PlanGEN: A Multi-Agent Framework for Generating Planning and Reasoning Trajectories for Complex Problem Solving

Mihir Parmar[1,2] **Xin Liu**[1] **Palash Goyal**[1] **Yanfei Chen**[1] **Long Le**[1]
Swaroop Mishra[1] **Hossein Mobahi**[1] **Jindong Gu**[1] **Zifeng Wang**[1]
Hootan Nakhost[1] **Chitta Baral**[2] **Chen-Yu Lee**[1] **Tomas Pfister**[1*] **Hamid Palangi**[1*]

[1]Google [2]Arizona State University

Abstract

Recent agent frameworks and inference-time algorithms often struggle with natural planning problems due to limitations in verifying generated plans or reasoning and varying complexity of instances within a single task. Many existing methods for these tasks either perform task-level verification without considering constraints or apply inference-time algorithms without adapting to instance-level complexity. To address these limitations, we propose PlanGEN, a model-agnostic and easily scalable agent framework with three key components: constraint, verification, and selection agents. Specifically, our approach proposes constraint-guided iterative verification to enhance performance of inference-time algorithms–Best of $\mathcal{N}$, Tree-of-Thought, and REBASE. In PlanGEN framework, the selection agent optimizes algorithm choice based on instance complexity, ensuring better adaptability to complex planning problems. Experimental results demonstrate significant improvements over the strongest baseline across multiple benchmarks, achieving state-of-the-art results on NATURAL PLAN (∼8%↑), OlympiadBench (∼4%↑), DocFinQA (∼7%↑), and GPQA (∼1%↑). Our key finding highlights that constraint-guided iterative verification improves inference-time algorithms, and adaptive selection further boosts performance on complex planning and reasoning problems.

1 Introduction

In many real-world tasks, we often encounter the word "plan for" or "plan to". For example, prompting large language models (LLMs) with *"Let us make a plan to ..."* yields completions such as "travel the world", "schedule a meeting", or "organize a visit" (App. B for examples). The tasks related to such prompts are referred to as "natural planning" which is different from the traditional classical AI planning, where given an initial state, goal, descriptions of executability and effect of actions, one has to come up with a sequence of executable actions which achieves the goal[1]. Recent works (Kambhampati et al., 2024; Valmeekam et al., 2024b) have shown that LLMs are not good at such planning, and natural planning with LLMs offers a promising direction, aligning better with real-world tasks (Hao et al., 2023; Zhao et al., 2023; Wang et al., 2024c; Jiao et al., 2024). Thus, our focus is only on natural planning (Zheng et al., 2024) and its effect on downstream reasoning tasks (App. G for examples) where the formulation does not necessarily match with a classical planning setting.

In recent years, LLM agents have shown impressive abilities to solve complex reasoning problems (Yao et al., 2023; Xiao et al., 2024; Wang et al., 2024a). Orthogonal to this exploration, scaling a search space during inference-time (i.e., test-time scaling) (Snell et al., 2024; Welleck et al., 2024) has gained popularity in tackling difficult problems such as mathematical reasoning (Zhang et al., 2024) and code generation (Wang et al., 2025). Despite the success of these frameworks, we hypothesize that they often struggle with complex planning and reasoning due to the lack of better verification module, and a failure to account for instance-level complexity across single-task. Furthermore, although some initial explorations exist (Bohnet et al., 2024; Lee et al., 2025)[2], effectiveness of these frameworks for natural planning is under-explored. Motivated by these, we proposed PlanGEN, a model-agnostic, easily scalable, multi-agent framework for effective natural plan generation.

PlanGEN consists of three specialized agents: *constraint agent*, *verification agent*, and *selection agent*. The constraint agent extracts instance-

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

[1]Classical AI planning, which in early settings had inputs in specific formats (i.e., not in natural language), has high computational complexity (Bylander, 1994).

[2]Extended related work is presented in App. A

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 20652–20678
November 4-9, 2025

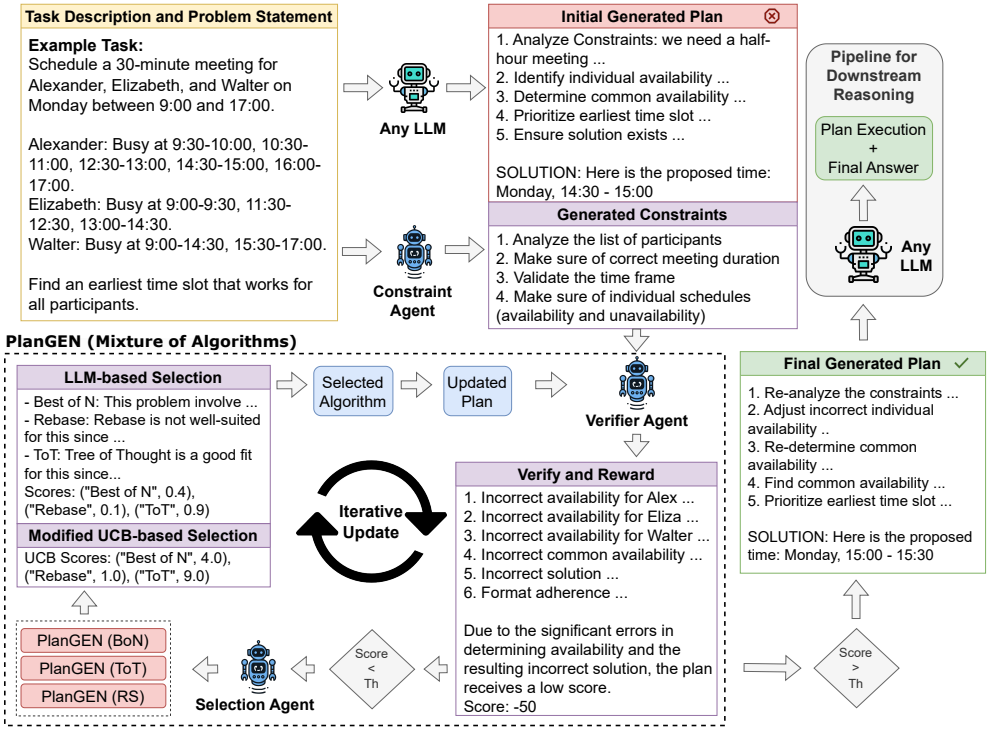

Figure 1: Schematic representation of PlanGEN (Mixture of Algorithms). An initial plan and constraints guide iterative plan refinement. The verification agent provides reward scores for plan quality, and the selection agent chooses inference algorithms until the highest-reward plan is found and used for downstream reasoning (if needed). UCB: Upper Confidence Bound, BoN: Best of $\mathcal{N}$, ToT: Tree-of-Thought, RS: REBASE.

specific constraints (e.g., budget, concepts, rules, etc.); the verification agent evaluates plan quality and provides a reward score considering the constraints; and the selection agent dynamically chooses the best inference algorithm using an improved Upper Confidence Bound (UCB) formula (Han et al., 2024) for instance of different complexity. We explore popular and widely used three inference algorithms within PlanGEN: Best of $\mathcal{N}$ (Brown et al., 2024), Tree-of-Thought (ToT) (Yao et al., 2024), and REward-BAlanced SEarch (REBASE) (Wu et al., 2024a). We combine our agents with these algorithms, yielding four frameworks: (1) PlanGEN (Best of $\mathcal{N}$), (2) PlanGEN (ToT), (3) PlanGEN (REBASE), and (4) PlanGEN (Mixture of Algorithms). In PlanGEN, "Multi-Agent" signifies using the constraint and verification agents for the first three approaches, and all three agents for the "Mixture of Algorithms" (Figure 1). Figure 1 shows example from NATURAL PLAN (Calendar scheduling), and App. G provides more examples.

We perform all experiments using Gemini-1.5-Pro (Team et al., 2024) as underlying model in PlanGEN. We further present case-study on Gemini-2.5-Pro, Gemini-2.0-Flash, and GPT-4o (Hurst et al., 2024) to show the model-agnostic nature. We evaluate natural planning ability on NATURAL PLAN (Zheng et al., 2024), scientific/mathematical reasoning on GPQA (Rein et al., 2024) and OlympiadBench (He et al., 2024), and financial reasoning on DocFinQA (Reddy et al., 2024). Performance is compared against Zero-shot Chain-of-Thought (CoT) and a multi-agent baseline. We achieve state-of-the-art results on NATURAL PLAN (∼8%↑ average across all categories), OlympiadBench (text-only) (∼5%↑ on MATH, ∼4%↑ on PHYSICS), and DocFinQA (∼7%↑). On GPQA, we outperform Gemini-1.5-Pro (∼13%↑), GPT-4o (∼12%↑), and Claude-3.5-Opus (∼9%↑), while achieving competitive performance compared to the multi-agent baseline (∼1%↑). The simplest method (i.e., PlanGEN (Best of $\mathcal{N}$)) achieves the best performance on NATURAL PLAN (Figure 5). PlanGEN (Mixture of Algorithms) achieves the best performance for complex problems (Figure 6) including GPQA, and OlympiadBench. PlanGEN's improvements target plan quality, rather than simply refining the final answer or solution. Our case study on QwQ-32B (Team, 2025) demonstrates PlanGEN's compatibility with recent open-source reasoning models. Our comparison with Buffer-of-Thought (BOT) (Yang

et al., 2024) show that PlanGEN achieves superior performance compared to reasoning-based prompting. Thorough analysis of the results which reveals several important findings. In summary, our contributions are: (1) PlanGEN, a novel, model-agnostic, and easily adaptable multi-agent framework for enhancing LLM natural planning; (2) state-of-the-art results on several complex planning and reasoning benchmarks; and (3) a novel approach to constraint-based verification and instance-level complexity-based inference algorithm selection.

2 PlanGEN

2.1 Proposed LLM Agents

PlanGEN comprises three specialized LLM agents: a *constraint agent*, a *verification agent*, and a *selection agent*. Each agent utilizes an off-the-shelf LLM (e.g., Gemini, GPT) which is equipped with task-specific prompts for efficient performance.

2.1.1 Constraint Agent

We define "constraints" as the instance-specific criteria necessary for verifying solutions to planning problems. For instance, in the calendar scheduling, relevant constraints include 'individual schedules', 'availabilities', and 'preferences'. In a scientific reasoning problems from GPQA, constraints might be the 'concepts used', 'calculation correctness', and 'formula selection'. We argue that careful extraction of these constraints is critical for successful verification. The constraint agent serves as a preprocessing component in the framework, designed to extract instance-specific constraints from the problem description. By analyzing the input problem, this agent identifies maximum possible constraints that are required for verifying generated plans to improve the overall relevance and quality of the planning process. The prompt used by the constraint agent enables it to systematically identify constraints by asking the underlying LLM to focus on specific aspects of the problem. This ensures that no critical information is overlooked and that the resulting constraints are comprehensive. While constraint agent is effective in capturing the majority of constraints, it does not offer a formal guarantee of exhaustiveness in constraint extraction. Prompts and examples for constraint agent are provided in App. C and App. G, respectively.

2.1.2 Verification Agent

The verification agent plays a critical role in the framework by assessing the quality of generated plans based on constraints. This agent ensures that plans are aligned with task objectives, adhere to constraints, and progress logically toward a correct and complete solution. This agent has two key components: (i) feedback generation, and (ii) numerical reward score generation based on feedback. Verification prompts and examples of verification are provided in App. C and App. G, respectively.

Feedback Generation While verifying each generated plan against different constraints, the verification agent generates detailed natural language reasoning regarding plan quality. We consider this explanation as "feedback", offering interpretability and actionable next step towards improvement.

Numerical Reward Generation Motivated by Zhang et al. (2024), we instruct the agent to evaluate the plan against various constraints and assign a reward score on a scale of -100 to 100. The scoring mechanism is designed to enforce strict quality standards, with a threshold (e.g., a score of 95 or higher) indicating a verified, high-quality plan.

2.1.3 Selection Agent

The selection agent dynamically determines the most suitable inference algorithm for solving a given problem instance based on its complexity. It leverages a combination of historical performance; diversity, and recovery scores; and guidance from a LLM to adaptively select the best algorithm (among three) for the given instance. To create the selection agent, we utilize a modified UCB policy. The policy combines multiple factors, including normalized rewards, exploration bonuses, diversity adjustments, and recovery scores. Additionally, the agent incorporates LLM-guided priors, which provide algorithm suitability scores based on the problem statement, task requirements, and previous plan (if available). These priors enable the agent to align its selections with the input instance complexity and corresponding constraints, improving the relevance of the chosen algorithm.

$$\begin{aligned}\mathrm{UCB}(a) = &\frac{R(a)}{N(a)} + \sqrt{\frac{2\log(T+1)}{N(a)}} \\ &+ \lambda_{\text{prior}} \cdot \mathrm{Prior}(a) \\ &+ \frac{\alpha_{\text{diversity}}}{N(a)+1} \\ &+ \alpha_{\text{recovery}} \cdot S_{\text{recovery}}(a)\end{aligned}$$

Modified UCB Policy equation combines several terms to balance exploitation, and exploration when selecting the best algorithm for given task instance.

