LRPlan: A Multi-Agent Collaboration of Large Language and Reasoning Models for Planning with Implicit & Explicit Constraints

T Karthikeyan[*1], **Om Dehlan**[*2], **Mausam**[1,2], **Manish Gupta**[3]
[1]Yardi School of Artificial Intelligence, IIT Delhi
[2]Department of Computer Science and Engineering, IIT Delhi
[3]Microsoft, India
tkarthikeyan132@gmail.com, om.dehlan@gmail.com,
mausam@cse.iitd.ac.in, gmanish@microsoft.com

Abstract

Our goal is to build language model based multi-agent systems for complex planning problems involving multiple explicit and implicit constraints, some of which may be commonsense. Our initial investigations reveal that large language models (LLMs) are often unable to maintain consistency across the planning process, whereas large reasoning models (LRMs) struggle with handling implicit commonsense constraints. In response, we introduce LRPlan, a novel domain-independent, language-based multi-agent architecture where LLM and LRM-based agents collaborate at training time to abstract important patterns, heuristics and insights about the domain. At test time, they collaborate in implementing these learned patterns and insights for a new planning instance. We perform experiments on two datasets, TravelPlanner and TimeArena-Static, and use two LLM-LRM combinations from GPT and DeepSeek families. We find that LRPlan outperforms various multi-agent and single-agent baselines obtaining notably higher accuracy as well as cost efficiency. We make the code publicly available.[1]

1 Introduction

Planning, a cornerstone of artificial intelligence, involves generating a sequence of actions to achieve a specified goal while adhering to various constraints. We study planning problems specified in natural language, where some constraints may be explicit, i.e., directly stated in the input, and others implicit, often requiring commonsense reasoning. Our goal is to develop domain-independent approaches that do not require human intervention per domain; rather, make use of minimal training data from that domain to invoke a domain-specific architecture for handling any new instance from the same domain.

Agentic architectures over modern large *language* models (LLMs), e.g., (Chen et al., 2024b; Song et al., 2024) have the potential to tackle planning problems in a generalizable manner. However, by themselves, LLMs often encounter difficulties with complex, multi-constraint problems. They struggle with maintaining constraint consistency throughout the planning process and often cannot perform robust planning or self-verification on their own (Kambhampati et al., 2024). A recent alternative are the large *reasoning* models (LRMs), which are trained to “think” before generating responses (DeepSeek-AI et al., 2025) and might be more suitable for planning. Our initial explorations with LRMs suggest that while they indeed exhibit better performance on problems with explicit constraints, they underperform if the domain has many implicit commonsense constraints.

We hypothesize that, similar to a human brain, where specialized skills in right and left hemispheres work together for handling complex stimuli (Hughes and Rutherford, 2013), agentic AI architectures will require collaboration between LLMs and LRMs for solving such complex planning problems. In response, we design LRPlan, a domain-independent, language-based, multi-agent architecture for complex planning that uses both LLMs and LRMs at both training and test times.

LRPlan first generates reasoning traces over training samples in a domain. It then implements several agents that are tasked with inducing common patterns, heuristics and an understanding of mistakes by analyzing these traces. This acquired domain-specific understanding informs a simple, streamlined planning process for any new problem instance, involving just two components: initial plan generator and refiner. Reasoning trace generation (train) and refinement of initial plan (test) are done by LRMs, and all other agents use LLMs.

To study the effectiveness of the architecture, we use two datasets, TravelPlanner (Xie et al.,

---

[*] Equal Contribution

[1] `https://github.com/dair-iitd/LRPlan`

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 8280–8310
November 4-9, 2025 ©2025 Association for Computational Linguistics

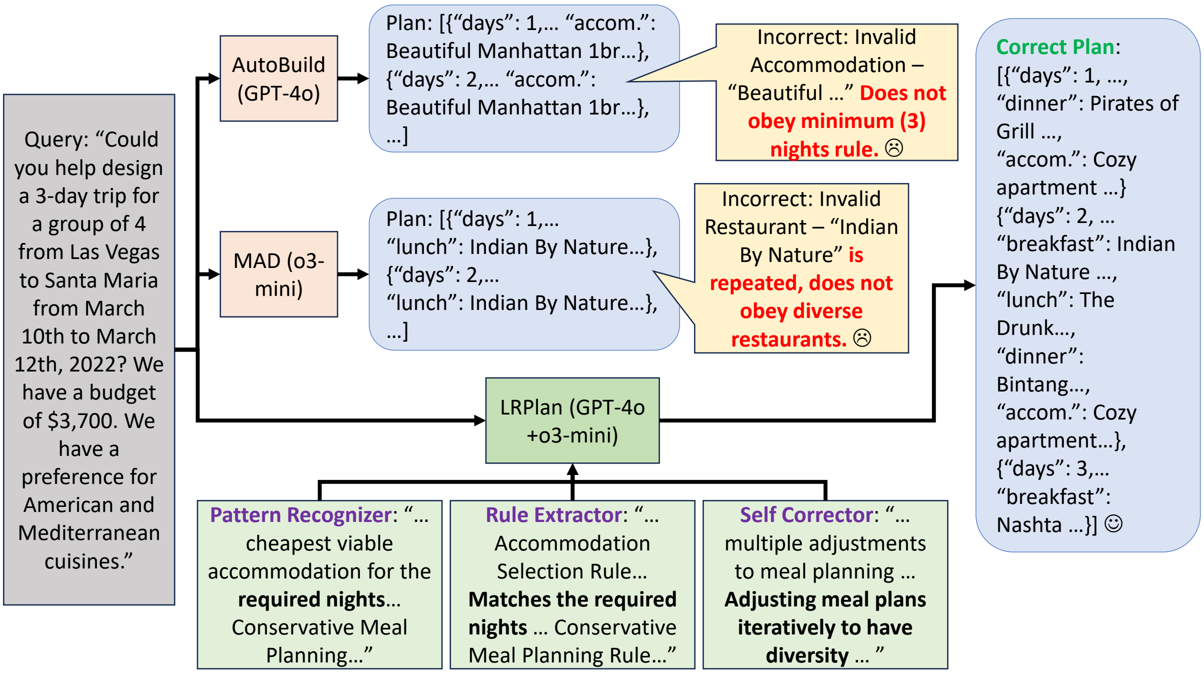

Query: "Could you help design a 3-day trip for a group of 4 from Las Vegas to Santa Maria from March 10th to March 12th, 2022? We have a budget of $3,700. We have a preference for American and Mediterranean cuisines."
AutoBuild (GPT-4o)
Plan: [{"days": 1,… "accom.": Beautiful Manhattan 1br…}, {"days": 2,… "accom.": Beautiful Manhattan 1br…}, …]
Incorrect: Invalid Accommodation – "Beautiful …" Does not obey minimum (3) nights rule. ☹
MAD (o3-mini)
Plan: [{"days": 1,… "lunch": Indian By Nature…}, {"days": 2,… "lunch": Indian By Nature…}, …]
Incorrect: Invalid Restaurant – "Indian By Nature" is repeated, does not obey diverse restaurants. ☹
LRPlan (GPT-4o +o3-mini)
Pattern Recognizer: "… cheapest viable accommodation for the required nights… Conservative Meal Planning…"
Rule Extractor: "… Accommodation Selection Rule… Matches the required nights … Conservative Meal Planning Rule…"
Self Corrector: "… multiple adjustments to meal planning … Adjusting meal plans iteratively to have diversity … "
Correct Plan: [{"days": 1, …, "dinner": Pirates of Grill …, "accom.": Cozy apartment …} {"days": 2, … "breakfast": Indian By Nature …, "lunch": The Drunk…, "dinner": Bintang…, "accom.": Cozy apartment…}, {"days": 3,… "breakfast": Nashta …}] ☺

Figure 1: Example from TravelPlanner dataset: Baselines like AutoBuild and Multi-Agent Debate fail to satisfy commonsense constraints like "minimim nights rule" and "diverse restaurants". Our proposed method, LRPLAN, generates an accurate plan satisfying all constraints with the help of synergistic collaboration between planner LLM and refiner LRM. Fig. 4 shows another example from TimeArena-Static dataset.

