset separately*vs.* training on their combination. In both the joint plan-and-answer ($\mathcal{M}_1$) and plan-only (($\mathcal{M}_2$)) variants, tuning on a single dataset yields the best accuracy on its own benchmark, 86.2% on GSM8K and 83.8% on MATH—whereas mixing the two corpora in a sample batch drops GSM8K performance by $\sim 2\%$ and fails to improve MATH beyond its single-dataset result. This consistent degradation (except slight improvement in the $\mathcal{M}_1$(GSM8k and MATH)) indicates that PLAN-TUNING relies on dataset-specific patterns of problem structure and reasoning style; when these patterns become heterogeneous, the model struggles to internalize a coherent planning policy, so domain-focused tuning can be more effective. The slight improvement in the $\mathcal{M}_1$(GSM8k and MATH) (75.6%) compared to only $\mathcal{M}_1$ (MATH) (74.2%) may seem to contradict our broader claim based on other results in Figure 4. However, in $\mathcal{M}_1$, the model jointly learns plan, execution, and answer generation. This may allow the model to benefit slightly from additional GSM8K examples, which, although simpler, still provide useful structure for learning generalizable output formatting.

Importance of two-stage filtering for data synthesis We conduct an ablation study in which we replace the two-stage filtering with using only the Plan Verifier (Figure 2) threshold to select better plans for training. We performed experiments using Qwen3-8B on GSM8K for $\mathcal{M}_1$, where the training data was filtered only based on the Plan Verifier. We evaluate it on GSM8k and one OOD dataset, AIME 2024. Here, $\mathcal{M}$(single-stage) indicates the model trained with data created using single-stage filtering, and $\mathcal{M}$(two-stage) indicates the model trained with data created using two-stage filtering. The results from Table 3 showcase that two-stage filtering leads to better improvements: a $\sim 4\%$ improvement on GSM8K and $\sim 9\%$ on AIME 2024. These findings support that two-stage filtering plays a critical role in improving the performance. This ablation shows the importance of combining plan quality with answer correctness for synthesizing good training data.

Qualitative Analysis App. D (Figure 5) provides an example of how PLAN-TUNED models improve reasoning and problem-solving capabilities over SFT. In this example, we demonstrate how planning trajectories steer the model toward a structured solution. The vanilla SFT model misapplies the 150% increase to the combined cost, yielding an incorrect profit of $120,000. In contrast, the plan-tuned model explicitly decomposes the task into four subtasks—total cost, value increase, new house value, and profit—and arrives at the correct answer of $70,000. The SFT pipeline collapses subtasks and propagates an early error, whereas our PLAN-TUNING framework enforces step-by-step reasoning aligned with human planning. This case highlights the importance of decomposing complex problems into clear intermediate goals to improve accuracy.

5 Conclusions

We introduce PLAN-TUNING, a novel post-training method that incorporates synthetic natural planning trajectories into smaller LLMs' parameters, rather than relying solely on inference-time prompts. We develop two complementary post-training strategies—SFT to imitate high-quality plan decompositions and GRPO to reinforce plan quality alongside answer correctness—thereby teaching models both how to plan and how to execute. Across two in-domain datasets (GSM8K, MATH), PLAN-TUNED models achieve an avg. $\sim 7\% \uparrow$ accuracy over baselines; on out-of-domain benchmarks (OlympiadBench, AIME), they significantly improved the performance. Through analyses, we show (i) that a good plan relies on dataset-specific consistency—mixing heterogeneous sources degrades performance, and (ii) that two-stage filtering is important.

Limitations

Our approach relies on a large base LLM to generate and verify planning trajectories; errors or biases in that upstream model can propagate into training data and limit downstream gains. Generating, filtering, and executing multiple candidate plans per example incurs non-trivial computational cost and implementation complexity, which may hinder large-scale or real-time applications. All experiments focus on mathematical reasoning; it remains to be validated whether PLAN-TUNING generalizes to other problem types (e.g., commonsense, code synthesis) without substantial adaptation. Key thresholds (e.g., plan-quality cutoff, reward weighting) require manual tuning and may not transfer directly across datasets or languages. For GRPO, we used only LLM-based verification for planning reward, we will certainly explore more objective-function evaluations such as ROSCOE (Golovneva et al., 2023) as future work.

Ethics Statement

The use of proprietary LLMs such as GPT-4, Gemini, and Claude-3 in this study adheres to their policies of usage. We have used AI assistants (Grammarly and Gemini) to address the grammatical errors and rephrase the sentences.