To modify UCB, we first conducted a preliminary ablation study, presented in App. C. All terms in equation given above are calculated across one evaluation run. Here, the cost of calculation is negligible since it only utilizes reward values from previous runs, but only one LLM call require to get score for Prior(a). The first term, $\frac{R(a)}{N(a)}$, represents the average reward for algorithm a, where $R(a)$ is the total reward accumulated by the algorithm, and $N(a)$ is the number of times the algorithm has been selected. This term ensures that algorithms with higher historical performance are prioritized. The second term, $\sqrt{\frac{2\log(T+1)}{N(a)}}$, serves as the exploration component, encouraging the selection of algorithms with fewer trials, denoted as T. This term ensures that under-explored options are adequately evaluated. Furthermore, $\lambda_{\text{prior}} \cdot \text{Prior}(a)$, which leverages LLM-guided priors to align algorithm selection with the instance-specific complexity. Here, λ_{prior} is a dynamically decaying weight defined as $\frac{\lambda_{\text{prior}}}{1+T}$, where T represents the total number of trials. This decay gradually shifts the focus from initial priors to historical performance as trials progress. The diversity bonus, $\frac{\alpha_{\text{diversity}}}{N(a)+1}$, penalizes overused algorithms, ensuring balanced exploration across all options. Finally, the recovery term, $\alpha_{\text{recovery}} \cdot S_{\text{recovery}}(a)$, rewards algorithms that recover effectively from failures, with $S_{\text{recovery}}(a)$ representing the recovery score for algorithm a.

Selection Process This process begins by initializing algorithm-specific variables, such as accumulated rewards, selection counts, and failure counts. Further details on this can be found in Algorithm 1 (App. C). The agent then incorporates LLM-guided priors to generate algorithm suitability scores based on the problem statement and any provided feedback. These priors are derived from a LLM (prompt for this given in App. C), and serve as initial estimates to adjust the UCB (Han et al., 2024) values.

2.2 Proposed Frameworks

Within `PlanGEN`, we propose four different frameworks: (1) `PlanGEN` (Best of $\mathcal{N}$) (Figure 2), (2) `PlanGEN` (ToT) (Figure 3), and (3) `PlanGEN` (REBASE) (Figure 4), and (4) `PlanGEN` (Mixture of Algorithms) (Figure 1).

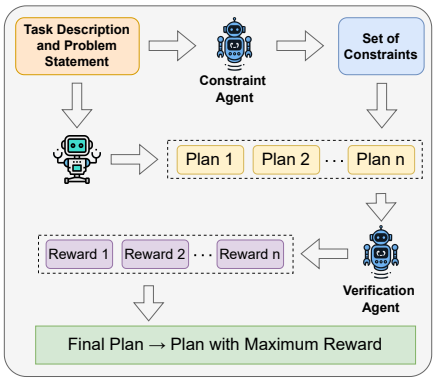

Figure 2: Schematic representation of `PlanGEN` (BoN).

2.2.1 `PlanGEN` (Best of $\mathcal{N}$)

Motivated by Brown et al. (2024), we adapted the Best of $\mathcal{N}$ algorithm and modified it using our constraint and verification agents as illustrated in Figure 2. The framework generates $\mathcal{N}$ candidate plans (Plan 1, Plan 2, ..., Plan n), and each plan is assessed by a verification agent based on a set of constraints. Then, a corresponding reward (Reward 1, Reward 2, ..., Reward n) gets assigned by the verification agent. Finally, the plan with the highest reward from generated candidates is selected, resulting a plan that aligns to the problem constraints.

2.2.2 `PlanGEN` (ToT)

ToT algorithm has been studied in detail for solving many complex problems (Yao et al., 2024). As shown in Figure 3, we modify the ToT algorithm with our constraint and verification agents. The method begins by initializing a root node that represents the problem and generating multiple potential next steps, creating a tree-like structure. The generated steps are verified using a verification agent which assigns reward scores based on a set of constraints. The iterative process involves evaluating all possible steps at a given depth, selecting the most promising path based on reward scores, and expanding it further by generating new steps. This process continues until a valid solution is identified or a pre-defined limit on iterations is reached. Further details on various prompts for the ToT are presented in App. D.

2.2.3 `PlanGEN` (REBASE)

The REBASE tree search method inherits the exploitation and pruning properties of tree search and is well-studied for mathematical reasoning (Wu et al., 2024a). As shown in Figure 4, the framework incorporates a dynamic selection and expansion strategy to iteratively refine solutions. At each

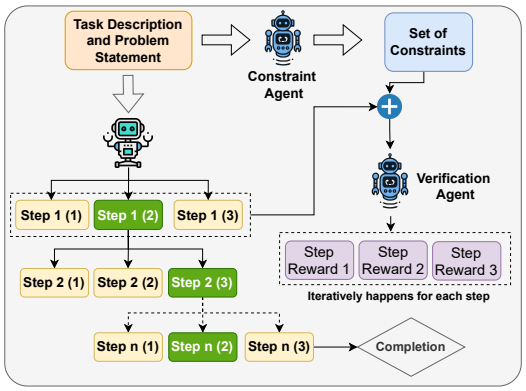

Figure 3: Schematic representation of PlanGEN (ToT). Highest-reward steps are highlighted in green.

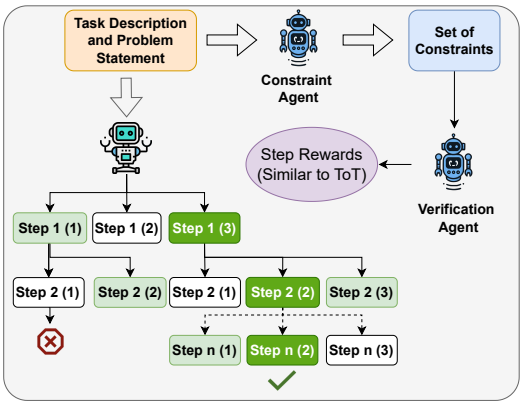

Figure 4: Schematic representation of PlanGEN (REBASE). Green shading indicates step reward (darker = higher). Darker steps prioritized for exploration.

depth of the tree, candidate nodes are ranked based on their assigned reward scores (obtained using a verification agent), ensuring that the most promising candidates are explored first. Even steps with lower rewards are considered but with a reducing number of children, meaning that their exploration depth is limited. This hierarchical pruning helps maintain efficiency, thereby reducing unnecessary exploration of weaker nodes. This process continues until either a valid, complete solution is found or a predefined depth or width limit is reached. Also, there is a completion check similar to ToT which identifies nodes that represent complete solutions, enabling REBASE to terminate early once a satisfactory outcome is identified. App. D provides further details on prompts for the REBASE.

2.2.4 PlanGEN (Mixture of Algorithms)

The Mixture of Algorithms framework (Figure 1) introduces a selection agent (§2.1.3) which dynamically selects the best possible inference-time algorithms proposed in the above sections based on instance-level complexity. The framework operates in a modular and iterative manner, ensuring adaptability in addressing planning and reasoning problems with different complexity effectively.

Orchestration The process begins with generating an initial plan using LLM based on the task description and problem statement. Along with this, the constraint agent (§2.1.1) is employed to generate an instance-specific set of constraints. Based on the constraints, the verification agent (§2.1.2) evaluates the quality of the initial plan and provides a reward score (indicated as 'Score' in Figure 1). If the initial plan meets the required threshold (denoted T_h), it is acceptable as the "Final Plan". Otherwise, the iterative refinement process begins.

Iterative Refinement The refinement loop is driven by a suite of inference algorithms as shown in Figure 1. During this iterative refinement, the selection agent (§2.1.3) determines the most suitable algorithm based on the instance-specific complexity and historical UCB values. The selected algorithm produces an updated plan, which is then re-evaluated by the verification agent. To ensure continual improvement, the framework incorporates feedback generated by a verification agent that provides guidance, and this feedback loop enables the system to refine the plan incrementally.

3 Experiments and Results

3.1 Experimental Setup

Datasets To demonstrate improvement in natural planning abilities, we utilize the NATURAL PLAN (Zheng et al., 2024). After improving the planning, we show that this significantly enhances the reasoning capabilities of LLMs on two benchmarks: GPQA (Rein et al., 2024) and OlympiadBench (text-only) (He et al., 2024). Additionally, we show that PlanGEN improves performance on a domain-specific dataset, DocFinQA (Reddy et al., 2024). Further details are presented in App. E.

Baselines and Our Frameworks We develop two baselines: (i) Zero-shot CoT (Kojima et al., 2024) and (ii) a Vanilla Multi-Agent Baseline. In the Zero-shot CoT, we provide an input prompt to the model, which generates outputs in the form of <CoT reasoning, Answer>. For the "Multi-Agent Baseline", the same model is called iteratively across multiple iterations. The system repeatedly refines its outputs through feedback loops, where the feedback is generated based on a self-

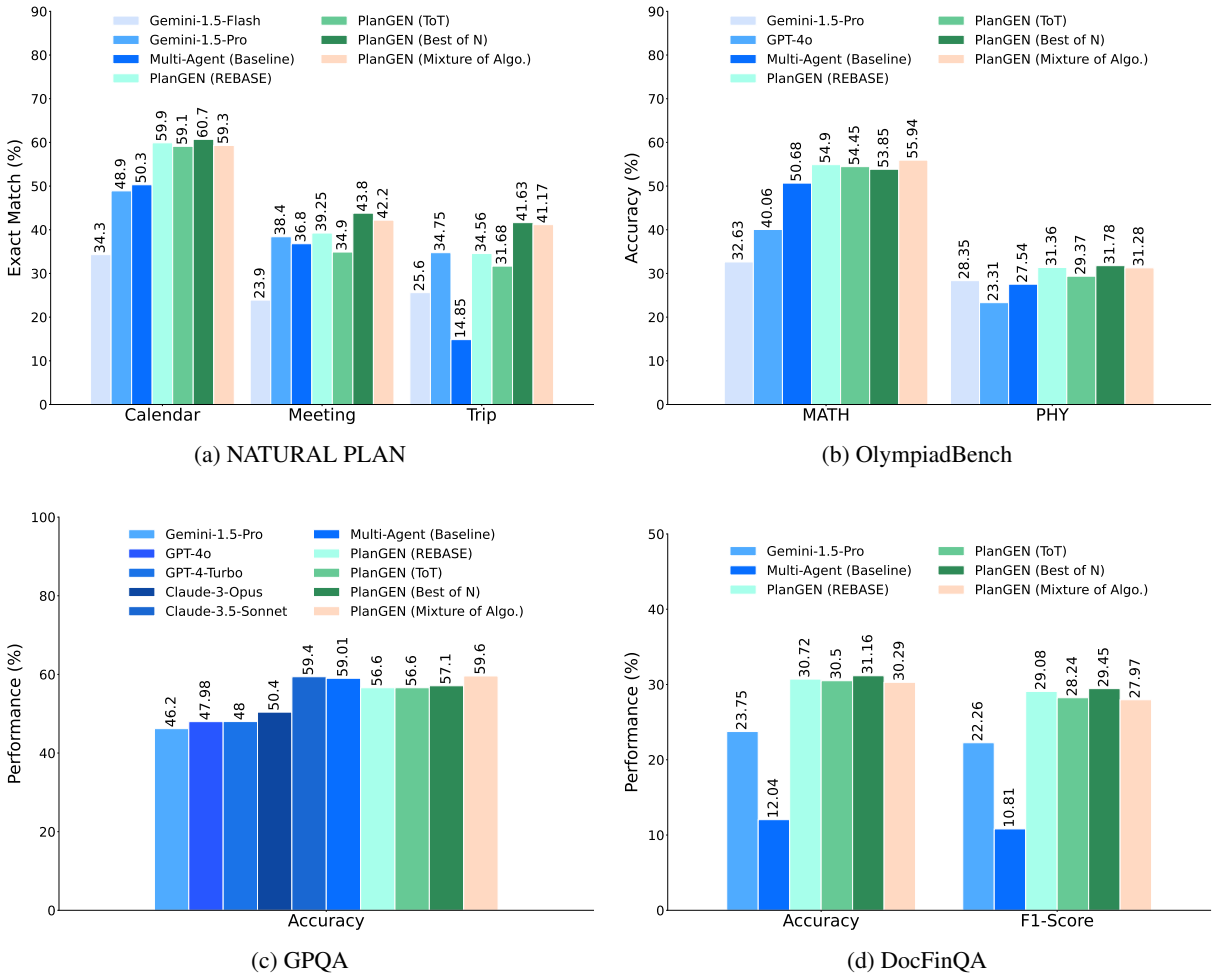

Figure 5: Performance comparison of the proposed multi-agent frameworks against baselines across four benchmarks. All experiments are conducted using Gemini-1.5-Pro. Algo: Algorithms, PHY: Physics.

reflective prompt (App. E) designed to improve reasoning. We evaluate all proposed frameworks (§2.2) on all benchmarks. For reasoning tasks, we use a two-stage approach: (1) generating plan using `PlanGEN`, and (2) executing the plan to produce the final answer (Figure 1). App. E presents further details on model selection, metrics, and experiment hyper-parameters including the hyper-parameter choices for inference-time algorithms.