Method	**No Validator Needed during inference**	**Refines Initial Response**	**Multi-Agent**	**No Domain-Specific Human Effort**	**Learns from Training Samples**	**LLM-LRM Synergy**
Standard Prompting	✓	✗	✗	✓	✗	✗
CoT (Wei et al., 2022)	✓	✗	✗	✓	✗	✗
ReAct (Yao et al., 2022)	✗	✓	✗	✓	✗	✗
Reflexion (Shinn et al., 2023)	✗	✓	✗	✓	✓	✗
Self-Refine (Madaan et al., 2023)	✓	✓	✗	✓	✗	✗
Multi-Agent Debate (Du et al., 2023)	✓	✓	✓	✓	✗	✗
AutoAgents (Chen et al., 2024a)	✓	✓	✓	✓	✗	✗
SPP (Wang et al., 2024)	✓	✓	✓	✓	✗	✗
AutoBuild[2] (Wu et al., 2024)	✓	✓	✓	✓	✗	✗
AgentVerse (Chen et al., 2024b)	✓	✓	✓	✓	✗	✗
LLMFP (Hao et al., 2025b)	✗	✗	✓	✓	✗	✗
LLM-Modulo (Kambhampati et al., 2024)	✗	✓	✓	✗	✗	✗
PMC (Zhang et al., 2025)	✓	✓	✓	✗	✗	✗
ExpeL (Zhao et al., 2024)	✗	✗	✗	✓	✓	✗
AutoGuide (Fu et al., 2024)	✗	✓	✓	✓	✓	✗
LRPLAN (ours)	✓	✓	✓	✓	✓	✓

Table 1: Comparison of various existing complex multi-constraint planning methods with LRPLAN. The characteristics used for comparison are elaborated in Appendix A.6.

2024) and TimeArena (Zhang et al., 2024). Since TimeArena is originally conceived as an online planning dataset, we reformulate it into the static planning setting, where a single final plan is outputted instead of repeated planner-simulator interaction. We compare LRPLAN with state-of-the-art multi-agent systems applicable to our setting. We use standard prompting, chain-of-thought prompting, AutoGen's AutoBuild[2] (Wu et al., 2024), and multi-agent debate techniques (Du et al., 2023) as strong baselines to compare with. Fig. 1 shows the supremacy of LRPLAN on an example from TravelPlanner.

We experiment on two families of models, OpenAI and DeepSeek, that both have at least one LLM and LRM each. Our results demonstrate that LRPLAN outperforms all baselines on each model family obtaining notable accuracy gains. Our best result is LRPLAN on TimeArena-Static with GPT-4o and o3-mini achieving a 12.7 pt accuracy improvement over the closest baseline, MAD with o3-mini. Similarly for TravelPlanner, our best result is LRPLAN with DeepSeek-V3 and DeepSeek-R1 achieving a 8.89 pt accuracy improvement over the

---

[2] `https://microsoft.github.io/autogen/0.2/blog/2023/11/26/Agent-AutoBuild/`

closest baseline, AutoBuild with GPT-4o. Moreover, LRPLAN's cost efficiency is superior to its nearest competitor model. Additional ablations assess the value of LLM-LRM collaboration versus using just LLM or LRM.

In summary, our main contributions are as follows. (1) We study complex multi-constraint planning problems that contain both explicit and implicit constraints, including commonsense constraints. We propose LRPLAN, a multi-agent architecture that combines LLMs and LRMs for this task. (2) We reformulate TimeArena dataset to a static setting, and introduce evaluation metrics for this dataset. (3) We perform several experiments on two domains and two model families to find that LRPLAN substantially outperforms the state of the art in both accuracy and overall costs. We release all code and TimeArena-Static dataset for reproducibility and fostering further research on the topic.[1]

2 Related Work

Planning has long been a central challenge in AI, traditionally addressed through classical planners (Ghallab et al., 1998; Höller et al., 2020; Helmert, 2006) that rely on domain-specific modeling and optimization. While effective in structured settings, these approaches require significant manual effort and lack adaptability. Supervised machine learning and imitation learning techniques learn from examples of problems and their solutions; but need large-scale domain-dependent training data. The emergence of LLMs introduced new paradigms for planning via zero-shot prompting and chain-of-thought reasoning (Wei et al., 2022), enabling generalization without domain-specific training. However, single-agent LLM frameworks (Yao et al., 2022; Shinn et al., 2023; Madaan et al., 2023) often struggle with constraint consistency and implicit reasoning.

Multi-agent systems (MAS) such as AutoBuild (Wu et al., 2024), AutoAgents (Chen et al., 2024a), and AgentVerse (Chen et al., 2024b) improve task decomposition and collaboration, but frequently lack mechanisms for learning from cumulative past experiences. Debate-based methods (Du et al., 2023; Narsupalli et al., 2025) enhance reasoning through iterative critique but are resource-intensive. Role-based MAS like PMC (Zhang et al., 2025) and LLM-Modulo (Kambhampati et al., 2024) introduce structured agent roles, yet require manual domain adaptation and prompt engineering.

Recent work emphasizes learning from experience (Zhao et al., 2024; Fu et al., 2024), enabling agents to reflect and adapt using past trajectories. However, these are typically LLM-centric and do not leverage the complementary strengths of reasoning-focused models.

In contrast, our proposed architecture, LRPLAN, introduces a domain-independent, multi-agent framework that synergistically combines LLMs and LRMs. LRPLAN learns from reasoning traces to induce domain-specific heuristics and patterns, enabling robust planning with minimal domain-specific supervision. As opposed to previous works, in LRPLAN the validator is only being used on a very small number of training examples (4-10 per domain). The cost of validating this tiny set is negligible, a simple Pass/Fail verdict can be provided by a human expert in minutes or handled by an LLM-based evaluator if a plan validator does not exist. Unlike prior systems, LRPLAN handles both explicit and implicit constraints, including commonsense reasoning, and generalizes across domains without re-engineering. Table 1 shows a comparison between our proposed LRPLAN system with other existing complex multi-constraint planning methods. We discuss more details of related work in Appendix A.

3 Problem Setting

We wish to design planners that map a natural-language query q (planning problem instance) to a plan p using a structured, step-by-step sequence of actions, $p = \langle s_1, s_2, \dots, s_T \rangle$, that satisfies two types of constraints:

- **Explicit (hard) constraints:** facts and rules explicitly mentioned in q; e.g. "*arrive before 10 a.m.*" or "*use public transport only*".
- **Implicit commonsense constraints**: unstated yet context-obvious requirements such as temporal feasibility, non-overlapping resources, or culturally acceptable sequences.

For each domain, as supervision, we are given a *few-shot training set* $\mathcal{D}_{\text{train}} = \{(q_i, \mathcal{V}_i)\}_{i=1}^{N}$ with $N \leq 10$. Here $\mathcal{V}_i : p_i \mapsto \{\mathit{pass}, \mathit{fail}\}$ is a black-box *validator* that returns a binary correctness verdict for a candidate plan for each sample in training set.

Given $\mathcal{D}_{\text{train}}$, our problem is to design a language-agent system that, for unseen query q_{test},

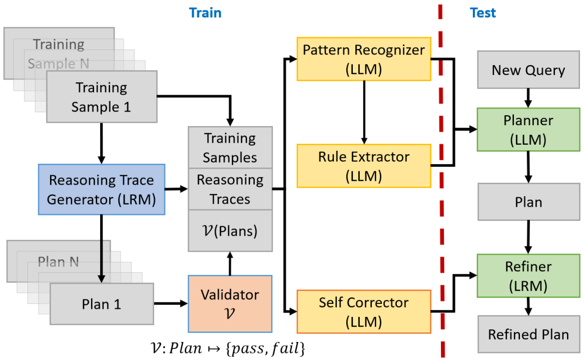


Figure 2: LRPlan architecture at train and test time.

produces a plan p_{test} that passes all implicit and explicit constraints.

4 The LRPlan Framework

LRPlan is a domain-independent planner, i.e., applies to any domain out-of-the-box. Specifically, it induces a test-time planner, which is targeted to a domain, without any human intervention. At a high level (see Fig. 2), LRPlan first prompts an LRM $\mathcal{M}$ with each of the N training samples q_i to generate a plan p_i along with a reasoning trace r_i. The validator $\mathcal{V}_i$ is then queried with the plan p_i, which generates a pass/fail label v_i that is added to the trace. All the labeled reasoning traces are collated and up to C samples per label (pass/fail) are then input to two meta-agents, Pattern Recognizer, and Self-Corrector. We call these *meta-agents*, because their output is at the domain-level, not at a query-level.

Pattern Recognizer is an LLM prompted to identify recurring structures (*patterns*) useful for solving problems in the domain. The output patterns are then provided to yet another meta-agent, Rule Extractor, whose job is to induce concrete *rules* required for a plan to be correct. We find that the use of two successive steps allows the LLMs to reflect on the domain better, with patterns denoting bottom-up first observations, and rules denoting a higher-level abstraction of these observations. Finally, the self-corrector meta-agent is tasked with extracting *insights* for correcting an incorrect plan. These patterns, rules and insights are combined to automatically construct prompts for planner and refiner agents.

Given a test query q_{test}, the planner agent drafts a candidate plan $\hat{p}_{\text{test}}$ guided by the domain's patterns and the rules. The refiner agent identifies and corrects any problems in $\hat{p}_{\text{test}}$ based on the self-corrector's insights to generate the output plan p_{test}. We now describe these components in detail.