References

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. 2024. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9354–9366, Bangkok, Thailand. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*, 9.

Olga Golovneva, Moya Peng Chen, Spencer Poff, Martin Corredor, Luke Zettlemoyer, Maryam Fazel-Zarandi, and Asli Celikyilmaz. 2023. Roscoe: A suite of metrics for scoring step-by-step reasoning. In *The Eleventh International Conference on Learning Representations*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Fangkai Jiao, Chengwei Qin, Zhengyuan Liu, Nancy F. Chen, and Shafiq Joty. 2024. Learning planning-based reasoning by trajectories collection and process reward synthesizing. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 334–350, Miami, Florida, USA. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Kirby Kuznia, Swaroop Mishra, Mihir Parmar, and Chitta Baral. 2022. Less is more: Summary of long instructions is better for program synthesis. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4532–4552.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, and 1 others. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Mihir Parmar, Xin Liu, Palash Goyal, Yanfei Chen, Long Le, Swaroop Mishra, Hossein Mobahi, Jindong Gu, Zifeng Wang, Hootan Nakhost, and 1 others. 2025. Plangen: A multi-agent framework for generating planning and reasoning trajectories for complex problem solving. *arXiv preprint arXiv:2502.16111*.

Pruthvi Patel, Swaroop Mishra, Mihir Parmar, and Chitta Baral. 2022. Is a question decomposition unit all we need? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4553–4569.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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*.

Amir Saeidi, Shivanshu Verma, Aswin RRV, and Chitta Baral. 2024. Triple preference optimization: Achieving better alignment with less data in a single step optimization. *arXiv preprint arXiv:2405.16681*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

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 preprint arXiv:2408.03314*.

Yifan Song, Weimin Xiong, Xiutian Zhao, Dawei Zhu, Wenhao Wu, Ke Wang, Cheng Li, Wei Peng, and Sujian Li. 2024a. AgentBank: Towards generalized LLM agents via fine-tuning on 50000+ interaction trajectories. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 2124–2141, Miami, Florida, USA. Association for Computational Linguistics.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. 2024b. Trial and error: Exploration-based trajectory optimization of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7584–7600, Bangkok, Thailand. Association for Computational Linguistics.

Haoxiang Sun, Yingqian Min, Zhipeng Chen, Wayne Xin Zhao, Zheng Liu, Zhongyuan Wang, Lei Fang, and Ji-Rong Wen. 2025. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. *arXiv preprint arXiv:2503.21380*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Evan Z Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, William Song, Vaskar Nath, Ziwen Han, Sean M Hendryx, Summer Yue, and Hugh Zhang. 2024a. Planning in natural language improves llm search for code generation. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

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, and 22 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024a. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *arXiv preprint arXiv:2406.07394*.

Jianguo Zhang, Tian Lan, RN Rithesh, Zhiwei Liu, Weiran Yao, Juntao Tan, Thai Quoc Hoang, Liangwei Yang, Yihao Feng, Zuxin Liu, and 1 others. 2024b. The agent ohana: Designing unified data and training pipeline for effective agent learning. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V Le, Ed H Chi, and 1 others. 2024. Natural plan: Benchmarking llms on natural language planning. *arXiv preprint arXiv:2406.04520*.

Tianyang Zhong, Zhengliang Liu, Yi Pan, Yutong Zhang, Yifan Zhou, Shizhe Liang, Zihao Wu, Yanjun Lyu, Peng Shu, Xiaowei Yu, and 1 others. 2024. Evaluation of openai o1: Opportunities and challenges of agi. *arXiv preprint arXiv:2409.18486*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and 1 others. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Example Prompts and Responses

These prompts exemplify everyday reasoning tasks that require multi-step planning and decision-making, making them ideal for evaluating LLMs' ability to perform structured, goal-oriented reasoning. From organizing a trip or scheduling a meeting to planning a birthday party or solving a math problem, each scenario involves decomposing a high-level objective into coherent subtasks. The corresponding responses reflect how a well-structured plan can guide users through complex decisions—highlighting both procedural thinking and domain knowledge. Such prompts can be used as qualitative case studies to assess models' competence in natural planning, temporal sequencing, and commonsense reasoning.