3.2 Main Results

Figure 5 compares performance of `PlanGEN` frameworks across various baselines (varies across benchmarks - some single-agent baselines for GPQA are obtained from `https://klu.ai/glossary/gpqa-eval`), showing that multi-agent frameworks are consistently outperforming the baselines.

Performance on NATURAL PLAN From Figure 5a, `PlanGEN` (Best of $\mathcal{N}$) achieves the highest EM scores across all tasks: 60.70 (Calendar), 43.80 (Meeting), and 41.63 (Trip). In calendar scheduling, all four frameworks surpass the strongest baseline (Multi-Agent) by $\sim 10\%$. For meeting and trip planning, all except ToT outperform the best baseline (Gemini-1.5-Pro) by $\sim 6\%$ and $\sim 7\%$, respectively. `PlanGEN` (Mixture of Algo.) achieves the second-highest performance in meeting and trip planning while remains competitive in calendar scheduling. These results demonstrate the effectiveness of our frameworks in handling diverse natural language planning tasks and establishing SOTA for all three categories of NATURAL PLAN.

Performance on OlympiadBench From Figure 5b, `PlanGEN` (Mixture of Algo.) achieves the highest accuracy in the MATH (55.94%), outperforming the Multi-Agent Baseline (50.68%) by $\sim 5\%$. The superior performance of the `PlanGEN` in MATH highlights its effectiveness in complex mathematical reasoning, setting a SOTA for the MATH. In the PHY, all `PlanGEN` frameworks sur-

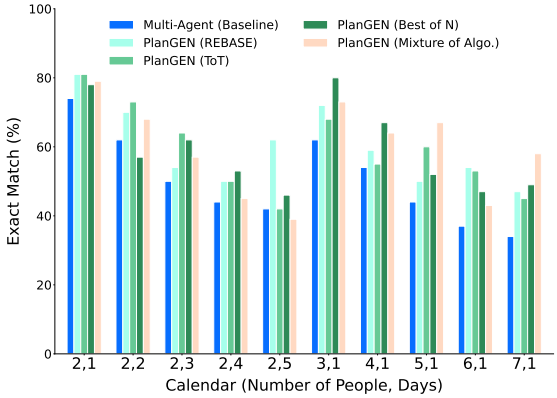

Figure 6: Performance comparison of inference-time algorithms across different complexity levels for calendar scheduling from NATURAL PLAN.

pass Gemini-1.5-Flash (strongest baseline), with PlanGEN (Best of $\mathcal{N}$) achieving the highest accuracy (31.78%), setting a SOTA for the PHY.

Performance on GPQA From Figure 5c, the PlanGEN (Mixture of Algo.) achieves the highest accuracy (59.6%). The individual inference-time algorithms achieve a lower performance, indicating the usefulness of selection. All proposed frameworks outperform Gemini-1.5-Pro (46.2%), GPT models ($\sim$ 48%), and Claude-3-Opus (50.4%) by a large margin. While Claude-3.5-Sonnet, and Multi-Agent Baseline perform competitively ($\sim$ 59%) compared to PlanGEN (Mixture of Algo.).

Performance on DocFinQA From Figure 5d, our frameworks significantly improve performance on DocFinQA, with PlanGEN (Best of $\mathcal{N}$) achieving the highest accuracy (31.16%) and F1-Score (29.45%), setting SOTA for the task. All our frameworks outperform the Gemini-1.5-Pro (strong baseline) by $\sim$ 7%. These results highlight the effectiveness of PlanGEN in performing financial reasoning.

Performance of PlanGEN w.r.t. different complexity As shown in Figure 6, we conduct a case study on calendar scheduling task from NATURAL PLAN to analyze the impact of varying complexity levels on the performance of different frameworks. For the calendar scheduling, we observe that PlanGEN (ToT) performs best for simple problems, while PlanGEN (Best of $\mathcal{N}$) is more effective for intermediate problems. As complexity increases, a PlanGEN (Mixture of Algo.) proves to be the most effective approach. We further conduct a similar analysis for meeting and trip planning from NATURAL PLAN presented in App. F.

Main Findings Compared to single-agent systems, PlanGEN consistently outperform in generating better planning trajectories (Figure 5). Furthermore, Multi-Agent (Baseline) is not always the strongest benchmark, as self-correction can introduce challenges as shown in Huang et al. (2024). Thus, different agents within the system require distinct handling strategies similar to our PlanGEN. Additionally, even in PlanGEN frameworks, relying on a single inference-time algorithm proves insufficient for more complex problems (Figure 6). A PlanGEN (Mixture of Algo.) approach offers substantial advantages for solving complex reasoning problems, highlighting the importance of algorithm selection based on instance-specific complexity (Figure 1). Given that our frameworks are multi-agent, we provide further discussion on latency (calls/time) *vs.* performance in App. F.

4 Analysis and Discussion

Importance of Verification Agent Figure 7 demonstrates the verification agent's crucial role in PlanGEN by showing a strong correlation between assigned reward values and prediction correctness (1 for correct, 0 for incorrect). The plotted points represent the average correctness rate for data buckets of varying reward values, each bucket containing hundreds of samples. A logistic regression model trained on DocFinQA and GPQA data ($\sim$ 1100 total samples) reveals a sigmoidal trend: higher rewards correlate with increased success probability, highlighting the agent's effectiveness. This reinforces the importance of constraint-guided verification for improving inference-time algorithms (see App. F for further details).

Importance of Selection Agent Figure 8 illustrates the importance of the selection agent by comparing the performance on the NATURAL PLAN. Here, Multi-Agent (Ver.) includes only the verification agent, while Multi-Agent (Ver. + Selection) further includes a selection agent. The results highlight the progressive impact of these components.

For example, in calendar scheduling, Multi-Agent (Ver.) improves performance to 56.1 EM compared to Multi-Agent (Baseline). However, Multi-Agent (Ver. + Selection) achieves 59.3 EM, demonstrating the additional benefit of algorithm selection. A similar trend is observed in trip planning where Multi-Agent (Ver. + Selection) outperforms Multi-Agent (Ver.) (41.17 EM vs. 35.44 EM) and the Multi-Agent (Baseline). For meeting

Methods	GPQA	NATURAL PLAN (Calendar)	OlympiadBench MATH	OlympiadBench PHY	Methods	GPQA	NATURAL PLAN (Calendar)	OlympiadBench MATH	OlympiadBench PHY	Methods	GPQA	Methods	GPQA
Gemini-1.5-Pro	46.20	48.90	32.63	28.35	Gemini-2.0-Flash	60.10	61.10	51.13	27.54	GPT-4o	47.98	Gemini-2.5-Pro	53.03
PlanGEN (BoN) Gemini-1.5-Pro	56.60	**60.70**	53.85	**31.78**	PlanGEN (BoN) Gemini-2.0-Flash	56.83	**68.90**	59.90	35.60	PlanGEN (BoN) GPT-4o	40.40	PlanGEN (BoN) Gemini-2.5	**77.27**
PlanGEN (ToT) Gemini-1.5-Pro	56.60	59.10	54.45	29.37	PlanGEN (ToT) Gemini-2.0-Flash	59.18	62.30	60.30	35.70	PlanGEN (ToT) GPT-4o	46.70	PlanGEN (ToT) Gemini-2.5-Pro	75.25
PlanGEN (REBASE) Gemini-1.5-Pro	57.10	59.90	54.90	31.36	PlanGEN (REBASE) Gemini-2.0-Flash	**64.14**	61.50	60.98	36.02	PlanGEN (REBASE) GPT-4o	41.40	PlanGEN (REBASE) Gemini-2.5-Pro	71.72
PlanGEN (MoA) Gemini-1.5-Pro	**59.60**	59.30	**55.94**	31.28	PlanGEN (MoA) Gemini-2.0-Flash	63.64	66.55	**64.10**	**37.29**	PlanGEN (MoA) GPT-4o	**49.40**	PlanGEN (MoA) Gemini-2.5-Pro	68.19

Table 1: Performance comparison for model-agnostic nature of PlanGEN. We utilize Gemini-1.5-Pro, Gemini-2.0-Flash, GPT-4o, and Gemini-2.5-Pro as baseline and underlying models in PlanGEN frameworks. Comparing methods that use the same base and underlying model for a fair assessment. MoA: Mixture of Algorithms.

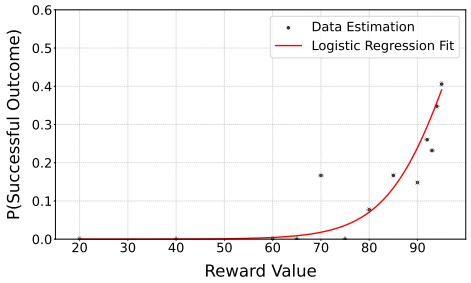

Figure 7: Logistic regression plot over the joint distribution showing verification agent's positive performance impact. P(Successful Outcome) = probability of prediction being correct.

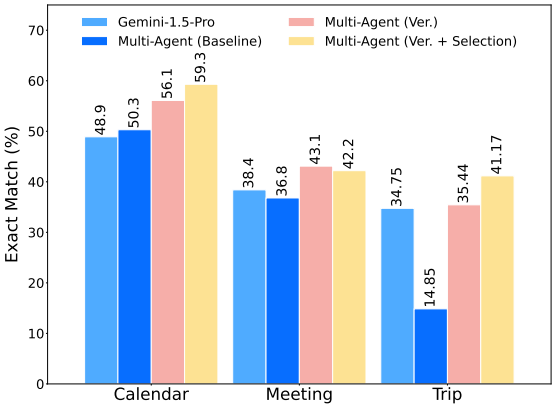

Figure 8: Case study on NATURAL PLAN, showing the impact of selection agent. Ver.: Verification

planning, Multi-Agent (Ver.) achieves 43.1 EM compared to 36.8 EM of Multi-Agent (Baseline), whereas, Multi-Agent (Ver. + Selection) achieves competitive performance. Together, verification and selection agents drive significant improvements over single-agent and multi-agent baselines.

Model-Agnostic Nature The results from Table 1 demonstrate the model-agnostic nature of PlanGEN frameworks. While the primary experiments were conducted using Gemini-1.5-Pro, the framework's effectiveness holds across different underlying models, such as Gemini-2.5-Pro, Gemini-2.0-Flash and GPT-4o. For instance, in the NATURAL PLAN (calendar scheduling), the PlanGEN (Best of $\mathcal{N}$) framework achieves a significant improvement, reaching 68.90 EM, outperforming Gemini-2.0-Flash (61.10 EM). Similarly, in OlympiadBench, the PlanGEN (Mixture of Algo.) achieves the highest scores in MATH (64.10) and PHY (37.29), surpassing Gemini-2.0-Flash (52.13 MATH, 27.54 PHY). Note that, the Mixture of Algo. outperforms other three frameworks, showing effectiveness of selection agent. On GPQA, Mixture of Algo. (49.40), PlanGEN (REBASE) (64.14), and PlanGEN (Best of $\mathcal{N}$) (77.27) outperform GPT-4o (47.98), Gemini-2.0-Flash (60.10), and Gemini-2.5-Pro (53.03), respectively. These results highlight that regardless of the underlying model, our frameworks consistently enhance performance by leveraging multi-agent collaboration.

Discussion on LLM calls *vs.* Performance (%) Figure 9 shows the relationship between the number of LLM calls and task performance across baselines (single-agent and multi-agent) and proposed frameworks, using OlympiadBench (MATH category). The single-agent system, zero-shot CoT, requires only one LLM call. The multi-agent baseline requires the same number of calls as PlanGEN (Best of $\mathcal{N}$), but our framework outperforms the multi-agent baseline. For PlanGEN (ToT) and PlanGEN (REBASE), we focus on LLM calls during the tree expansion phase. PlanGEN (ToT) involves dynamic exploration, where each explored path requires three LLM calls: step generation, reward evaluation, and completion verification. The total cost is the per-path cost multiplied by the number of paths explored, constrained by either the number of steps generated for each problem or a predefined iteration budget (i.e., 20). For PlanGEN (REBASE), the number of LLM calls depends on the search width (i.e., 10). Each solution path expansion in-

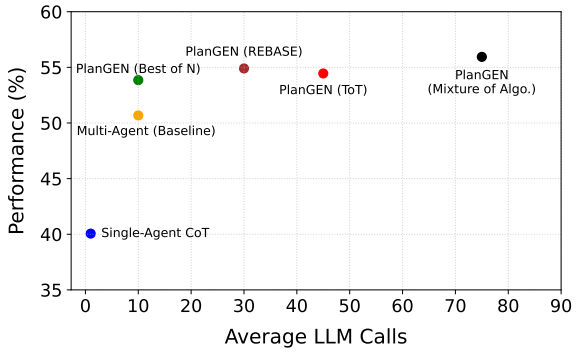

Figure 9: Comparison of baselines and our frameworks, showing the trade-off between LLM calls and performance (%) for OlympiadBench (MATH).

volves three calls: step generation, quality evaluation, and completion verification, thus, giving maximum 30 LLM calls for single problem. For PlanGEN (Mixture of Algo.), we estimate the average LLM calls by summing the estimated calls for each selected algorithm per problem, then dividing by the total number of problems. As shown in Figure 9, the single-agent system exhibits the lowest performance despite requiring just one LLM call. Multi-agent approaches show improved performance, with PlanGEN (ToT) and PlanGEN (REBASE) balancing LLM call efficiency and accuracy. The PlanGEN (Mixture of Algo.) method achieves the highest performance, suggesting that combining diverse planning strategies enhances efficiency.