4.1 Training Meta-Agents in LRPlan

Pattern Recognizer is tasked with studying how different reasoning traces unfold, finding common patterns or steps that often lead to the right answers, and spotting where things tend to go wrong. By breaking down and comparing several examples, the meta-agent learns what makes for good decision-making and what causes mistakes. This helps create clear, general tips or heuristics that might be useful for solving new problems more effectively in the future, while also making it easier to understand and explain how decisions are made. We list the Pattern Recognizer prompt in Appendix C.1.

Rule Extractor is tasked with converting the common patterns found in reasoning traces into a set of clear, easy-to-follow decision-making rules (heuristics) that can be used to solve similar problems in the future. The agent abstracts these patterns into general rules about what actions to take in specific situations, making planning faster, more consistent, and easier to understand. Rules help ensure that good strategies are reused and that decisions are made efficiently and coherently for a new query. We list the Rule Extractor prompt in Appendix C.2.

Self-Corrector: Sometimes, an LRM's reasoning trace shows that the model goes back, reflects, and changes its answer. These self-corrections can happen when the model notices a logical mistake, a wrong assumption, or a calculation error, and then tries a new approach or checks its work. Self-Corrector analyzes reasoning traces to learn how the LRM fixed its own mistakes, and based on this, identifies the typical types of errors that are being corrected and strategies for correction. These are output as a set of insights that form the basis of the refiner's prompt. We list the Self-Corrector prompt in Appendix C.3.

As an example, the pattern recognizer may find a pattern that "Plans that mix self-driving with other modes of transport (e.g., flights, trains, buses) often result in inconsistencies or failures." Rule extractor will convert it to this rule: "Avoid mixing self-driving with other transport modes in a single plan. Choose one consistent mode of transport throughout the itinerary." Finally, the corresponding insight could be "When a plan includes self-driving, replace all other transport modes with self-driving to maintain consistency and avoid conflicts." The output patterns, rules and insights generated by these meta-agents are listed in Appendix D.

4.2 Test-Time Agents in LRPLAN

The prompts for Planner and Refiner are automatically constructed based on the meta-agents' outputs, i.e., extracted patterns and rules, and insights, respectively. Planner and refiner are agents (and not meta-agents) as they operate on each test query separately.

Planner takes the input query and, using the patterns and rules, constructs a step-by-step plan that is meant to achieve the desired goal, while satisfying all relevant constraints. The planner interprets the task requirements, applies proven reasoning strategies and heuristics, and organizes the solution into logical, actionable steps. By integrating heuristics from previous reasoning and decision-making patterns and rules, the planner tries to ensure that the resulting plan is tailored to the specific needs of the task, and likely avoids common mistakes. LRPLAN uses an LLM for this agent.

Refiner critically evaluates and enhances the initial plan produced by the Planner Agent, using both the original input query and self-correction insights provided by the Self-Corrector. The Refiner Agent carefully reviews each step of the plan, checks for potential errors, inefficiencies, and overlooked constraints, and applies relevant self-correction strategies to improve the plan's feasibility and correctness. This agent is implemented with an LRM.

In summary, LRPLAN uses LLMs and LRMs in both training and testing phases to benefit from their complementary strengths in planning. LRMs are used in reasoning trace generation and plan improvement, where more emphasis is on explicit reasoning and validation. Other components use LLMs as they need a stronger inherent creativity in the models to create generalizations/abstractions, and propose a candidate plan. As an informal analogy to human brain, LRPLAN gets the intuition and creativity (right brain, LLMs) to coordinate with analytical thinking (left brain, LRMs) for complex planning tasks. A detailed, step-by-step walkthrough of the LRPLAN with concrete examples for each component is provided in Appendix B.

5 Datasets

We conduct experiments on two challenging benchmarks: **TravelPlanner** and a static version of **TimeArena**, both requiring long-horizon planning under explicit and implicit constraints. We now describe the two datasets and evaluation metrics used in each.

5.1 TravelPlanner

TravelPlanner presents a complex real-world planning environment with $\sim$4M database entries of Restaurants, Flights, Accommodations and Attractions, sourced from the internet. Agents must generate detailed itineraries covering transportation, accommodations, dining, and attractions while adhering to both explicit constraints (e.g., budget limits) and implicit commonsense constraints (e.g., avoiding repetitive visits, returning the car to origin, instead of leaving it in the middle of the journey).

The dataset consists of 1,225 curated planning intents, with a train/validation/test split of 45/180/1000. We use the first $N = 10$ samples from the training set of 45 to extract initial reasoning traces and set $C = 5$ for insight extraction sample selection. The evaluation is conducted on the validation set of 180 queries.[3] Queries are categorized into Easy, Medium, and Hard levels based on the number and complexity of constraints.

Evaluation Metrics: Following the original work on TravelPlanner (Xie et al., 2024), we evaluate AI-generated plans across multiple dimensions: *delivery rate* (fraction of queries where the agent delivers a feasible final plan), *commonsense constraint pass rate* (fraction of commonsense constraints satisfied, evaluated both using micro and macro scores), *hard constraint pass rate* (same value for explicit user constraints), and *final pass rate* (proportion of plans satisfying all constraints in the query). Micro pass rate measures per-constraint success, while macro pass rate measures per-plan success.

We note that, since TravelPlanner has been introduced, several works have used the dataset for their experiments, but in vastly different settings. Some settings use external (non-LLM) tools along with LLMs (Hao et al., 2025a; Gundawar et al., 2024; Zhang et al., 2025), some have human experts write domain-specific prompts just to solve the specific dataset (Lee et al., 2025), and yet others make the implicit commonsense constraints explicit in the prompt (Chen et al., 2024c). These different settings naturally yield vastly different pass rates, and are not directly comparable with each other. We experiment in the setting of domain-independent planning, i.e., that no human expert writes any specific prompt for the domain. Moreover, none of compared systems get access to any external tools.

---

[3] Since test set is large, many other papers also report results on validation set to save on LLM/LRM costs.

5.2 TimeArena-Static

TimeArena is a textual simulation evaluating multitasking capabilities under temporal constraints. Agents must complete multiple tasks, manage sub-task dependencies, and minimize idle times. The dataset contains 90 samples across household, cooking, and laboratory domains. Each domain contains samples requiring achievement of 1, 2, or 3 goals (tasks). While the original dataset is developed in an online setting with planner-simulator interactions, to enable use for our scenario, we reformulate it to directly provide all valid actions and their durations at the start. This makes the benchmark focused purely on planning, instead of reacting to unexpected observations. The train/-validation split is 27/63 queries, and maintains diversity across domains and difficulty levels. We again use $N = 10$ samples from the training set of 27 to generate reasoning traces. TimeArena-Static reasoning traces turned out to be longer than that of Travelplanner. Hence, we set a smaller $C = 2$ for sample selection (to ensure that these samples can fit within the context window of meta agents. This leaves a few samples from even the smaller $N = 10$ sample set unused) and evaluate on the validation set.

Evaluation Metrics: We assess performance on TimeArena-Static using several evaluation metrics like *full completion rate (FCR)* (proportion of queries where all tasks are completed in the plan) and *task completion rate (TCR)* (fraction of tasks correctly completed aggregated over the whole dataset). *Average progress (AP)* is computed as average of percentage of task completed successfully weighted in terms of duration of successfully completed subtasks. Here t is a subtask of task s. The progress score denotes the percentage of a task completed, measured in terms of total duration of successfully completed subtasks. Other similar metrics include *average completion time (ACT)* (average time taken for completing tasks successfully) and *progress speed (PS)* (sum of highest progress scores achieved by each task, divided by the sum of time taken to achieve them). Other metrics include *Idle violation rate* (total number of times an action was initiated but was invalid, because another non-idle task was underway) and *Dependency violation rate* (total number of times an action was initiated but was invalid, because its prerequisite sub-task(s) was (were) not completed). Lower is better for violation rates and ACT, whereas higher is better for other metrics. Details of each metric are in Appendix E.

6 Experimental Setup

Through our experiments, we seek answers to the following research questions. (1) How does LRPLAN compare with other domain-independent architectures for planning with multiple constraints? (2) What is the incremental value of using an LLM-LRM combination compared to just using an LLM or an LRM? (3) How valuable is each meta-agent in the LRPLAN architecture? (4) What is the nature of errors made by LRPLAN?

Experimental Details. We use LLMs and LRMs from two model families (one proprietary and one open source). We use a combination of GPT-4o and o3-mini as LLM and LRM, respectively. In another setting, we use DeepSeek-V3 with DeepSeek-R1. We run the models using `autogen` library for inference, employing caching and setting temperature to 0.01 for reproducibility. We always use DeepSeek-R1 as the LRM to generate reasoning traces. This is because as of May 2025, o3-mini is a proprietary model which does not offer detailed reasoning traces via its API.