How to plan a trip to Japan

Response

1. **Decide on Travel Dates** – Choose the season (e.g., cherry blossom in spring or autumn foliage) and trip duration.
2. **Set a Budget** – Estimate total costs including flights, transport, lodging, food, and attractions.
3. **Check Entry Requirements** – Look into visa policies, passport validity, and any travel restrictions.
4. **Choose Destinations** – Select cities and regions (e.g., Tokyo, Kyoto, Osaka, Nara, Hokkaido, Okinawa) based on your interests.
5. **Book Flights and Accommodation** – Use platforms like Google Flights, Booking.com, or Airbnb; book early in peak seasons.
6. **Plan Transportation Within Japan** – Consider a Japan Rail Pass and IC cards (Suica, Pasmo) for local travel.
7. **Create an Itinerary** – Map out a day-by-day plan including travel time, attractions, and rest periods.
8. **Arrange Travel Essentials** – Pocket Wi-Fi/SIM card, power adapter (Type A/B), Japanese yen (cash is still widely used).
9. **Learn Local Etiquette** – Study basic customs (e.g., removing shoes indoors, quiet on trains, no tipping).
10. **Pack Appropriately** – Bring weather-appropriate clothing, walking shoes, and a small daypack for city exploration.

How to schedule a meeting with your PhD colleague

Response

1. **Define the Purpose** – Clarify the topic (e.g., paper discussion, experiment planning).
2. **Propose a Timeframe** – Suggest a few potential date/time slots that work for you.
3. **Use Scheduling Tools** – Use Doodle, Google Calendar, or Outlook to find mutual availability.
4. **Check Their Availability** – Send a quick message (email/Slack/WhatsApp) asking for their availability.
5. **Suggest a Duration** – Be clear about how long you expect the meeting to take (30 mins, 1 hour, etc.).
6. **Decide the Format** – In-person or virtual (Zoom, Teams, Google Meet).
7. **Send Calendar Invite** – Include topic, agenda (if needed), and location or meeting link.
8. **Follow Up Politely** – If they don't respond in a day or two, send a short, polite reminder.
9. **Prepare Materials** – If needed, send any relevant documents or reading material beforehand.
10. **Confirm on the Day** – Send a quick message confirming the meeting a few hours before, especially for virtual calls.

How to organize a party for the birthday

Response

1. **Set a Budget** – Decide how much you want to spend overall.
2. **Choose a Date & Time** – Pick a convenient day for the birthday person and guests.
3. **Pick a Venue** – At home, a restaurant, park, or rented space—based on budget and guest size.
4. **Decide a Theme (optional)** – Themes like retro, beach, movie night, or favorite colors add fun.
5. **Make a Guest List** – Know how many people you're inviting to plan food and space.
6. **Send Invitations** – Use digital invites (WhatsApp, email, Evite) or printed cards.
7. **Plan Food & Drinks** – Homemade, catered, or ordered—match it to the guest profile and theme.
8. **Arrange Decorations & Cake** – Buy or DIY decorations and order a cake in advance.
9. **Plan Activities or Entertainment** – Music, games, speeches, or photo booth to keep guests engaged.

10. **Prepare a Backup Plan** – Especially for outdoor events, have a Plan B in case of weather or other issues.

How to solve a given mathematical reasoning problem

Response

1. **Understand the Problem** – Read carefully and identify what is being asked.
2. **Identify Key Information** – Extract numbers, relationships, and conditions from the problem.
3. **Break Down the Problem** – Divide the complex problem into smaller, manageable parts.
4. **Visualize or Represent** – Use diagrams, equations, or tables if helpful.
5. **Recall Relevant Concepts** – Think of formulas, theorems, or reasoning strategies that apply.
6. **Formulate a Plan** – Decide on the method (e.g., algebra, logic, geometry) to approach the problem.
7. **Execute Step-by-Step** – Carry out your plan methodically, solving subparts if needed.
8. **Check Units and Logic** – Make sure the answer is in the correct form and logically consistent.
9. **Verify Your Answer** – Plug it back into the original problem to confirm it satisfies all conditions.
10. **Reflect on the Method** – Consider if there's a more efficient or alternative solution strategy.

B Data Synthesis Prompts

These prompts form a structured framework for evaluating and improving mathematical reasoning in large language models. The *Plan Generation Prompt* encourages models to decompose complex math problems into step-by-step solution strategies, fostering procedural thinking. The *Constraints Generation Prompt* identifies key logical and mathematical conditions that must be satisfied by any valid solution plan, serving as a verification checklist. Finally, the *Plan Verification Prompt* introduces a rigorous reward-based scoring scheme, allowing evaluators to assign interpretable, constraint-aware scores to the quality of generated plans. This framework promotes transparency, robustness, and fidelity in evaluating model reasoning capabilities.