PlanGEN with Recent Reasoning Models We conducted a case study using the latest QwQ-32B model (Team, 2025) on GPQA. In a zero-shot setting, it achieves 65.15% accuracy. When plugged into our PlanGEN frameworks, its performance further improves—reaching 68.01% under PlanGEN (Best of $\mathcal{N}$), 70.00% under PlanGEN (ToT), and 60.71% under PlanGEN (REBASE)—demonstrating that PlanGEN consistently boosts even state-of-the-art reasoning models. Hence, we can say that PlanGEN is model-agnostic (also discussed in Table 1) and compatible with recent reasoning models.

PlanGEN vs. BOT We incorporate Buffer-of-Thoughts (BOT) (Yang et al., 2024)—a recent reasoning method—into our evaluation. We implement BOT using Gemini-1.5-Pro (to match our PlanGEN setup), we measured accuracy across four benchmarks. BOT scores 55.50% on GPQA, 52.96% on MATH, and 27.11% on PHY. As shown in Figure 5, all of our PlanGEN variants outperform BOT by $\sim$ 4% on GPQA and OlympiadBench. These results show that PlanGEN remains superior to state-of-the-art single-agent reasoning methods.

Dataset	Baseline	Best Framework
GPQA	58.08%	**59.60%**
OlympiadBench(PHY)	25.85%	**31.78%**

Table 2: Comparison between a constraints-based CoT baseline and best-performing PlanGEN framework.

Constraints-based CoT *vs.* PlanGEN This baseline first prompts to generate constraints, then verifies them, and finally revises the final solution, *without* employing independent agents for these stages, similar to PlanGEN. We evaluated this baseline on two datasets, GPQA and OlympiadBench (PHY). For fair comparison we used the same model as in our main results (Gemini-1.5-Pro; Figure 5). As shown in Table 2, our full framework, which assigns dedicated agents for constraint extraction, verification, and strategy selection, consistently outperforms baseline on both benchmarks. These results indicate that explicit modularization of planning steps is more effective than a monolithic iterative revision approach. For creating baseline, prompts for each stage are used as shown in App. C.

Additional Analysis We provide further analysis on PlanGEN performance on solving AI planning in natural language such as blocksworld (Valmeekam et al., 2024a), selection of parameters for PlanGEN, random baseline, and many more in App. F.

5 Conclusions

In this work, we proposed PlanGEN, an easily scalable multi-agent approach incorporating three key components: constraint, verification, and selection agents. We leveraged these agents to improve the verification process of existing inference algorithms and proposed three frameworks: Multi-Agent Best of $\mathcal{N}$, ToT, and REBASE. Further, we introduced a Mixture of Algorithms, an iterative framework that integrates the selection agent (Figure 1) to dynamically choose the best algorithm. We evaluated our frameworks on NATURAL PLAN, OlympiadBench, GPQA, and DocFinQA. Experimental results demonstrate that PlanGEN outperforms strong baselines, achieving SOTA results across datasets. Furthermore, our findings suggest that the proposed frameworks are scalable and generalizable to different LLMs, improving their natural language planning ability.

Limitations

Despite the strong performance of our frameworks, an area of improvement is the reliance on predefined heuristics for selecting inference-time algorithms, which may not always generalize optimally across all tasks and domains. Additionally, while our frameworks demonstrate strong performance, their computational overhead could be further optimized for efficiency. In addition, the use of reinforcement learning or meta-learning techniques to dynamically adapt agent strategies based on task complexity could be an interesting area to explore. Moreover, broadening the scope to multi-modal and multi-lingual reasoning would significantly expand the applicability of our approach, and exploring the use of generated planning trajectories for model training offers valuable direction.

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

Bernd Bohnet, Azade Nova, Aaron T Parisi, Kevin Swersky, Katayoon Goshvadi, Hanjun Dai, Dale Schuurmans, Noah Fiedel, and Hanie Sedghi. 2024. Exploring and benchmarking the planning capabilities of large language models. *arXiv preprint arXiv:2406.13094*.

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*.

Tom Bylander. 1994. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204.

Weizhe Chen, Sven Koenig, and Bistra Dilkina. 2024. Reprompt: Planning by automatic prompt engineering for large language models agents. *arXiv preprint arXiv:2406.11132*.

Qiyang Han, Koulik Khamaru, and Cun-Hui Zhang. 2024. Ucb algorithms for multi-armed bandits: Precise regret and adaptive inference. *arXiv preprint arXiv:2412.06126*.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore. Association for Computational Linguistics.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 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, Bangkok, Thailand. Association for Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

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.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. 2024. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2024. Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, Dave Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. 2025. Evolving deeper llm thinking. *arXiv preprint arXiv:2501.09891*.

Yanming Liu, Xinyue Peng, Yuwei Zhang, Jiannan Cao, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. 2024. Tool-planner: Dynamic solution tree planning for large language model with tool clustering. *arXiv preprint arXiv:2406.03807*.

Varshini Reddy, Rik Koncel-Kedziorski, Viet Dac Lai, Michael Krumdick, Charles Lovering, and Chris Tanner. 2024. DocFinQA: A long-context financial reasoning dataset. In *Proceedings of the 62nd Annual*

Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 445–458, Bangkok, Thailand. Association for Computational Linguistics.

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*.

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*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2024a. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. *Advances in Neural Information Processing Systems*, 36.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. 2024b. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench. *arXiv preprint arXiv:2409.13373*.

Evan Z Wang, Federico Cassano, Catherine Wu, Yunfeng Bai, William Song, Vaskar Nath, Ziwen Han, Sean M. Hendryx, Summer Yue, and Hugh Zhang. 2025. Planning in natural language improves LLM search for code generation. In *The Thirteenth International Conference on Learning Representations*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024a. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024b. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*.

Yulong Wang, Tianhao Shen, Lifeng Liu, and Jian Xie. 2024c. Sibyl: Simple yet effective agent framework for complex real-world reasoning. *CoRR*.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. 2024. From decoding to meta-generation: Inference-time algorithms for large language models. *arXiv preprint arXiv:2406.16838*.

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

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. 2024b. Os-copilot: Towards generalist computer agents with self-improvement. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. 2024. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin CUI. 2024. Buffer of thoughts: Thought-augmented reasoning with large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

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

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

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

Hongyu Zhao, Kangrui Wang, Mo Yu, and Hongyuan Mei. 2023. Explicit planning helps language models in logical reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11155–11173, Singapore. Association for Computational Linguistics.

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

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Ningyu Zhang, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. 2024. Knowagent: Knowledge-augmented planning for llm-based agents. *arXiv preprint arXiv:2403.03101*.

A Related Works

LLM Agents for Planning Agent-based frameworks for planning have gained interest, focusing on enhancing how LLMs decompose tasks and refine their outputs. The Sibyl framework (Wang et al., 2024c) effectively decomposes tasks into smaller subtasks, assigning each to specialized agents that iteratively collaborate until a solution is reached. OS-Copilot (Wu et al., 2024b) introduces a generalist computer agent that employs self-improvement through modularization and feedback loops. Another approach is KnowAgent (Zhu et al., 2024), which integrates knowledge-augmented planning to enhance the decision-making capabilities of LLM agents. Similarly, Tool-Planner (Liu et al., 2024) proposed grouping tools based on similar functionalities into toolkits, allowing LLMs to select the best tool for a given task. Many agent-based works focusing on planning have been developed (Chen et al., 2024; Wang et al., 2024b). Despite the progress, these methods generally (i) focus on domain-specific tasks or limited benchmarks, reducing generalizability, and (ii) lack or under-explore mechanisms for verifying and refining plans iteratively. While some works explore natural language planning (Bohnet et al., 2024; Lee et al., 2025), they either single-agent frameworks or evaluate proposed framework on domain-specific benchmarks.

Inference-time Algorithms Inference-time algorithms have recently shown a significant improvement in LLMs performance during inference. For instance, Best of $\mathcal{N}$ sampling (Brown et al., 2024) selects the most promising output from multiple generations performed using temperature sampling, while Tree-of-Thought (ToT) (Yao et al., 2024) models reasoning as an iterative tree search. REBASE (Wu et al., 2024a) optimizes search-space pruning using reward balancing. One very popular approach is Monte Carlo Tree Search (MCTS) (Zhang et al., 2024) which iteratively explores solution paths during inference. Applied to models such as LLaMa-3-8B, it enables self-refinement by revisiting and improving initial solutions. Test-time optimization (Snell et al., 2024), focuses on dynamically adjusting computational resources during inference (Wu et al., 2024a). Furthermore, Wang et al. (2025) uses the inference time algorithms to improve LLMs planning capabilities to solve code synthesis problems. In inference-time algorithms, verification is the key component. In contrast to these past works, here, we enhance performance of inference-time algorithms utilizing constraint-guided verification, and multi-agent collaboration for natural language planning, and its applications in downstream complex reasoning tasks.

B Natural Planning Prompt Examples

To show open-ended, "everyday" plans from LLMs, we show below two prompt templates and their ChatGPT-4o (checkpoint: May 2025) completions.

B.1 Prompt: "Let us make a plan to . . . "

- Let us make a plan to travel the world on a budget.
- Let us make a plan to surprise our parents with a visit.
- Let us make a plan to start our own small business.
- Let us make a plan to eat healthier without giving up flavor.
- Let us make a plan to finish that project we started months ago.
- Let us make a plan to learn a new language together.
- Let us make a plan to volunteer in our local community.
- Let us make a plan to save enough money for a down payment.
- Let us make a plan to reconnect with old friends.
- Let us make a plan to spend more quality time offline.

B.2 Prompt: "Let us make a plan for . . . "

- Let us make a plan for our next big adventure.
- Let us make a plan for retirement that actually excites us.
- Let us make a plan for the weekend so it doesn't slip away.
- Let us make a plan for dealing with unexpected emergencies.
- Let us make a plan for hosting the perfect dinner party.
- Let us make a plan for our child's education journey.
- Let us make a plan for how to reach our fitness goals.
- Let us make a plan for moving into our dream home.
- Let us make a plan for celebrating our anniversary in style.

C Further Details on LLM Agents

In this section, we provide additional details about each specialized agent in PlanGEN. We present the prompts used for each agent, highlighting their roles in the framework. The prompt for the constraint agent includes task-specific parameters that can be adjusted to extract relevant constraints for different tasks. In contrast, the prompts for the verification agent and selection agent are entirely task-agnostic, ensuring generalizability and adaptability across various problem domains.

Prompts for Constraint Agent The constraint agent is responsible for extracting problem-specific constraints that guide the planning process. To enable systematic extraction of constraints, we design a task-specific prompt for the constraint agent:

Prompt

You are an expert in understanding an input problem and generating set of constraints. Analyze the input problem and extract all relevant instance-specific constraints and contextual details necessary for accurate and feasible planning.

(Optional) These constraints may include:

<You may provide any specific type of constraints>

<You may provide any formatting instruction>

Input Problem: <problem statement>

Based on the above prompts, we define the types of constraints used in the NATURAL PLAN benchmark for different planning tasks: calendar scheduling, meeting planning, and trip planning. For DocFinQA, we provide a set of formatting instructions to ensure structured constraint generation. For GPQA and OlympiadBench, the constraint extraction follows the general prompt outlined above.

Prompts for Verification Agent The prompt for the verification agent is designed to be task-agnostic, meaning it can be applied across different problem domains without modification. By enforcing strict evaluation criteria, this agent enhances the reliability of PlanGEN, making it robust for various planning and reasoning tasks. In this prompt, list of constraints are generated using constraint agent. Notably, the list of constraints used in the verification prompt is dynamically generated by the constraint agent. This ensures that the verification process is based on instance-specific constraints rather than relying on predefined, static rules.