Baselines. For fair comparison in domain-independent planning, we wish to identify baselines that represent state-of-the-art multi-agent frameworks that minimize human intervention in agent configuration and prompt engineering. Hence, we select AutoBuild[2] (Wu et al., 2024) ,generalized multi-agent debate (MAD) (Du et al., 2023), and ExpeL (Zhao et al., 2024) as our baselines. AutoBuild dynamically provisions a team of agents and orchestrates a group chat based on the input task description and query to derive a final response. The Multi-agent debate involves iterative group interaction, typically among homogeneous agents over multiple rounds, to converge on a solution. ExpeL extracts insights from Reflexion (Shinn et al., 2023) traces generated on training samples and infers using these insights via a ReAct (Yao et al., 2022) framework.

These approaches are suitable, as they automate key aspects of agent creation and interaction logic. This contrasts with alternative multi-agent methods that require significant manual effort in defining agent roles or crafting specific prompts for each agent in a domain. We also compare against a vanilla single-agent baseline that queries an LLM

Strategy	Model	Commonsense ↑		Hard Constraints ↑		Final	Cost($) ↓	#API Calls ↓
		Micro	Macro	Micro	Macro	Pass Rate ↑		
Direct	GPT-4o	84.93	29.44	41.67	22.22	3.89	7.10	1
Direct	o3-mini	78.82	17.22	54.29	41.11	8.33	**5.99**	1
CoT	GPT-4o	85.28	31.67	45.71	25.00	7.78	7.36	1
AutoBuild	GPT-4o	86.81	35.56	60.24	36.67	12.78	33.02	5
AutoBuild	o3-mini	78.96	13.33	72.62	**54.44**	5.56	23.23	5
MAD	GPT-4o	79.44	7.78	10.00	6.67	2.78	44.06	6
MAD	o3-mini	77.01	7.78	56.67	40.56	3.33	37.06	6
ExpeL	GPT-4o	51.60	29.44	30.95	18.33	10.00	13.00	3
ExpeL	o3-mini	34.58	13.88	33.57	24.44	8.88	10.86	3
LRPlan (ours)	GPT-4o + o3-mini	**88.61**	**37.22**	**71.19**	45.56	**17.78**	12.41	2
Direct	DeepSeek-V3	80.83	19.44	44.76	23.33	4.44	**0.55**	1
Direct	DeepSeek-R1	77.36	16.11	60.24	45.56	11.11	1.97	1
CoT	DeepSeek-V3	71.60	9.44	16.19	10.56	2.22	0.58	1
AutoBuild	DeepSeek-V3	83.06	21.11	46.43	26.11	5.00	2.03	5
AutoBuild	DeepSeek-R1	81.32	18.89	52.62	50.56	11.67	17.92	5
MAD	DeepSeek-V3	77.01	14.44	25.24	15.00	6.11	5.57	6
MAD	DeepSeek-R1	75.97	8.33	10.71	11.11	6.11	16.18	6
ExpeL	DeepSeek-V3	27.29	13.33	18.33	9.44	4.44	0.64	3
LRPlan (ours)	DeepSeek V3 + DeepSeek R1	**85.90**	**30.00**	**72.14**	**60.56**	**21.67**	3.30	2

Table 2: Performance comparison of different models and methods for the TravelPlanner dataset.[4]

Strategy	Model	Violations ↓		Progress			Completion Rate ↑		Cost($) ↓	#API Calls ↓
		#Idle Viol.	#Dep. Viol.	CT ↓	Prog. speed ↑	Avg Prog. ↑	Task CR	Full CR		
Direct	GPT-4o	50	76	26.47	2.40	69.63	37.30	23.81	**0.30**	1
Direct	o3-mini	3	43	28.00	2.79	88.30	61.90	55.56	1.70	1
CoT	GPT-4o	95	78	**25.60**	2.72	56.81	19.84	9.52	0.45	1
AutoBuild	GPT-4o	70	71	27.62	2.80	68.07	31.71	22.22	4.17	5
AutoBuild	o3-mini	**0**	49	27.96	2.62	88.75	59.20	52.38	8.73	5
MAD	GPT-4o	46	73	27.74	2.71	71.23	36.51	20.63	3.17	6
MAD	o3-mini	**0**	41	28.30	2.90	92.53	68.25	61.90	11.07	6
ExpeL	GPT-4o	77	82	29.42	2.36	63.68	26.19	14.28	3.04	3
ExpeL	o3-mini	**0**	25	31.48	2.84	**95.52**	80.95	73.01	2.97	3
LRPlan (ours)	GPT-4o + o3-mini	1	**17**	31.94	**3.01**	94.64	**83.33**	**74.60**	2.41	2
Direct	DeepSeek-V3	109	95	25.80	2.47	51.89	15.87	14.29	**0.04**	1
Direct	DeepSeek-R1	**31**	44	28.02	2.91	75.70	43.75	25.40	1.06	1
CoT	DeepSeek-V3	80	76	29.55	2.75	60.99	31.75	17.46	0.05	1
AutoBuild	DeepSeek-V3	109	74	29.76	2.62	61.12	29.37	25.40	0.58	5
AutoBuild	DeepSeek-R1	51	51	27.66	3.15	79.39	48.41	34.92	6.45	5
MAD	DeepSeek-V3	107	92	**23.63**	2.77	47.94	15.08	15.87	0.34	6
MAD	DeepSeek-R1	37	55	26.24	**3.32**	75.77	41.07	25.40	8.12	6
ExpeL	DeepSeek-V3	43	58	27.70	2.84	76.17	50.00	31.75	0.25	3
LRPlan (ours)	DeepSeek V3 + DeepSeek R1	33	**39**	27.84	3.14	**89.95**	**68.25**	**53.97**	0.96	2

Table 3: Performance comparison of different models and methods for the TimeArena-Static dataset.[4]

or an LRM directly (with or without chain of thought).

7 Results

7.1 Main results

Tables 2 and 3 present a comparison between performances of direct prompting, chain of thought, and other MAS methods across two different family of models, OpenAI GPT-4o and o3-mini, and DeepSeek V3 and R1. The results show that LRPlan can generalize across various domains and backbone language models with limited training on a domain and give considerable accuracy gains at a much lower inference cost. Additionally from the detailed performance of other methods, we can infer that LLMs, GPT-4o and DeepSeek-V3, have better scores on commonsense constraints than their LRM counterparts, o3-mini and DeepSeek-R1, when using the same methods. Equivalently, the LRMs show better performance on hard constraints than the LLMs.[4]

Planner	Refiner	Commonsense ↑		Hard Constraints ↑		Final
		Micro	Macro	Micro	Macro	Pass Rate ↑
GPT-4o	GPT-4o	**89.93**	**43.89**	63.81	35.56	15.56
o3-mini	o3-mini	80.97	20.00	**74.52**	**57.78**	11.11
o3-mini	GPT-4o	81.46	20.56	70.71	53.33	10.00
GPT-4o	o3-mini	88.61	37.22	71.19	45.56	**17.78**
DeepSeek-V3	DeepSeek-V3	**85.90**	**31.11**	72.86	57.78	16.67
DeepSeek-R1	DeepSeek-R1	83.61	21.11	**75.95**	**66.11**	16.11
DeepSeek-R1	DeepSeek-V3	83.19	19.44	74.29	63.33	14.44
DeepSeek-V3	DeepSeek-R1	**85.90**	30.00	72.14	60.56	**21.67**

Table 4: Different LLM/LRM combinations for Planner/Refiner for TravelPlanner

---

[4] Results are not available for ExpeL on DeepSeek-R1 due to incompatibility of the ExpeL system with the DeepSeek-R1 API, which does not allow consecutive calls with the role 'system' or 'assistant' but is used in the ExpeL system.