Plan Generation Prompt

Prompt

Analyze the given maths question, and create a plan to solve it:

```
<question>
{question}
</question>
```

Feel free to break down the problem in whatever way you think is most effective. Consider key concepts, formulas, relevant facts, or any logical approach that would help solve this. Your task is to only provide a plan and not solve it during this process.

Constraints Generation Prompt

Prompt

You are an expert in identifying explicit and implicit constraints for verifying plans generated to solve complex maths problems. Your job is to generate those constraints for the following question, which can be helpful in verifying and evaluating the given plan.

```
<question>
{question}
</question>
```

Make sure to identify all constraints in the question. Please output the constraints as a list. DO NOT include any other text in your response.

Plan Verification Prompt

Prompt

Provide a reward score between -100 and 100 for the plan quality, using very strict standards. Do not give a full score above 95. Make sure the reward score is an integer.
Input:

```
{input}
```

Generated plan to evaluate:

```
{response}
```

Consider constraints below while evaluating:

```
{verification_prompt}
```

Make sure to check all the constraints before giving the reward score.
Please provide a reward in the format below:

- Steps: [step-by-step reasoning for the reward score]
- Score: [Strictly provide the reward score as an integer between -100 and 100]

C Prompt for Planning Reward

The box below defines the exact evaluation prompt we use to score planning quality in GRPO. It asks the model to compare a generated plan against a

Question: Josh decides to try flipping a house. He buys a house for $80,000 and then puts in $50,000 in repairs. This increased the value of the house by 150%. How much profit did he make?

SFT (Reasoning + Answer):

1. The house was worth $80,000 + $50,000 = 130,000
2. The value increased by 150% of $80,000 which is 1.5 * $80,000 = 120,000
3. So the house is now worth $130,000 + $120,000 = 250,000
4. So he made a profit of $250,000 - $130,000 = 120,000

Final Answer: 120000 (Wrong!!)

SFT applies the 150% increase to $130 k (total cost) instead of the $80 k base, whereas Plan-Tuning correctly applies it to the original price.

Plan-Tuned Model:

<plan>
1. Calculate the total cost: <description>
2. Calculate the increase in value: <description>
3. Calculate the new value of the house: <description>
4. Calculate the profit: <description>
</plan>
<Plan Execution>
1. The purchase price is $80,000 and the repair costs are $50,000. The total cost is $80,000 + $50,000 = $130,000.
2. The increase in value is 150% of the original purchase price, which is $80,000. 150% of $80,000 is (1.50) * $80,000 = $120,000.
3. The original purchase price was $80,000, and the increase in value is $120,000. The new value is $80,000 + $120,000 = $200,000.
4. The total cost was $130,000, and the final value of the house is $200,000. The profit is $200,000 - $130,000 = $70,000.

Answer: 70000 (Correct!!)
</Plan Execution>

Figure 5: Qualitative Example: Comparison of SFT vs. PLAN-TUNING on a sample problem from GSM8k.

gold plan, provide a brief similarity analysis, and emit a single scalar score on a 0–1 scale. This score then serves directly as the planning reward during RL fine-tuning.

D Qualitative Analysis

As shown in Figure 5, we provide an example that demonstrate how planning trajectories steer the model toward a structured solution.

Prompt

You are an expert evaluator of problem-solving plans. Compare the following two plans and rate their similarity on a scale from 0 to 1.
0.0 = Completely different plans with no shared approach or reasoning steps. 0.25 = Minimal similarity with some overlapping concepts but fundamentally different approaches. 0.5 = Moderate similarity with shared key ideas but significant differences in execution or reasoning. 0.75 = High similarity with mostly aligned reasoning and steps, with minor differences. 1.0 = Nearly identical plans that follow the same approach and reasoning steps.

Generated Plan:
{generated plan}

Gold Plan:
{gold plan}

First, provide a brief analysis of the similarity, then output only a single float number between 0 and 1, representing the similarity score.
Please **STRICTLY** use the format below:

```
Analysis: [brief analysis]
Score: [float number between 0 and 1]
```

(Note: this score will be used as the planning reward in GRPO.)