Prompt

Provide a reward score between -100 and 100 for the quality of the provided plan steps, using strict evaluation standards. Ensure the reward reflects how effectively the plan contributes to progressing toward the correct solution.
Problem Statement:
{problem}
Plan:
{plan}
Consider the following constraints while evaluating:
- [Constraint 1]
- [Constraint 2]
- [Constraint 3]
Provide feedback in the following format:
[Step-by-step reasoning for the reward score]
Score: [Strictly provide an integer reward score between -100 and 100]

Prompts for Selection Agent The prompt for the Selection Agent is task-agnostic, allowing it to be applied across various domains without modification. It processes feedback from the verification agent and contextual information from the problem statement to assign suitability scores to different inference-time algorithms.

Prompt

Analyze the following planning problem and explain your reasoning for assigning priority scores to the algorithms based on their suitability. Scores should be between 0 and 1, where 1 represents the most suitable algorithm for the given problem.
Problem Statement: <problem statement>
Requirements: <feedback>
Context: <context if context else 'None provided'>

Start by providing a brief reasoning for each algorithm's suitability based on problem complexity. Then, **ONLY output your response strictly as a list** with the exact format below:
Reasoning:

- **Best of N:** [Explain why this algorithm is or isn't suitable]
- **Rebase:** [Explain why this algorithm is or isn't suitable]
- **ToT:** [Explain why this algorithm is or isn't suitable]

Scores:

[("Best of N", float),
("Rebase", float),
("ToT", float)]

Algorithm for Selection using UCB The algorithm (Algorithm 1) presented is a modified UCB selection strategy that incorporates additional factors for exploration, diversity, and recovery. It initializes each algorithm with basic statistics like reward ($R(a)$), count of trials ($C(a)$), and recovery score ($Rec(a)$). The algorithm computes a normalized reward $\bar{R}_{\text{norm}}(a)$ for each option, balancing the reward with exploration ($E(a)$), which encourages trying less-used algorithms. A diversity bonus $D(a)$ penalizes overused algorithms, while a recovery bonus $RecB(a)$ rewards algorithms that perform well after prior failures. LLM-guided priors (LLM_prior) are used to influence the selection process based on prior knowledge. The final selection is made by maximizing the UCB score, which combines these factors to balance exploitation and exploration.

Ablation Study on UCB Modifications To design our selection agent, we conducted an ablation study evaluating modifications to the UCB formula, shown in Figure 10. Initially, we replaced the selection agent with a simple sequential strategy, termed "Multi-Agent (Sequential)", where algorithms execute in sequence, and the verification agent selects the highest-scoring plan. Next, we implemented a UCB selection agent, but excluded the 'diversity bonus' and 'recovery term' introduced in our proposed formulation in the main paper, denoted as "Multi-Agent (UCB w/o div. and rec.)". Finally, we

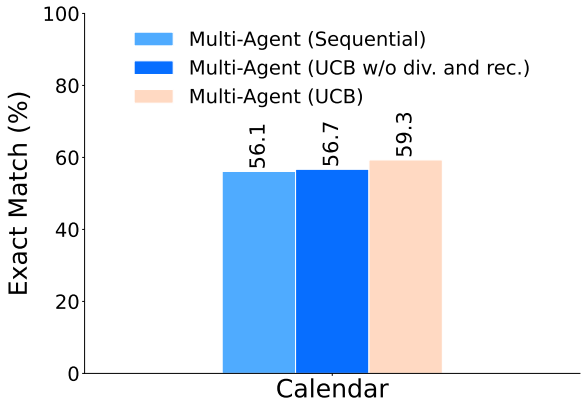

Figure 10: Ablation Study of UCB Modifications on Selection Agent and its impact on Multi-Agent Mixture of Algorithms framework. div.: diversity bonus, rec: recovery term.

implemented the complete selection agent incorporating our proposed UCB, labeled "Multi-Agent (UCB)". As shown in Figure 10, the inclusion of the diversity bonus and recovery terms in the UCB formula ("Multi-Agent (UCB)") resulted in $\sim 3.5\%$ performance gain compared to the UCB variant without these terms, further enhancing overall results. Note that the LLM-guided priors are still the part of Multi-Agent (UCB w/o div. and rec.) and Multi-Agent (UCB).

D Details on Proposed Frameworks

We provide further details in this section regarding the prompts used for `PlanGEN` (ToT) and `PlanGEN` (REBASE), as well as the specific algorithms used to execute these inference-time methods.

Prompts used for ToT and REBASE `PlanGEN` (ToT) and `PlanGEN` (REBASE) employ three prompt types: (1) step prompt, (2) step reward prompt, and (3) completion prompt. Step prompt guide the model to generate subsequent steps based on the problem statement and previously generated steps. Step reward prompt evaluate each intermediate step against the problem statement and constraints, similar to the prompts used by a verification agent. Completion prompt check for a complete solution after each step. If a solution is found, exploration terminates; otherwise, the process continues until a solution is reached.

Step Prompt

You are an expert assistant for generating step-by-step plan to solve a given question using specified tools. Given the problem and any intermediate steps, output only the next step in the plan. Ensure that the next action helps in moving toward the correct plan to solve the given question. Do not provide the full plan. Keep responses concise, focusing solely on the immediate next step that is most effective in progressing toward the correct plan.

<problem>
{Add a problem statement here}
</problem>

<intermediate_step>
{Append previously generated steps}
</intermediate_step>

Completion Prompt

You are an assistant tasked with verifying if the final, complete plan to solve the given question has been achieved within the intermediate steps. Output only '1' if the intermediate steps contain the full solution needed to solve the question. If the full plan has not yet been reached, output only '0'. Provide no additional commentary—return exclusively '1' or '0'.

<problem>
{Add a problem statement here}
</problem>

<intermediate_step>
{Append previously generated steps}
</intermediate_step>

E Further Details on Benchmarks and Experiments

Statistics of Benchmarks For evaluation, we utilize evaluation sets of all four benchmarks. For NATURAL PLAN, we employed the provided evaluation sets, consisting of 1000 instances each for Calendar Scheduling and Meeting Planning, and 1600 instances for Trip Planning. The GPQA evaluation was conducted using the Diamond set, which comprises 198 highly challenging instances. From OlympiadBench, we selected the text-only problems, excluding those requiring a theorem prover, resulting in 674 instances for the MATH category and 236 for the PHY category. We also used 922 instances from the DocFinQA evaluation set.

Models Our primary evaluations use Gemini-1.5-Pro for all the experiments. We also present a case study with Gemini-2.0-Flash, Gemini-2.5-Pro, GPT-4o, and QwQ-32B (recent reasoning model) to showcase the model-agnostic nature and generalizability of `PlanGEN`. For all models, we utilize a checkpoint from January 2025.

Metrics We use task-specific metrics for all evaluations. Specifically, we use Exact Match (EM) for NATURAL PLAN similar to Zheng et al. (2024), micro-average accuracy for OlympiadBench similar to He et al. (2024), and accuracy for GPQA and DocFinQA (along with F1-Score for DocFinQA).

Feedback prompt for Multi-Agent Baseline In the multi-agent baseline, we employ a feedback prompt to iteratively generate improved and refined outputs. The prompt is provided below:

Feedback Prompt

Analyze the following planning problem and explain your reasoning for assigning priority scores You are an intelligent assistant capable of self-reflection and refinement. I will provide you with your last response, and your task is to improve it, if necessary.
Here is your previous response:
{previous_response}
Analyze and refine your response step-by-step:
1. Reflect on your reasoning process. Where might it be unclear or incorrect? Improve it.
2. Revise the explanation to address any identified issues and make it more logical and comprehensive.
3. Ensure the final answer is correct, supported by clear reasoning.

Hyper-parameters for Experiments To ensure deterministic behavior, we set the temperature of all models to 0 for each agent. For the inference-time algorithms, we used the following settings: `PlanGEN` (Best of $\mathcal{N}$) with five samples at a temperature of 0.7; Tree of Thoughts (ToT) with three

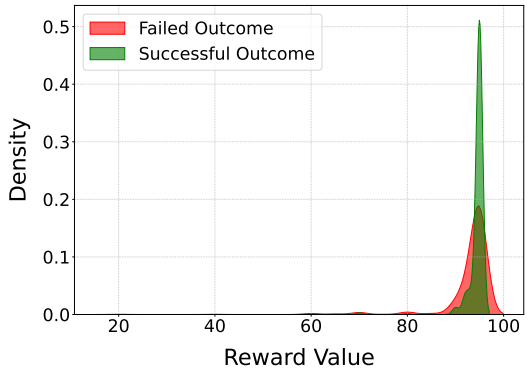

Figure 11: KDE plot illustrating the relationship between reward value and outcome (success/failure)

Methods	OlympiadBench	
	MATH	PHY
PlanGEN (Best of $\mathcal{N}$) (5)	53.26	32.63
PlanGEN (Best of $\mathcal{N}$) (10)	54.90	31.36
PlanGEN (Best of $\mathcal{N}$) (20)	53.22	29.38
PlanGEN (ToT) (3)	52.97	31.36
PlanGEN (ToT) (5)	55.20	32.05
PlanGEN (ToT) (10)	55.79	32.52
PlanGEN (REBASE) (10)	54.45	31.78
PlanGEN (REBASE) (20)	54.45	29.37
PlanGEN (REBASE) (30)	55.04	30.28

Table 3: Performance impact of hyper-parameters on inference-time algorithms in OlympiadBench

children per root node, generated at a temperature of 0.7; and REBASE, initialized with width 10 at temperature of 0.7, decremented by 1 after each call to expand. For our experiments, we set λ_{prior} = 10, $\alpha_{diversity}$ = 1, $\alpha_{recovery}$ = 1, and capped the exploration term at M = 5 (retaining the standard 2 under the square root). All these hyperparameters are selected based on empirical study.

F More Analysis

Performance of our frameworks w.r.t. different complexity From Figure 12, in the meeting planning, PlanGEN (Best of $\mathcal{N}$) excels in both simple and intermediate problems, whereas a PlanGEN (Mixture of Algo.) performs better for complex problems. The trip planning presents a different trend, where PlanGEN (Best of $\mathcal{N}$) and a PlanGEN (Mixture of Algo.) consistently outperform other approaches across all complexity levels. Nonetheless, in very complex problems for meeting and trip planning, all algorithms exhibit poor performance.

Importance of Verification Agent The kernel density estimation (KDE) plot visualizes the distribution of reward values assigned to two distinct outcomes: "Success" (green) and "Failure" (red). The plot reveals a clear separation between the reward distributions, with "Success" outcomes strongly associated with high reward values (around 80-100) and "Failure" outcomes primarily associated with low reward values (around 20-40). The sharply peaked green curve suggests consistent and high rewards for successful outcomes, while the broader red curve reflects more variability in rewards assigned to failures. However, a small bump in the red curve at high reward values (around 80-90) suggests a few instances where failures received unexpectedly high rewards, warranting further investigation. This observation is further supported by a statistically significant difference between the reward distributions, a Mann-Whitney U test ($U = 116128.0$, $p < 0.0001$). The low p-value (3.42e-09) provides evidence that the difference in reward distributions is statistically significant.

Frameworks	NATURAL PLAN	OlympiadBench	GPQA	DocFinQA
PlanGEN (BoN)	19.55%	7.09%	8.56%	81.03%
PlanGEN (ToT)	68.85%	90.09%	85.59%	12.5%
PlanGEN (REBASE)	11.6%	2.82%	5.86%	6.47%

Table 4: Algorithm Selection Frequency by Dataset

Different hyper-parameters of inference-time algorithms vs. their performance We conduct a case study on OlympiadBench, where we analyze the impact of varying hyper-parameters on the performance of different inference-time algorithms. The results (Table 3) indicate that while increasing the number of samples (Best of $\mathcal{N}$), steps (ToT), or refinements (REBASE) lead to marginal improvements, the overall differences remain relatively small. Given this, we opted for lower hyper-parameter values across all inference-time algorithms to balance efficiency and performance.