Planner	Refiner	Violations ↓		Progress			Completion Rate ↑	
		#Idle Viol.	#Dep. Viol.	CT ↓	Prog. speed ↑	Avg Prog. ↑	Task CR	Full CR
GPT-4o	GPT-4o	34	49	**30.31**	2.65	83.15	53.17	41.27
o3-mini	o3-mini	**1**	23	31.47	2.90	93.69	80.16	**74.60**
o3-mini	GPT-4o	18	27	31.91	2.73	91.00	73.02	65.08
GPT-4o	o3-mini	**1**	**17**	31.94	**3.01**	**94.64**	**83.33**	**74.60**
DeepSeek-V3	DeepSeek-V3	44	44	31.16	2.91	85.11	64.29	50.79
DeepSeek-R1	DeepSeek-R1	35	46	28.68	3.07	86.38	62.70	46.03
DeepSeek-R1	DeepSeek-V3	39	50	31.03	2.90	81.85	57.14	39.68
DeepSeek-V3	DeepSeek-R1	**33**	**39**	**27.84**	**3.14**	**89.95**	**68.25**	**53.97**

Table 5: Different LLM/LRM combinations for Planner/Refiner for TimeArena-Static

7.2 Benefit of LLM-LRM synergy

Tables 4 and 5 show that during inference of LRPLAN, keeping planner agent as LLM and refiner agent as LRM, in that order, gives the best performance. Table 4 shows that keeping both agents as LLMs gives best commonsense scores, while keeping both agents as LRMs gives best hard constraints score. It shows that best results are achieved using the strengths of both types of models. Even using an LRM planner and an LLM refiner results in worse performance, showcasing poor consistency enforcement of LLMs as refiners. In the best scenario, the LLM first creates a plan which enforces commonsense and then the LRM refines the plan, enforcing consistency with the hard constraints. Table 6 also reaffirms this observation as comparing with and without refiner agent in LRPLAN, we can see that refiner introduced some commonsense errors but gives a larger boost in hard constraints, overall giving best results.

While it is true that many LRMs are initialized from LLM checkpoints, the training objectives and data distributions diverge significantly. LRMs are typically fine-tuned with a strong emphasis on explicit reasoning traces, multi-step logical deduction, and verifiability. This may come at the cost of the flexibility, creativity, and contextual fluency that LLMs exhibit: traits that are crucial for satisfying soft constraints such as diversity, user preferences, or commonsense expectations. In our experiments, we observed that: 1) LLMs tend to generate plans that are more natural, diverse, and aligned with human preferences, but often violate hard constraints (e.g., accommodation rules, budget limits) and 2) LRMs, in contrast, produce plans that are more logically consistent and rule-abiding, but sometimes lack the nuance or adaptability needed to satisfy softer, implicit constraints. This divergence is a reflection of specialization: LRMs are optimized for structured reasoning, while LLMs retain broader linguistic and commonsense priors. Our architecture leverages this complementarity by assigning creative abstraction and pattern recognition to LLMs and analytical validation and refinement to LRMs. A similar yet different observation, that LLMs perform better than LRMs on simpler, low-compositional problems, is made by another recent work, Illusion of Thinking (Shojaee* et al., 2025).

7.3 Ablation Study for LRPLAN

From Tables 6 and 7 we can see that all meta-agents contribute to the final accuracy, although incrementally in some cases. Pattern Recognizer identifies patterns in the reasoning traces during training which can guide the Planner in its process to formulate a plan during test time. The Rule Extractor creates actionable rules from these patterns during training time, which the Planner follows to ensure correctness of a generated plan. The Self-Corrector identifies and collates possible mistakes and possible refinement paths/insights during training guiding the refinement of plan by the Refiner during test time. Combined with very good cost and time efficiency of training, we can conclude that it is a good idea to keep all the meta-agents for best performance. Similarly, even the removal of Refiner causes siginificant effect on the metrics, denoting its importance in LRPLAN.

7.4 Error Analysis

We conducted a detailed error analysis of LRPLAN across both the TravelPlanner and TimeArena-

	Commonsense ↑		Hard Constraints ↑		Final
	Micro	Macro	Micro	Macro	Pass Rate ↑
GPT-4o + o3-mini					
Full LRPLAN	88.61	37.22	**71.19**	**45.56**	**17.78**
w/o pattern recognizer	88.33	38.89	68.57	**45.56**	**17.78**
w/o rule extractor	88.06	37.78	69.05	43.33	16.11
w/o self corrector	89.10	40.00	69.29	43.33	17.22
w/o refiner	**90.35**	**45.56**	65.71	36.67	15.56
DeepSeek-V3 + DeepSeek-R1					
Full LRPLAN	85.90	30.00	72.14	60.56	**21.67**
w/o pattern recognizer	85.07	27.78	74.05	**65.56**	18.89
w/o rule extractor	**86.25**	**31.67**	73.10	60.00	21.11
w/o self corrector	**86.25**	31.11	**75.48**	60.56	18.33
w/o refiner	85.56	30.56	72.14	57.78	16.67

Table 6: Ablation Study for TravelPlanner

	Violations ↓		Progress			Completion Rate ↑	
	#Idle Viol.	#Dep. Viol.	CT ↓	Prog. speed ↑	Avg Prog. ↑	Task CR	Full CR
GPT-4o + o3-mini							
Full LRPLAN	**1**	**17**	31.94	**3.01**	**94.64**	**83.33**	**74.60**
w/o PR	2	61	**25.52**	2.95	82.96	47.62	36.51
w/o RE	**1**	37	27.13	2.98	90.38	68.25	60.32
w/o SC	5	28	31.54	2.84	91.28	75.40	66.67
w/o refiner	31	44	30.07	2.70	83.90	57.14	41.27
DeepSeek-V3 + DeepSeek-R1							
Full LRPLAN	33	**39**	**27.84**	3.14	**89.95**	68.25	**53.97**
w/o PR	41	**39**	28.00	3.18	88.93	**69.84**	49.21
w/o RE	**29**	40	28.77	**3.22**	89.87	65.85	49.21
w/o SC	39	44	28.06	3.13	87.85	63.49	47.62
w/o refiner	30	41	30.08	2.96	87.71	66.67	57.14

Table 7: Ablation Study for TimeArena-Static (PR=pattern recognizer, RE=rule extractor, SC=self corrector)

Static domains (see Tables 8 and 9 in Appendix F). In the TravelPlanner domain, the most frequent error involved violations of the "minimum nights stay" constraint, where the generated plans did not always ensure that each accommodation met the required minimum stay. Additionally, LRPLAN occasionally failed to enforce commonsense constraints regarding transportation modes, sometimes including both self-driving and flight segments within a single itinerary which is an impractical scenario since a self-driven vehicle cannot be abandoned mid-trip. The system also exhibited difficulties in consistently honoring traveler cuisine preferences, particularly when balancing these preferences with budget constraints. Table 11 shows evaluation across difficulty levels for TravelPlanner.

Both variants of LRPLAN (GPT-4o + o3-mini and DeepSeek-V3 + DeepSeek-R1) demonstrate balanced performance across commonsense and hard constraints, as shown in Fig. 3. This suggests that the multi-agent collaboration and LLM-LRM synergy effectively generalize across constraint types, unlike other strategies that tend to skew toward one.

For the TimeArena-Static domain, our analysis revealed that while LRPLAN achieves strong results in the Cooking subdomain and performs reasonably well in Laboratory tasks, its efficiency in the Household subdomain remains comparable to baseline methods and is suboptimal. Also, there is room for improvement in reducing the average time required for successful task completion. Despite being a widely used prompting strategy, Chain of Thought performs poorly, especially in FCR (e.g., <14% across domains). This highlights that reasoning traces alone are insufficient without structured correction mechanisms like those in LRPLAN. Table 10 shows evaluation across difficulty levels for TimeArena. LRPLAN maintains high TCR and FCR across Easy, Medium, and Hard tasks, unlike other strategies that show steep performance drops as difficulty increases.

Finally, we provide analysis of how does the iterative process of baselines (AutoBuild and Multi-Agent Debate) perform in Appendices G and H respectively. We perform this analysis across different constraints and metrics, and LLMs.

8 Conclusion

We explored the capabilities of language models in solving complex multi-constraint planning problems, introducing a novel multi-agent architecture LRPLAN, which trains over a domain using very few training instances and creates a streamlined, simple and cost-efficient inference planning process for new instances of the domain. We reformulate TimeArena (Zhang et al., 2024) to a static setting for efficient and easier inference and explore the planning domains TravelPlanner (Xie et al., 2024) and TimeArena-Static using various methods, standard prompting, CoT, MAD (Du et al., 2023), and AutoBuild (Wu et al., 2024), with both LLMs and LRMs. We found that LLMs show poor results in adhering to explicit constraints while showing better results in implicit ones. On the other hand, LRMs show poor results for implicit constraints, being good at explicit constraints. We show that the complementary nature of LLMs and LRMs can be leveraged in a synergistic system to produce best results.

9 Future Work

There are several promising avenues to extend this work. We plan to broaden LRPLAN from static to online planning, where the planner interacts iteratively with an environment and adapts to new observations, enabling more realistic and uncertain scenarios. Another direction is exploring LRPLAN's integration with formal solvers and external tools (Mittal et al., 2024), combining the flexibility of language-based reasoning with the rigor of symbolic verification. We also aim to study selective incorporation of meta-agent insights, where voting or consensus mechanisms admit only high-confidence rules and corrections, reducing noise and improving reliability. These extensions can further enhance LRPLAN's robustness and applicability across diverse planning domains.

Acknowledgements

The work is supported by an IBM AI Horizons Network grant, and grants from Verisk, Huawei, Google, and Microsoft. We would also like to thank the IIT-D HPC facility for its computational resources. We are grateful to Microsoft Accelerate Foundation Models Research (AFMR) program that provided us access to OpenAI models. We thank Yardi School of AI publication grant for travel funds. Om Dehlan also acknowledges the supportive and friendly working environment at DAIR Lab at IIT Delhi that encouraged him to pursue serious research as an undergraduate student.

Limitations

This work has been performed on English data only. Given that state of the art LLMs and LRMs can handle other popular languages nicely as well, we believe that the proposed system, LRPLAN, should be effective on multi-lingual tasks as well but that needs to rigorously tested.

We worked with tasks across few domains like travel (TravelPlanner) and household, cooking, and laboratory (TimeArena-Static). The effectiveness of LRPLAN needs to be further explored for other domains. While LRPLAN is domain-independent in principle, its performance may degrade in domains with highly diverse or ambiguous implicit constraints, especially if the training data is sparse or unrepresentative.

LRPLAN is currently limited to static planning scenarios. Static (or classical) planning serves as a foundational setting commonly explored as a first step when studying a planning scenario, before further extensions to more complex scenarios are considered. As part of future work, we plan to extend our approach to more realistic settings, including probabilistic planning, multi-objective planning, and dynamic adaptation.

Deeper LRM–LLM dynamics, as to why LRMs become worse at commonsense after reasoning training, are out of scope; we leverage their current complementary strengths. This is a rich area for future work and plan to explore whether joint training or hybrid objectives can better unify these capabilities.

The effectiveness of LRPLAN hinges on the quality and diversity of reasoning traces generated during training. If the traces are noisy, incomplete, or biased, the downstream pattern and rule extraction may be suboptimal.

The architecture assumes that LLMs and LRMs can collaborate effectively. However, differences in their internal representations or reasoning styles might lead to inconsistencies or inefficiencies in coordination.

Although LRPLAN is more cost-efficient than some baselines, the multi-agent architecture still involves multiple model invocations (planner, refiner, meta-agents), which could introduce latency and computational overhead in real-time applications.

Ethics Statement

The system may generate incorrect plans with hallucinations. Users may over-trust the system's outputs, assuming that the plans are optimal or error-free, even when implicit constraints are misunderstood or missed. Hence, we expect all deployments of this system to build post-processing verifiers (automated or human) to ensure correctness of these plans before exposing them to real users.

The LLMs and LRMs used in LRPLAN may inherit biases from their training data, which could lead to unfair or discriminatory plans. A travel plan might consistently recommend more expensive or Western-centric destinations, ignoring budget-friendly or culturally diverse options.

The datasets employed in our study are distributed under the following licenses:

- **TravelPlanner**: MIT License
- **TimeArena**: Publicly available without any specific license

The models have been utilized in our experiments as APIs as per their terms of intended use.

Both of our datasets have been cleaned for personally identifiable information at source itself.

References

Blai Bonet and Héctor Geffner. 2001. Planning as heuristic search. *Artif. Intell.*, 129(1–2):5–33.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. 2024a. Autoagents: a framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 22–30.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024b. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.

Yanan Chen, Ali Pesaranghader, Tanmana Sadhu, and Dong Hoon Yi. 2024c. Can we rely on llm agents to draft long-horizon plans? let's take travelplanner as an example. *ArXiv*, abs/2408.06318.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 179 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *ArXiv*, abs/2501.12948.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Ho-Sun Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. In *Neural Information Processing Systems*.

M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. 1998. PDDL—The Planning Domain Definition Language.

Atharva Gundawar, Mudit Verma, L. Guan, Karthik Valmeekam, Siddhant Bhambri, and Subbarao Kambhampati. 2024. Robust planning with llm-modulo framework: Case study in travel planning. *ArXiv*, abs/2405.20625.

Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. 2025a. Large language models can solve real-world planning rigorously with formal verification tools. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3434–3483, Albuquerque, New Mexico. Association for Computational Linguistics.

Yilun Hao, Yang Zhang, and Chuchu Fan. 2025b. Planning anything with rigor: General-purpose zero-shot planning with LLM-based formalized programming. In *The Thirteenth International Conference on Learning Representations*.

Malte Helmert. 2006. The fast downward planning system. *ArXiv*, abs/1109.6051.

Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier, and Ron Alford. 2020. Hddl: An extension to pddl for expressing hierarchical planning problems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 9883–9891.

Andrew J Hughes and Barbara J Rutherford. 2013. Hemispheric interaction, task complexity, and emotional valence: Evidence from naturalistic images. *Brain and cognition*, 81(2):167–175.

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*.

Kuang-Huei Lee, Ian Fischer, Yueh-Hua Wu, David Marwood, Shumeet Baluja, Dale Schuurmans, and Xinyun Chen. 2025. Evolving deeper llm thinking. *ArXiv*, abs/2501.09891.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651.

Chinmay Mittal, Krishna Kartik, Mausam, and Parag Singla. 2024. Fcorebench: Can large language models solve challenging first-order combinatorial reasoning problems? *CoRR*, abs/2402.02611.

Yaswanth Narsupalli, Abhranil Chandra, Sreevatsa Muppirala, Manish Gupta, and Pawan Goyal. 2025. Refer: Improving evaluation and reasoning through hierarchy of models. *Transactions of Machine Learning Research (TMLR)*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*.

Parshin Shojaee*, Iman Mirzadeh*, Keivan Alizadeh, Maxwell Horton, Samy Bengio, and Mehrdad Farajtabar. 2025. The illusion of thinking: Understanding the strengths and limitations of reasoning models via the lens of problem complexity.

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. 2024. Adaptive in-conversation team building for language model agents. *ArXiv*, abs/2405.19425.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2024. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 257–279, Mexico City, Mexico. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. Travelplanner: A benchmark for real-world planning with language agents. In *International Conference on Machine Learning*, pages 54590–54613. PMLR.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629.

Cong Zhang, Xin Deik Goh, Dexun Li, Hao Zhang, and Yong Liu. 2025. Planning with multi-constraints via collaborative language agents. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 10054–10082, Abu Dhabi, UAE. Association for Computational Linguistics.

Yikai Zhang, Siyu Yuan, Caiyu Hu, Kyle Richardson, Yanghua Xiao, and Jiangjie Chen. 2024. TimeArena: Shaping efficient multitasking language agents in a time-aware simulation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3894–3916, Bangkok, Thailand. Association for Computational Linguistics.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pages 19632–19642. AAAI Press.

Overview of Appendices

- Appendix A: Detailed Related Work.
- Appendix B: Walkthrough of LRPLAN
- Appendix C: Meta Agent Prompts.
- Appendix D: Meta Agent Outputs
- Appendix E: Evaluation Metrics for TimeArena-Static.
- Appendix F: Error Analysis.
- Appendix G: Analyzing AutoBuild.
- Appendix H: Analyzing Multi-Agent Debate.
- Appendix I: TimeArena Case Study.

A Detailed Related Work

A.1 Classic Planning Literature

Traditional approaches to multi-constraint planning relied on classical, domain-independent planners and optimization techniques. These methods, often utilizing formalisms (Ghallab et al., 1998; Höller et al., 2020) like the Planning Domain Definition Language (PDDL), enabled the generation of hierarchial and partially ordered plans. PDDL standardized AI planning languages by separating domain descriptions from problem-specific details, allowing planners to tackle a wide range of domains. Classical planning (Helmert, 2006; Bonet and Geffner, 2001) typically addressed problems with known initial states, deterministic actions, and a single agent. Many planning problems can indeed be viewed as optimization problems, seeking an optimal solution under given constraints. To alleviate the significant human effort required in crafting domain models in PDDL and to enhance adaptability, researchers turned to supervised machine learning and imitation learning, these techniques aimed to learn action strategies for specific planning domain from examples of problems and their solutions. However, a primary drawback was the need for domain-dependent training data or the manual creation of domain files, limiting broad applicability.

A.2 Single-agent LLMs for Planning

The advent of LLMs introduced new paradigms for planning. LLMs are increasingly explored as single-agent planners, leveraging techniques like zero-shot prompting and chain-of-thought (CoT) (Wei et al., 2022) reasoning to generate plans without task-specific training. In online settings, where agents interact with an environment and receive feedback to adapt their plans, several frameworks have emerged. ReAct (Yao et al., 2022) integrates reasoning and action steps. Reflexion (Shinn et al., 2023) uses environmental feedback for iterative plan improvement and episodic memory updates. Given that LLMs are trained for next-token prediction, their initial plan may require refinement, a process addressed by frameworks like Self-Refine (Madaan et al., 2023) where models iteratively critique and improve their own inputs.