Frequency of inference-time algorithm selection across datasets For the PlanGEN (Mixture of Algo.) method, we analyze how frequently each inference-time algorithm (Best of $\mathcal{N}$, ToT, and REBASE) is selected across different datasets. The results (shown in Table 4) show that PlanGEN (ToT) is the most frequently chosen algorithm in NATURAL PLAN, OlympiadBench, and GPQA, indicating its effectiveness in these domains. In contrast, for DocFinQA, PlanGEN (Best of $\mathcal{N}$) is the dominant choice, suggesting that its strategy aligns better with financial reasoning tasks. PlanGEN (REBASE) is selected the least across all datasets, implying that its refinements are less favored by the

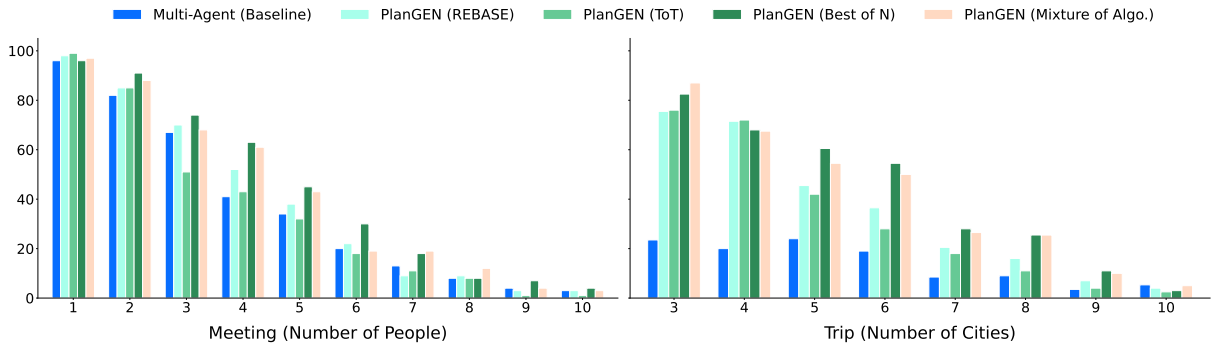

Figure 12: Performance comparison of inference-time algorithms across different complexity levels for meeting and trip planning from NATURAL PLAN

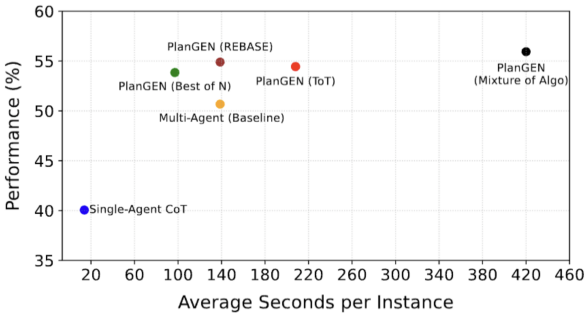

Figure 13: Comparison of baselines and our frameworks, showing the trade-off between time (seconds) and performance (%) for OlympiadBench (MATH).

selection mechanism. These findings highlight the dataset-dependent nature of inference-time algorithm effectiveness and the adaptability of the mixture approach in dynamically choosing the most suitable method.

Discussion on Time Complexity *vs.* Performance (%) Figure 13 shows the relationship between the time for running per instance and task performance across baselines (single-agent and multi-agent) and proposed frameworks, using OlympiadBench (MATH category). Since the algorithms used with the `PlanGEN` framework are dynamic and to abstract from transient issues like the existing quota limit and retrying, we calculate the average seconds per instance based on the latency of a single call (denoted as latency_of_single_call). For example, for a method that supports running LLM calls in parallel, the total time for 10 calls would be approximately 1 × latency_of_single_call. In contrast, for methods that process calls sequentially, the total time would be closer to 10 × latency_of_single_call.

Method	Blocksworld (v1)	Blocksworld (v3)
Gemini-2.5-Flash	88.40%	92.93%
PlanGEN (BoN)	85.40%	95.96%
PlanGEN (REBASE)	**95.54%**	**98.98%**

Table 5: Accuracy on blocksworld v1 and v3 across different methods for plan generation task.

Direct Solution *vs.* Plan Generation We conducted a case–study experiment on GPQA where we use Best-of-$\mathcal{N}$ with the same computational setting as `PlanGEN` (Best-of-$\mathcal{N}$), but rather than generating a plan, we generate a direct final CoT solution. The table below presents the results. From the results, we can observe that there is a $\sim 2\%$ performance improvement using `PlanGEN` (Best-of-$\mathcal{N}$) (57.10%) compared to Best-of-$\mathcal{N}$ (Baseline) (55.50%). Here, the computational cost of using Best-of-$\mathcal{N}$ is similar, despite `PlanGEN` (Best-of-$\mathcal{N}$) showing a performance gain. For ToT and REBASE, without gold step-by-step solutions to guide node selection, standalone ToT and REBASE become challenging to apply fairly to benchmarks.

Performance of `PlanGEN` on Blocksworld (Plan Generation) We evaluated our approach on the *Blocksworld* domain (versions v1 and v3) for the *Plan Generation* category from Valmeekam et al. (2024a). In PlanBench, the Plan Generation task requires generating a valid sequence of grounded actions that transforms a given initial state into a specified goal state while satisfying all domain constraints. For experiments, we utilize default prompts from Valmeekam et al. (2024a). The results, summarized in Table 5, compare a strong LLM baseline (Gemini-2.5-Flash) with two variants of our framework: `PlanGEN`(Best of $\mathcal{N}$) and `PlanGEN`(REBASE). As shown in Ta-

ble 5, PlanGEN(REBASE)[3] achieves the highest accuracy across both Blocksworld versions, outperforming Gemini-2.5-Flash by about 7.1% points on v1 and 6.0% points on v3. Even the simpler PlanGEN(Best of $\mathcal{N}$) variant surpasses Gemini-2.5-Flash on the more challenging v3, highlighting the value of diverse plan exploration. These results demonstrate that PlanGEN's structured, multi-agent approach—combining diverse plan sampling with iterative verification and constraint enforcement—substantially mitigates the weaknesses of standard LLMs identified by PlanBench and yields robust improvements.

Constraint-Guided Verification We conducted targeted experiments on GPQA and OlympiadBench(PHY). In this case study, we compared a Best of $\mathcal{N}$ baseline, where the constraint agent was removed and a generated plan was provided directly to the verifier agent, against our PlanGEN (Best of $\mathcal{N}$) variant with constraint-guided verification. The results show that on GPQA accuracy improves from 55.56% to 57.10%, and on OlympiadBench(PHY) from 29.08% to 31.78%. These $\sim$ 2% gains demonstrate that grounding the verifier in problem-specific constraints leads to higher-quality plans.

Discussion on Random Baseline we conducted an ablation on GPQA with the Gemini-2.0-Flash model to compare our adaptive selection module against a random-selection baseline. To develop a random-selection baseline, at each iteration, we randomly choose among Best of $\mathcal{N}$, ToT, and REBASE under the same stopping threshold, yielding 50.51% accuracy. In contrast, PlanGEN (Mixture of Algorithms) achieves 63.64% accuracy, where the selection agent selects the inference-time algorithm. This result demonstrates that informed, adaptive algorithm choice at test time significantly outperforms a random baseline.

G Various Examples for Different Components of PlanGEN

Examples for Constraint Agent To illustrate the output of our constraint agent, Table 6, Table 7, and Table 8 present representative examples of generated constraints. These tables highlight the diverse constraints generated for problem instances of different tasks.

[3]Results on REBASE are calculated for only 315 samples for v1 due to resource constraints, whereas we utilize all 500 samples for v1 and 100 samples for v3.

Example for Verification Agent To illustrate the output of our verification agent, Table 9 presents representative examples of verification process for NATURAL PLAN (calendar scheduling). This table highlights the how the verification agent verifies the generated plan using constraints.

Examples of Generated Plans To demonstrate the plan generation process, Table 10, Table 11, Table 12, and Table 13 present example plans for NATURAL PLAN, GPQA, DocFinQA, and OlympiadBench. Generated using PlanGEN (Best of $\mathcal{N}$), these tables highlight the varied nature of plans produced across different task types. For GPQA, DocFinQA, and OlympiadBench (i.e., downstream reasoning tasks), the examples additionally illustrate how these generated plans are executed to derive the final answer.

More examples for agents and frameworks within PlanGEN are provided at https://anonymous.4open.science/r/plangen-0C99

Algorithm 1 Selection using Modified UCB with LLM-Guided Priors

Initialize: $R(a) \leftarrow 0, C(a) \leftarrow 1, Rec(a) \leftarrow 0, F(a) \leftarrow 0, D(a) \leftarrow 1, T \leftarrow 0$
Set $\lambda_{\text{prior}}, \alpha_{\text{diversity}}, \alpha_{\text{recovery}}$
Load LLM-guided priors
procedure SELECTALGORITHM(args)
    Compute prior decay: $\lambda_{\text{prior}} \leftarrow \frac{\lambda_{\text{prior}}}{1+T}$ ▷ Reduces as trials increase
    Set max exploration term $M \leftarrow 5$
    Obtain LLM prior scores: $LLM\_prior \leftarrow$ LLM_Guided_Prior$(args)$
    Compute max reward: $R_{\max} \leftarrow \max(R(a))$ (set to 1 if all rewards are 0)
    for each algorithm a **do**
        Compute normalized reward:

$$\bar{R}_{\text{norm}}(a) \leftarrow \frac{R(a)}{C(a)R_{\max}}$$

▷ Scales rewards between 0 and 1 for comparability
        Compute exploration term:

$$E(a) \leftarrow \min\left(\sqrt{\frac{2\log(T+1)}{C(a)}}, M\right)$$

▷ Encourages trying less-used algorithms, capped at M
        Compute diversity bonus:

$$D(a) \leftarrow \frac{\alpha_{\text{diversity}}}{C(a)+1}$$

▷ Penalizes frequently used algorithms to encourage variety
        Compute recovery bonus:

$$RecB(a) \leftarrow \alpha_{\text{recovery}} \cdot Rec(a)$$

▷ Rewards algorithms that perform well after failures
        Compute final UCB score:

$$UCB(a) \leftarrow \bar{R}_{\text{norm}}(a) + E(a) + \lambda_{\text{prior}} LLM\_prior(a) + D(a) + RecB(a)$$

▷ Balances exploitation, exploration, diversity, and recovery
    end for
    Select best algorithm:

$$a^* \leftarrow \arg\max_a UCB(a)$$

    return $(a^*, UCB(a^*))$
end procedure

Benchmark	Problem Statement	Generated Constraints
Calendar Scheduling	You are an expert at scheduling meetings. You are given a few constraints on the existing schedule of each participant, the meeting duration, and possibly some preferences on the meeting time. Note there exists a solution that works with existing schedule of every participant. ——— **TASK:** You need to schedule a meeting for Michelle, Steven and Jerry for one hour between the work hours of 9:00 to 17:00 on Monday. ——— Here are the existing schedules for everyone during the day: - Michelle has meetings on Monday during 11:00 to 12:00; - Steven has blocked their calendar on Monday during 9:00 to 9:30, 11:30 to 12:00, 13:30 to 14:00, 15:30 to 16:00; -Jerry has blocked their calendar on Monday during 9:00 to 9:30, 10:00 to 11:00, 11:30 to 12:30, 13:00 to 14:30, 15:30 to 16:00, 16:30 to 17:00; ——— Find a time that works for everyone's schedule and constraints.	**Instance-Specific Constraints and Details:** Participants: Michelle, Steven, Jerry - The meeting must include all three. ——— Meeting Duration: 1 hour - The meeting requires a contiguous 60-minute slot. ——— Work Hours: 9:00 to 17:00 (Monday) - The meeting must start and end within these working hours. This acts as a global constraint. ——— Individual Schedules (Existing Meetings/Blocked Times): - Michelle: Busy from 11:00 to 12:00 on Monday. - Steven: Busy from 9:00 to 9:30, 11:30 to 12:00, 13:30 to 14:00, and 15:30 to 16:00 on Monday. - Jerry: Busy from 9:00 to 9:30, 10:00 to 11:00, 11:30 to 12:30, 13:00 to 14:30, and 15:30 to 17:00 on Monday (note this constraint ends at the end of the workday). ——— While the problem states "Note there exists a solution," this isn't a constraint in itself; it's a guarantee that a feasible solution exists given the constraints. The actual constraints are the schedules, meeting duration, and working hours.
Meeting Planning	You are visiting San Francisco for the day and want to meet as many friends as possible. Solve the problem by considering various different schedules and picking the best one to optimize your goals. ——— Travel distances (in minutes): Bayview to Golden Gate Park: 22. Golden Gate Park to Bayview: 23. ——— CONSTRAINTS: You arrive at Bayview at 9:00AM. Barbara will be at Golden Gate Park from 8:00AM to 11:30AM. You'd like to meet Barbara for a minimum of 90 minutes. ——— Your response should start with 'SOLUTION:'	**Instance-Specific Constraints and Details:** Travel Times: - Bayview to Golden Gate Park: 22 minutes. - Golden Gate Park to Bayview: 23 minutes (Note: This information is provided but not strictly necessary for this specific problem as the return trip isn't relevant to meeting Barbara). ——— Locations: - Starting Location: Bayview - Meeting Location: Golden Gate Park ——— Scheduled Meeting Times/Participant Availability: - Barbara is available at Golden Gate Park from 8:00 AM to 11:30 AM. ——— Arrival Time: - You arrive at Bayview at 9:00 AM. ——— Minimum Meeting Duration: - The meeting with Barbara must be at least 90 minutes long. ——— Sequence Requirements: - Travel (Bayview to Golden Gate Park) $\rightarrow$ Wait (if necessary, due to arriving before Barbara's availability ends) $\rightarrow$ Meeting (with Barbara) ——— Objective: Maximize the number of friends met (in this case, it's simplified to just meeting Barbara, so the objective is to determine IF a meeting is possible given the constraints).