A.3 Multi-agent systems

Multi-agent systems (MAS) offer a collaborative approach to solving complex tasks and have found success in diverse areas such as web-navigation, e-commerce, question answering, and robotics. Frameworks like AutoGen's AutoBuild (Wu et al., 2024) and AutoAgents (Chen et al., 2024a) automate the creation and coordination of specialized agent teams tailored to task descriptions. AutoAgents, for instance dynamically generates required agents and includes an observer roles for reflection and improvement. A potential drawback of dynamically creating new agent sets for every query is redundancy and difficulty in learning from cumulative past experiences. The multi-agent debate (Du et al., 2023) paradigm involves multiple, often identical, LLM agents proposing and critiquing solutions iteratively to converge on a refined answer, enhancing factuality and reasoning. AgentVerse (Chen et al., 2024b) structures multi-agent problem-solving into four stages: expert recruitment, collaborative decision-making, action execution, and evaluation with feedback for iterative refinement. LLMFP (Hao et al., 2025b) is a framework where LLMs convert planning problems into formal optimization problems for dedicated solvers. This approach assumes that all the constraints will be explicitly stated which might not be the case always. ReFeR (Narsupalli et al., 2025) is a tuning-free framework that evaluates generative outputs in both text and images by leveraging a hierarchical multi-agent strategy with pre-trained LLM and VLM evaluators, enhancing performance through increased test-time reasoning.

A.4 Role-based Multi-agent systems

Collaborative LLM-based multi-agent planning frameworks further explore specialized roles. For instance, the LLM-Modulo (Kambhampati et al., 2024) framework is conceptualized with multiple critics for different analytical purposes. The Planning with Multi-Constraints (PMC) (Zhang et al., 2025) framework employs a manager agent for hierarchical task decomposition into a sub-task graph, executor agents to handle these sub-tasks, a supervisor agent for information sharing, and a deliverer agent to consolidate final results. Both PMC and LLM-Modulo necessitate initial human effort for each new domain, particularly in defining agent roles, available tools, and tailoring prompt structures for the agents to function effectively within that specific domain's constraints.

A.5 Learning from experience in Multi-agent systems

Learning from experience is a critical theme in recent LLM-based planning research. Reflexion (Shinn et al., 2023) highlights the importance of updating episodic memory by reflecting on past attempts to understand constraints and improve future performance. ExpeL (Zhao et al., 2024) enables agents to learn from a collection of experiences across multiple instances, deriving insights and using successful past trajectories as in-context examples. AutoGuide (Fu et al., 2024) automatically generates context-aware guidelines from offline trajectories to aid decision-making. These approaches underscore a trend towards enabling agents to continuously learn and adapt from their interactions.

A.6 Characteristics used for comparison in Table 1

- `No Validator Needed during inference`: Indicates whether the method can generate valid outputs without relying on an external validator at test time.
- `Refines Initial Response`: Specifies if the method includes a mechanism to revise or improve its initial output through self-correction or refinement.
- `Multi-Agent`: Denotes whether the approach uses multiple interacting agents or components to collaboratively solve the task.
- `No Domain-Specific Human Effort`: Shows whether the method avoids manual engineering or human intervention tailored to specific domains.
- `Learns from Training Samples`: Indicates if the method leverages any training data to extract insights for better performance.
- `LLM-LRM Synergy`: Highlights whether the method explicitly combines large language models (LLMs) and large reasoning models (LRMs) to exploit their complementary strengths.

B Walkthrough of LRPLAN

The LRPLAN framework is designed to enhance the planning capabilities of Large Language Models (LLMs) by learning from reasoning traces. The architecture, depicted in Fig. 2, operates in two distinct phases: a Training Phase where the system learns successful strategies, rules, and error-correction patterns, and a Test Phase where these learned components are applied to solve new planning queries.

B.1 Training Phase: Learning from Reasoning

The training process begins with a set of Training Samples, each containing an input query, a generated plan, a binary PASS/FAIL for the generated plan, and a corresponding reasoning trace. These elements are used to train three specialized modules.

B.1.1 Data Generation and Validation

For each training sample, a Large Reasoning Model (LRM) generates a step-by-step Reasoning Trace that explains its thought process for creating a Plan. For instance, given Train Sample 1:

> Input: Please help me plan a trip from St. Petersburg to Rockford spanning 3 days from March 16th to March 18th, 2022. The travel should be planned for a single person with a budget of $1,700.

The LRM produces both the detailed trace and the plan:

> Reasoning Trace: Okay, let's tackle this query step by step. The user wants a 3-day itinerary from St. Petersburg to Rockford from March 16th to 18th, 2022, for a single person with a $1,700 budget. Let's see the reference info provided. First, check transportation options. There's a flight from St. Petersburg to Rockford on March 16th (F3573659) priced at $474. The return flight on March 18th (F3573120) is $346. The total flight cost is $820. Since self-driving and taxi options have no valid info, flights are the only viable option . . .
> . . .
> . . .
> . . .
> Day 1: After arrival, maybe visit a couple of attractions. For example, Burpee Museum and Discovery Center. Day 2: More attractions like Midway Village Museum, Anderson Japanese Gardens, etc. Day 3: Maybe one attraction in the morning before departure. Putting it all together, ensuring all data is from reference info.

> Plan:
> Day 1 (from St. Petersburg → Rockford): Transportation: Flight F3573659 (Dep. 15:40, Arr. 17:04) Dinner: Coco Bambu, Rockford Accommodation: Private Room in a two-bedroom apartment, Rockford Attractions: Burpee Museum of Natural History; Discovery Center Museum
> Day 2 (Rockford): Breakfast: Cafe Coffee Day, Rockford Lunch: Flying Mango, Rockford Dinner: The Gourmet Shack, Rockford Accommodation: Private Room in a two-bedroom apartment, Rockford Attractions: Midway Village Museum; Anderson Japanese Gardens; Tinker Swiss Cottage Museum and Gardens
> Day 3 (from Rockford → St. Petersburg): Transportation: Flight F3573120 (Dep. 19:00, Arr. 22:43) Breakfast: Subway, Rockford Lunch: Aroma Rest O Bar, Rockford Attraction: Nicholas Conservatory & Gardens

The generated plan is then evaluated by a Validator function, $V{:}\text{Plan} \rightarrow \{\text{pass},\text{fail}\}$, which labels the plan's adherence to all explicit and implicit constraints. For Train Sample 1, the plan successfully met all requirements, resulting in a PASS label. Conversely, Train Sample 2 failed validation due to constraint violations, receiving a FAIL label. This collection of samples, traces, and validation labels serves as the foundational dataset for the learning modules.

B.1.2 Learning Modules

The annotated training data is concatenated and fed into three LLM-based modules that learn distinct aspects of the planning process:

Pattern Recognizer: This module primarily analyzes successful reasoning traces (i.e., those leading to a PASS validation) to identify high-level, winning strategies. It abstracts the general workflow from specific examples.

> Sample traces generally proceed through a sequence of steps: identifying requirements (dates, budget, destination), evaluating transportation, selecting accommodation... High-level principles include prioritizing budget constraints, sequential decision-making, adherence to reference information, and iterative cost verification.

Rule Extractor: This module also primarily processes successful traces but focuses on distilling concrete, actionable heuristics or rules that can be explicitly followed from the output of the **Pattern Recognizer**.

> Key heuristics emerge... First, planners must prioritize budget alignment... Second, they should follow a sequential decision-making process—transportation → accommodation → meals → attractions. Third, iterative budget checks after major decisions help avoid misalignment.

Self Corrector: This module learns from both successful and failed traces to identify common error patterns and the self-correction steps taken within the reasoning process. It learns how to detect and fix mistakes.

> Reasoning traces reveal frequent initial errors such as miscalculations, incorrect assumptions, and logical inconsistencies... These are typically resolved through self-corrections, including revisiting assumptions, recalculating costs, and re-evaluating decisions against reference information.

B.2 Test Phase: Planning and Refinement

During the test phase, the insights gained during training are operationalized to solve new, unseen queries.

B.2.1 Planner

When a New Query is received, it is first sent to the Planner module. This LLM is prompted using the high-level strategies from the **Pattern Recognizer** and the explicit heuristics from the **Rule Extractor**. This guidance enables it to generate a structured and logical initial plan.

> Test Input: Please create a travel plan for me where I'll be departing from Washington and heading to Myrtle Beach for a 3-day trip from March 13th to March 15th, 2022. Can you help me keep this journey within a budget of $1,400?