Table 6: Examples of constraints generated by the constraint agent given problem statement. These examples illustrate the constraint generation for calendar scheduling and meeting planning from NATURAL PLAN

Question: Two quantum states with energies E1 and E2 have a lifetime of 10^{-9} sec and 10^{-8} sec, respectively. We want to clearly distinguish these two energy levels. Which one of the following options could be their energy difference so that they can be clearly resolved?
(A) 10^{-11} eV
(B) 10^{-8} eV
(C) 10^{-9} eV
(D) 10^{-4} eV

Constraints for Verifying the Energy Difference Solution for Resolvable Quantum States

This problem revolves around the energy-time uncertainty principle, which dictates the resolvability of energy levels given their lifetimes. To verify a proposed energy difference ($\Delta E = |E_2 - E_1|$) that allows clear resolution, we need constraints encompassing the following:

1. Uncertainty Principle Adherence:
Primary Constraint: The product of the uncertainty in energy (ΔE) and the uncertainty in time (Δt) must satisfy the energy-time uncertainty relation: $\Delta E \Delta t \geq \frac{\hbar}{2}$, where $\hbar$ is the reduced Planck constant.
Specific Constraint: Since we have two states, we must consider the *combined* uncertainty in time. We can approximate this as the average of the individual lifetimes: $\Delta t \approx \frac{\tau_1 + \tau_2}{2}$, where τ_1 and τ_2 are the lifetimes of the two states.
Operational Constraint: For "clear resolution," we typically require ΔE to be significantly greater than the uncertainty dictated by the uncertainty principle. A common rule of thumb is $\Delta E \geq \frac{\hbar}{\Delta t}$. A stricter constraint might use a larger factor (e.g., $\Delta E \geq \frac{2\hbar}{\Delta t}$) for even clearer resolution.

2. Units Consistency:
Unit Constraint: All calculations must use consistent units. Ensure that energy (ΔE), time (Δt), and $\hbar$ are expressed in compatible units (e.g., Joules and seconds, or eV and seconds). Clearly state the chosen unit system.

3. Numerical Accuracy:
Precision Constraint: Calculations should be performed with sufficient numerical precision to avoid rounding errors. Define the acceptable level of precision.
Constant Accuracy: Use a precise value for the reduced Planck constant ($\hbar$).

4. Solution Justification:
Explanation Constraint: The solution must not only provide a numerical value for ΔE but also clearly explain why this value satisfies the constraints and allows for clear resolution. This explanation should reference the uncertainty principle and the chosen resolution criterion.

5. Alternative Solutions Consideration (Optional):
Exploration Constraint: Explore whether multiple values of ΔE could satisfy the constraints. If so, discuss the trade-offs between different solutions.

Example Implementation of Constraints:

Let's assume we're working in SI units (Joules and seconds).
1. Uncertainty Principle: $\Delta E \geq \frac{\hbar}{[(10^{-9}\,\text{s} + 10^{-8}\,\text{s})/2]}$, where $\hbar \approx 1.054 \times 10^{-34}$ J s.
2. Units: All calculations are in Joules and seconds.
3. Numerical Accuracy: Calculations will be performed to at least 4 significant figures.
4. Justification: The solution will explain how the calculated ΔE is significantly larger than the uncertainty, ensuring clear resolution.

Table 7: Examples of constraints generated by the constraint agent given reasoning question. The example illustrates the constraint generation for solving physics question from GPQA

Question: For a positive integer a, define a sequence of integers $x_1, x_2, \ldots$ by letting $x_1 = a$ and $x_{n+1} = 2x_n + 1$ for $n \geq 1$. Let $y_n = 2^{x_n} - 1$. Determine the largest possible k such that, for some positive integer a, the numbers $y_1, \ldots, y_k$ are all prime.

Constraints for Verifying the Solution Plan for Prime Sequence Problem

This problem requires a solution plan that likely involves a combination of mathematical reasoning and computational verification. Here's a breakdown of constraints to verify and evaluate such a plan:

I. Mathematical Reasoning Constraints:

1. **Correct Recurrence Relation:** The plan must correctly interpret and utilize the given recurrence relations for x_n and y_n. Verify that the plan doesn't misinterpret or modify these definitions.
2. **Mersenne Primes Understanding:** The plan should demonstrate an understanding that y_n are Mersenne numbers (numbers of the form $2^p - 1$). It should acknowledge that for y_n to be prime, x_n must itself be prime (though this is a necessary but not sufficient condition).
3. **Logical Flow and Justification:** The plan's reasoning should be clear, logical, and well-justified. Each step should follow from the previous ones, with clear explanations for any claims or deductions made. Avoid leaps of logic or unsubstantiated assertions.
4. **Handling of Base Case ($k = 1$):** The plan should explicitly address the trivial case where $k = 1$. Since $y_1 = 2^a - 1$, any prime value of 'a' will result in y_1 being a potential Mersenne prime. This confirms that $k \geq 1$.
5. **Strategy for Finding Maximum k:** The plan must outline a clear strategy for determining the *largest* possible k. This likely involves exploring increasing values of k and checking the primality of the resulting y_n sequence. The plan should explain how it avoids infinite search or unnecessary computations.
6. **Optimality Argument:** The plan should include an argument (or at least a strategy for developing an argument) for why the found value of k is indeed the *largest* possible. This might involve demonstrating that for $k + 1$, no suitable 'a' exists.

II. Computational Verification Constraints:

1. **Primality Testing Algorithm:** If the plan involves computational primality testing, it should specify the algorithm used (e.g., probabilistic tests like Miller-Rabin or deterministic tests like AKS). The plan should justify the choice of algorithm based on the size of the numbers being tested and the desired level of certainty.
2. **Correct Implementation:** If code is involved, it must be correct and free of errors. The plan should include testing procedures to ensure the code accurately implements the chosen primality test and the recurrence relations.
3. **Computational Resources:** The plan should consider the computational resources required for the primality testing. For large values of x_n, testing the primality of y_n can be computationally expensive. The plan should address how it manages these computational challenges.
4. **Range of 'a' considered:** The plan should specify the range of values for 'a' that will be explored. It should justify this range and explain why exploring values outside this range is unnecessary.
5. **Termination Condition:** The plan must have a clear termination condition for its computational search. It should explain how it determines when to stop searching for larger values of 'k'.

III. Presentation and Clarity Constraints:

1. **Clear and Concise Explanation:** The plan should be presented in a clear, concise, and easy-to-understand manner. Avoid unnecessary jargon or overly complex language.
2. **Well-Organized Structure:** The plan should be well-organized, with a logical flow of ideas. Use headings, subheadings, and bullet points to improve readability.
3. **Reproducibility:** The plan should be sufficiently detailed to allow another person to reproduce the results. All assumptions, parameters, and procedures should be clearly stated.

Table 8: Examples of constraints generated by the constraint agent given mathematical reasoning problem. The example illustrates the constraint generation for solving MATH question from OlympiadBench

Problem Statement: You are an expert at scheduling meetings. You are given a few constraints on the existing schedule of each participant, the meeting duration, and possibly some preferences on the meeting time. Note there exists a solution that works with existing schedule of every participant.

TASK: You need to schedule a meeting for Michelle, Steven and Jerry for one hour between the work hours of 9:00 to 17:00 on Monday.

Here are the existing schedules for everyone during the day:
- Michelle has meetings on Monday during 11:00 to 12:00;
- Steven has blocked their calendar on Monday during 9:00 to 9:30, 11:30 to 12:00, 13:30 to 14:00, 15:30 to 16:00;
-Jerry has blocked their calendar on Monday during 9:00 to 9:30, 10:00 to 11:00, 11:30 to 12:30, 13:00 to 14:30, 15:30 to 16:00, 16:30 to 17:00;

Find a time that works for everyone's schedule and constraints.

Instance-Specific Constraints and Details:

Participants: Michelle, Steven, Jerry - The meeting must include all three.

Meeting Duration: 1 hour - The meeting requires a contiguous 60-minute slot.

Work Hours: 9:00 to 17:00 (Monday) - The meeting must start and end within these working hours. This acts as a global constraint.

Individual Schedules (Existing Meetings/Blocked Times):
- Michelle: Busy from 11:00 to 12:00 on Monday.
- Steven: Busy from 9:00 to 9:30, 11:30 to 12:00, 13:30 to 14:00, and 15:30 to 16:00 on Monday.
- Jerry: Busy from 9:00 to 9:30, 10:00 to 11:00, 11:30 to 12:30, 13:00 to 14:30, and 15:30 to 17:00 on Monday (note this constraint ends at the end of the workday).

While the problem states "Note there exists a solution," this isn't a constraint in itself; it's a guarantee that a feasible solution exists given the constraints. The actual constraints are the schedules, meeting duration, and working hours.

Generated Plan:

1. Analyze Constraints:
- Participants: Michelle, Steven, Jerry
- Duration: 1 hour
- Work hours: 9:00 - 17:00
- Day: Monday

2. Identify Individual Availability:
- Michelle: 9:00-11:00, 12:00-17:00
- Steven: 9:30-11:30, 12:00-13:30, 14:00-15:30, 16:00-17:00
- Jerry: 9:30-10:00, 11:00-11:30, 12:30-13:00, 14:30-15:30, 16:00-16:30

3. Determine Common Availability: Looking for 1-hour slots:
- Comparing Michelle and Steven: 9:30-11:00, 12:00-13:30, 14:00-15:30, 16:00-17:00
- Comparing the above with Jerry: 14:30-15:30

4. Prioritize Earliest Time Slot: The earliest 1-hour slot is 14:30-15:30

5. Ensure Solution Exists: The slot 14:30-15:30 works for all three.

6. Strict Format Requirement:
SOLUTION: Here is the proposed time: Monday, 14:30 - 15:30

Verification:

The plan demonstrates significant improvement compared to the initial plan. It correctly identifies individual availabilities and successfully finds the two viable one-hour slots: 9:30-10:30 and 14:30-15:30. It avoids exploring irrelevant shorter slots. The plan is well-structured and clearly explains each step. It's on the verge of providing the correct solution. However, it has not yet explicitly stated the final answer (the prioritized earliest time slot) which is a key part of the task. Therefore, while much improved, it doesn't quite merit a top score due to this omission.

Score: 95

Table 9: Examples of verification generated by the verification agent given generated plan and constraints. This example illustrates the verification process for calendar scheduling from NATURAL PLAN

Category	Problem Statement	Generated Plan
Calendar Scheduling	You are an expert at scheduling meetings. You are given a few constraints on the existing schedule of each participant, the meeting duration, and possibly some preferences on the meeting time. Note there exists a solution that works with existing schedule of every participant. ——— **TASK:** You need to schedule a meeting for Michelle, Steven and Jerry for one hour between the work hours of 9:00 to 17:00 on Monday. ——— Here are the existing schedules for everyone during the day: - Michelle has meetings on Monday during 11:00 to 12:00; - Steven has blocked their calendar on Monday during 9:00 to 9:30, 11:30 to 12:00, 13:30 to 14:00, 15:30 to 16:00; -Jerry has blocked their calendar on Monday during 9:00 to 9:30, 10:00 to 11:00, 11:30 to 12:30, 13:00 to 14:30, 15:30 to 16:00, 16:30 to 17:00; ——— Find a time that works for everyone's schedule and constraints.	1. Analyze Constraints: - Participants: Michelle, Steven, Jerry - Duration: 1 hour - Work hours: 9:00 - 17:00 - Day: Monday ——— 2. Identify Individual Availability: - Michelle: 9:00-11:00, 12:00-17:00 - Steven: 9:30-11:30, 12:00-13:30, 14:00-15:30, 16:00-17:00 - Jerry: 9:30-10:00, 11:00-11:30, 12:30-13:00, 14:30-15:30, 16:00-16:30 ——— 3. Determine Common Availability: Looking for 1-hour slots: - Comparing Michelle and Steven: 9:30-11:00, 12:00-13:30, 14:00-15:30, 16:00-17:00 - Comparing the above with Jerry: 14:30-15:30 ——— 4. Prioritize Earliest Time Slot: The earliest 1-hour slot is 14:30-15:30 ——— 5. Ensure Solution Exists: The slot 14:30-15:30 works for all three. ——— 6. Strict Format Requirement: SOLUTION: Here is the proposed time: Monday, 14:30 - 15:30
Meeting Planning	You are visiting San Francisco for the day and want to meet as many friends as possible. Solve the problem by considering various different schedules and picking the best one to optimize your goals. ——— Travel distances (in minutes): Bayview to Golden Gate Park: 22. Golden Gate Park to Bayview: 23. ——— CONSTRAINTS: You arrive at Bayview at 9:00AM. Barbara will be at Golden Gate Park from 8:00AM to 11:30AM. You'd like to meet Barbara for a minimum of 90 minutes.	You leave Bayview at 9:00 AM and travel 22 minutes to Golden Gate Park, arriving at 9:22 AM. You meet Barbara from 9:22 AM – 10:52 AM, fulfilling the 90-minute requirement.
Trip Planning	You plan to visit 4 European cities for 12 days in total. You only take direct flights to commute between cities. You plan to stay in Budapest for 6 days. You plan to stay in Reykjavik for 5 days. You would like to meet your friends at Reykjavik between day 6 and day 10 to tour together. You want to spend 2 days in Milan. You are going to attend a wedding in Milan between day 10 and day 11. You would like to visit Porto for 2 days. ——— Here are the cities that have direct flights: Budapest and Reykjavik, Reykjavik and Milan, Budapest and Milan, Milan and Porto. ——— Find a trip plan of visiting the cities for 12 days by taking direct flights to commute between them.	Day 1-6: Arriving in Budapest and visit Budapest for 6 days. Day 6: Fly from Budapest to Reykjavik. Day 6-10: Visit Reykjavik for 5 days. Day 10: Fly from Reykjavik to Milan. Day 10-11: Visit Milan for 2 days. Day 11: Fly from Milan to Porto. Day 11-12: Visit Porto for 2 days.

Table 10: Examples of generated plans by PlanGEN (Best of $\mathcal{N}$) given problem statement. These examples illustrate the plan generation for all three task categories from NATURAL PLAN

Question: Two quantum states with energies E1 and E2 have a lifetime of 10^{-9} sec and 10^{-8} sec, respectively. We want to clearly distinguish these two energy levels. Which one of the following options could be their energy difference so that they can be clearly resolved?

Plan to solve the problem:

1. Identify the relevant physics principle: This problem involves the energy-time uncertainty principle, which states that the product of the uncertainty in energy (ΔE) and the uncertainty in time (Δt) is greater than or equal to h-bar/2 ($\hbar/2$). The uncertainty in time is related to the lifetime of the states. To clearly distinguish the energy levels, the energy difference between them must be greater than the uncertainty in their energies.

2. Relate lifetimes to time uncertainty: The lifetimes given (10^{-9} s and 10^{-8} s) represent the uncertainties in the time for each state (Δt_1 and Δt_2). We need to consider which lifetime is more restrictive for resolving the energy levels. Since we want to *clearly* distinguish them, we should use the *smaller* lifetime as the relevant Δt. This is because the shorter lifetime leads to a larger energy uncertainty, making the levels harder to resolve.

3. Calculate the minimum energy difference: Use the energy-time uncertainty principle ($\Delta E * \Delta t \geq \hbar/2$) with the smaller lifetime to calculate the minimum resolvable energy difference (ΔE). We can approximate this by setting $\Delta E * \Delta t \approx \hbar/2$. We'll need the value of $\hbar$ (Planck's constant divided by 2π) in eV·s.

4. Compare the calculated ΔE with the choices: Compare the calculated minimum resolvable energy difference (ΔE) with the given options (A) through (D). The correct answer will be the option that is *greater* than the calculated ΔE, ensuring that the energy difference is large enough to resolve the two levels.

5. Select the correct answer: Based on the comparison in step 4, choose the option that represents an energy difference greater than the minimum resolvable energy difference.

Plan Execution:

1. Identify the relevant physics principle: The energy-time uncertainty principle ($\Delta E * \Delta t \geq \hbar/2$) governs this situation.

2. Relate lifetimes to time uncertainty: The shorter lifetime, 10^{-9} s, is the relevant Δt because it leads to a larger energy uncertainty, making resolution more difficult. So, $\Delta t = 10^{-9}$ s.

3. Calculate the minimum energy difference:
We have $\Delta E * \Delta t = \hbar/2$. We need $\hbar$ in eV·s. $\hbar = 6.582 \times 10^{-16}$ eV·s.
$\Delta E = (\hbar/2)/\Delta t$
$\Delta E = (6.582 \times 10^{-16}$ eV·s$/2)/(10^{-9}$ s$)$
$\Delta E = 3.291 \times 10^{-7}$ eV
This is approximately 3×10^{-7} eV.

4. Compare the calculated ΔE with the choices: We need an energy difference greater than 3×10^{-7} eV to clearly resolve the levels.
(A) 10^{-11} eV - Too small
(B) 10^{-8} eV - Too small
(C) 10^{-9} eV - Too small
(D) 10^{-4} eV - Large enough

5. Select the correct answer: Option (D) is the only one significantly larger than the calculated minimum energy difference.

Answer: D

Table 11: Examples of generated plans by PlanGEN (Best of $\mathcal{N}$) given physics problem. The example illustrates the plan generation and it's execution for physics problem from GPQA

<Entire Document>

Question: What is the percentage change in cash flow hedges in 2011 compare to the 2010?

Plan to solve the problem:

1. Locate the information on cash flow hedges: Search the document for "cash flow hedges" or similar terms related to derivative instruments used for hedging cash flows. Focus on sections discussing derivative instruments, hedging activities, and related notional amounts.

2. Identify the notional amounts for 2011 and 2010: Once the relevant section is found (Note 2i), extract the notional amounts of the cash flow hedges specifically for fiscal year 2011 and fiscal year 2010.

3. Calculate the percentage change: Use the formula for percentage change: [(Notional Amount 2011 - Notional Amount 2010) / Notional Amount 2010] * 100. This will provide the percentage change in cash flow hedges from 2010 to 2011.

Plan Execution:

1. Locate the information on cash flow hedges: The information on cash flow hedges was located in a section discussing derivative instruments and hedging activities. Specifically, it was found within a sentence detailing the notional amounts of various hedging instruments.

Code (if needed):
```python
# No code needed for this step.
```
Result: Information on cash flow hedges located successfully.

2. Identify the notional amounts for 2011 and 2010: The notional amounts for cash flow hedges were identified as $153.7 million for 2011 and $139.9 million for 2010.

Code (if needed):
```python
# No code needed for this step.
```
Result: 2011 notional amount: $153.7 million; 2010 notional amount: $139.9 million.

3. Calculate the percentage change:

Code (if needed):
```python
notional_2011 = 153.7
notional_2010 = 139.9
percentage_change = ((notional_2011 - notional_2010) / notional_2010) * 100
rounded_percentage_change = round(percentage_change, 1)
print(f"{rounded_percentage_change}%")
```

Answer: 9.9%

Table 12: Examples of generated plans by PlanGEN (Best of $\mathcal{N}$) given financial reasoning problem. The example illustrates the plan generation and it's execution for problem from DocFinQA

Question: For a positive integer a, define a sequence of integers $x_1, x_2, \ldots$ by letting $x_1 = a$ and $x_{n+1} = 2x_n + 1$ for $n \geq 1$. Let $y_n = 2^{x_n} - 1$. Determine the largest possible k such that, for some positive integer a, the numbers $y_1, \ldots, y_k$ are all prime.

Plan for Solving the IMO Problem:
1. Problem Understanding: The problem defines a recursive sequence 'x_n' based on an initial value 'a', and a related sequence 'y_n' based on 'x_n'. The goal is to find the largest 'k' such that 'y_1', 'y_2', ..., 'y_k' can all be prime numbers for some choice of 'a'.
2. Key Information and Variables:
'a': A positive integer, the starting value for the sequence 'x_n'. This is the variable we need to choose optimally.
'x_n': Defined recursively as 'x_1 = a' and 'x_(n+1) = 2x_n + 1'.
'y_n': Defined as 'y_n = 2^(x_n) - 1'. This is the sequence we need to check for primality.
'k': The length of the subsequence of 'y_n' consisting only of primes. We want to maximize this.
3. Problem Type and Method: This is a number theory problem involving sequences, primality, and recursion. The method will involve a combination of:
Analyzing the structure of 'x_n' and 'y_n': Finding a closed-form expression for 'x_n' in terms of 'a' and 'n' will be crucial. This will help us understand the structure of 'y_n'.
Primality testing/properties: We'll need to leverage known properties of prime numbers, particularly those related to Mersenne primes (primes of the form 2^p - 1, where p is prime).
Case analysis/exploration: We might need to explore small values of 'a' and 'k' to identify patterns and formulate conjectures.
Proof by contradiction: To establish the largest possible 'k', we may need to assume a larger value is possible and derive a contradiction.
4. Solution Process:
Step 1: Derive a closed-form expression for 'x_n'. The recursive definition suggests a linear recurrence relation. We can solve it to express 'x_n' directly in terms of 'a' and 'n'.
Step 2: Substitute 'x_n' into the definition of 'y_n'. This gives us a formula for 'y_n' in terms of 'a' and 'n'.
Step 3: Analyze the primality conditions for 'y_n'. Given that 'y_n' has the form 2^m - 1, we know that for 'y_n' to be prime, 'm' (which is 'x_n') must itself be prime. This suggests focusing on values of 'a' that lead to prime values of 'x_n'.
Step 4: Explore small values of 'a' and 'k'. This helps to identify patterns and limitations on the possible values of 'k'.
Step 5: Investigate divisibility properties. Consider the factors of 'y_n' and how they might relate to previous terms in the sequence. Look for patterns in the factors that might prevent multiple consecutive 'y_n' from being prime.
Step 6: Formulate a conjecture for the maximum 'k'. Based on the observations from the previous steps, propose a value for the maximum 'k'.
Step 7: Prove the conjecture. Use a proof by contradiction. Assume a larger 'k' is possible and demonstrate that this leads to a contradiction based on the divisibility properties of 'y_n'. This will establish the proposed value of 'k' as the maximum.

Plan Execution:
1. Closed-form for x_n: The recurrence relation $x_{n+1} = 2x_n + 1$ with $x_1 = a$ can be solved as follows:
$x_2 = 2a + 1$
$x_3 = 2(2a + 1) + 1 = 4a + 3 = 2^2a + 2^2 - 1$
$x_4 = 2(4a + 3) + 1 = 8a + 7 = 2^3a + 2^3 - 1$
By induction, we can show that $x_n = 2^{n-1}a + 2^{n-1} - 1 = 2^{n-1}(a + 1) - 1$.
2. y_n in terms of a and n: Substituting the expression for x_n into $y_n = 2^{x_n} - 1$, we get $y_n = 2^{2^{n-1}(a+1)-1} - 1$.
3. Primality of y_n: For y_n to be prime, x_n must be prime. Let $x_n = p_n$, where p_n is a prime number.
4. Exploring small values:
If $a = 1$, $x_1 = 1$, $y_1 = 2^1 - 1 = 1$, which is not prime.
If $a = 2$, $x_1 = 2$, $y_1 = 2^2 - 1 = 3$, which is prime.
$x_2 = 2(2) + 1 = 5$, $y_2 = 2^5 - 1 = 31$, which is prime.
$x_3 = 2(5) + 1 = 11$, $y_3 = 2^{11} - 1 = 2047 = 23 \cdot 89$, which is not prime. So, for $a = 2$, $k = 2$.
5. Divisibility:
If $x_1 = a$ is prime, say $a = p$, then $x_2 = 2p + 1$. If x_2 is also prime, then y_1 and y_2 are potentially Mersenne primes. However, $x_3 = 2(2p + 1) + 1 = 4p + 3$. If $p = 2$, $x_3 = 11$, and y_3 is not prime. If $p = 3$, $x_2 = 7$, $x_3 = 15$, which is not prime. If $p = 5$, $x_2 = 11$, $x_3 = 23$, $x_4 = 47$, $x_5 = 95$, which is not prime.

If $k = 3$, we need $x_1 = a$, $x_2 = 2a + 1$, and $x_3 = 4a + 3$ to all be prime. If $a = 2$, $x_1 = 2$, $x_2 = 5$, $x_3 = 11$. But $y_3 = 2^{11} - 1 = 2047 = 23 \cdot 89$, which is not prime.

6. Conjecture: The largest possible k is 2.
7. Proof: We have shown that for $a = 2$, y_1 and y_2 are prime. We need to show that y_3 cannot be prime. $x_3 = 4a + 3$. If $a \equiv 2 \pmod 3$, then $4a + 3 \equiv 4(2) + 3 \equiv 11 \equiv 2 \pmod 3$. If $a \equiv 1 \pmod 3$, then $4a + 3 \equiv 4(1) + 3 \equiv 7 \equiv 1 \pmod 3$. If $a \equiv 0 \pmod 3$, then $4a + 3 \equiv 3 \equiv 0 \pmod 3$. So, if $a > 3$ and $a \equiv 0 \pmod 3$, then x_3 is divisible by 3 and greater than 3, so it is not prime. If $a = 3$, $x_3 = 15$, not prime. If $a = 2$, $x_3 = 11$, and $y_3 = 2047 = 23 \cdot 89$.

So the final answer is $\boxed{2}$.

Table 13: Examples of generated plans by PlanGEN (Best of $\mathcal{N}$) given maths problem. The example illustrates the plan generation and it's execution for problem from OlympiadBench (MATH)