The Planner, guided by the learned rules, produces a coherent initial plan:

> Day 1 (from Washington → Myrtle Beach): Transportation: Flight F3927581 (Dep. 11:03, Arr. 13:31) Lunch: Catfish Charlie's Dinner: Exotic India Accommodation: Yellow Submarine Attractions: SkyWheel, WonderWorks
> Day 2 (Myrtle Beach): Breakfast: First Eat Lunch: Nagai Dinner: Twigly Accommodation: Yellow Submarine Attractions: Ripley's Aquarium, Broadway at the Beach
> Day 3 (from Myrtle Beach → Washington): Transportation: Flight F3791200 (Dep. 11:36, Arr. 13:06) Breakfast: The Night Owl

B.2.2 Refiner

The initial plan is then passed to the Refiner module. This LRM leverages the knowledge from the **Self Corrector** module. It proactively scans the plan for common error types (e.g., budget miscalculations, logical inconsistencies, constraint violations) identified during training and applies the learned correction strategies to produce a more robust and accurate Refined Plan. This final output is then presented to the user.

Day 1 (from Washington → Myrtle Beach): Transportation: Flight F3927581 (Dep. 11:03, Arr. 13:31) Lunch: Catfish Charlie's Dinner: Exotic India Accommodation: Yellow Submarine Attractions: SkyWheel, WonderWorks
Day 2 (Myrtle Beach): Breakfast: First Eat Lunch: Nagai Dinner: Twigly Accommodation: Yellow Submarine Attractions: Ripley's Aquarium, Broadway at the Beach
Day 3 (from Myrtle Beach → Washington): Breakfast: The Night Owl Lunch: Quick meal at the airport before departure Transportation: Flight F3791200 (Dep. 11:36, Arr. 13:06)

C Meta Agent Prompts

This section presents the prompts used for the three meta agents in our framework: Pattern Recognizer, Rule Extractor, and Self-Corrector. These prompts were carefully designed to enable the agents to extract insights from reasoning traces and support the planning process.

C.1 Pattern Recognizer Agent Prompt

Pattern Recognizer

```
You are an expert in recognizing patterns and structures in logical reasoning traces. Your goal is to
analyze multiple instances of reasoning traces and identify recurring themes, decision flows, and
common problem-solving steps.

## Instructions:
1. **Segment Analysis**: Break down each reasoning trace into key logical steps.
2. **Pattern Detection**: Identify frequently occurring sequences, transitions, or dependencies in
the reasoning.
3. **Comparison**: Compare reasoning traces to detect similarities in decision-making strategies.
4. **Abstraction**: Generalize patterns into high-level principles that can be applied across
different contexts.
5. **Edge Case Identification**: Detect cases where the reasoning process struggles, including
ambiguous scenarios, conflicting evidence, and novel problem structures.
6. **Success and Failure Analysis**: Compare reasoning traces with correct and incorrect verdicts to
identify factors contributing to successful reasoning ("success recipe").
7. **Output Format**:
    - **Identified Patterns**: List of common reasoning steps and their frequency.
    - **Edge Cases**: Difficult scenarios and their characteristics.
    - **Success Recipe**: Identifiable patterns in correct reasoning traces that lead to accurate
    responses.
    - **Examples**: Instances from the traces where the patterns occur.
    - **Observations**: Any deviations or anomalies in reasoning.

Ensure that the identified patterns are clear, interpretable, and useful for extracting decision
heuristics.

Here are the sample reasoning traces for your analysis:
<<<sample_input_reasoning_trace>>>
```

C.2 Rule Extractor Agent Prompt

Rule Extractor

```
You are an expert in extracting decision-making rules from structured reasoning patterns. Your goal is
to convert observed reasoning patterns into explicit heuristics that can be applied to future problems.

## Instructions:
1. **Pattern Input**: Given a set of identified patterns from the Pattern Recognizer Agent, infer the
underlying rules that guide decision-making.
2. **Heuristic Formation**: Convert patterns into well-defined, generalizable rules.
```

```
3. **Condition-Based Rules**: Specify conditions under which each rule is applied.
4. **Efficiency Evaluation**: Assess whether the extracted rule optimizes problem-solving efficiency.
5. **Output Format**:
    - **Rule Description**: A concise statement describing the heuristic.
    - **Supporting Patterns**: Reference to identified patterns that led to the rule.
    - **Example Application**: A brief example illustrating the rule in action.

Your heuristics should be logically sound, interpretable, and effective for improving decision making
in similar scenarios.

Here are the identified research patterns for rule extraction:
<<<identified_research_patterns>>>
```

C.3 Self Corrector Agent Prompt

Self Corrector

```
You are an expert in iterative self-improvement, specializing in analyzing reasoning traces to extract
insights from self-correction processes. Your goal is to track instances where the reasoning model
self-corrects, identify the nature of these corrections, and refine future decision-making to
minimize errors.

## **Instructions:**
1. **Correction Detection**: Identify points in the reasoning trace where self-correction occurs.
2. **Error Categorization**: Classify the types of mistakes being corrected (e.g., logical
inconsistencies, incorrect assumptions, miscalculations).
3. **Correction Strategies**: Extract patterns in how corrections are made (e.g., revisiting
assumptions, adjusting calculations, re-evaluating premises).
4. **Cumulative Learning**: Maintain a repository of frequently occurring errors and corresponding
self-correction strategies.
5. **Refinement Mechanism**: Suggest proactive adjustments to prevent similar mistakes in future
reasoning.
6. **Output Format**:
    - **Common Errors**: List of frequently occurring errors and their nature.
    - **Correction Strategies**: General strategies used to fix these errors.
    - **Heuristic Refinements**: How these self-corrections can be integrated into a proactive
    decision-making process.

Your role is to **accumulate self-correction insights** and refine the reasoning process **to reduce
reliance on post-hoc corrections**. Ensure that extracted strategies improve overall robustness
and efficiency.

Here are the sample reasoning traces for your analysis:
<<<sample_input_reasoning_trace>>>
```

D Meta Agent Outputs

This section presents the detailed outputs from our meta agents across TravelPlanner and TimeArena-Static.

D.1 TravelPlanner

We include comprehensive meta agent responses generated by LRPLAN with DeepSeek-V3 and DeepSeek-R1.

Pattern Recognizer

```
### Identified Patterns

1. **Initial Setup and Constraints Check**:
   - **Frequency**: High (100% of traces)
   - **Description**: The reasoning starts by identifying the task's constraints (e.g., budget,
   duration, preferences) and verifying available options from the reference data.
   - **Example**:
    - Trace #1: Checks budget ($1,700), duration (3 days), and transportation options (flight, taxi,
      self-driving).
     - Trace #2: Checks budget ($900), duration (3 days), and transportation options (flight, taxi,
    self-driving).

2. **Transportation Selection**:
   - **Frequency**: High (100% of traces)
   - **Description**: The reasoning evaluates transportation options (flight, taxi, self-driving)
   based on cost, availability, and budget constraints. Flights are prioritized if available and
   affordable; otherwise, self-driving is chosen.
   - **Example**:
     - Trace #1: Selects flights due to availability and budget fit.
     - Trace #2: Selects self-driving due to no flights and taxi being too expensive.

3. **Accommodation Selection**:
   - **Frequency**: High (100% of traces)
   - **Description**: The reasoning picks the cheapest viable accommodation for the required nights,
   ensuring it fits the remaining budget after transportation.
   - **Example**:
     - Trace #1: Chooses the cheapest Private Room in Rockford for two nights.
     - Trace #2: Chooses the cheapest "Amazing apartment" in Pensacola for two nights.

4. **Meal Planning**:
   - **Frequency**: High (100% of traces)
   - **Description**: The reasoning selects restaurants for each meal (breakfast, lunch, dinner) based
   on average cost and cuisine preferences (if specified). It skips meals if timing conflicts with
   travel.
   - **Example**:
     - Trace #1: Skips lunch on Day 1 due to flight timing; selects affordable restaurants for other
     meals.
     - Trace #2: Plans meals for all days but struggles with high-cost options.

5. **Attraction Selection**:
   - **Frequency**: High (100% of traces)
   - **Description**: The reasoning includes free or low-cost attractions each day, ensuring they align
   with the city and timing.
   - **Example**:
     - Trace #1: Visits Burpee Museum and Discovery Center on Day 1.
     - Trace #2: Plans visits to Historic Pensacola Village and Pensacola Museum of Art.

6. **Budget Tracking**:
   - **Frequency**: High (100% of traces)
   - **Description**: The reasoning continuously updates the remaining budget after accounting for
   transportation, accommodation, and meals.
   - **Example**:
     - Trace #1: Tracks expenses ($820 flights, $210 accommodation, $289 meals) to ensure $1,700 budget
     is not exceeded.
     - Trace #2: Tracks expenses ($143 transportation, $148 accommodation, ~$200 meals) but misses some
     costs.

7. **Output Structuring**:
   - **Frequency**: High (100% of traces)
   - **Description**: The reasoning formats the output into daywise dictionaries with keys like
   `current_city`, `transportation`, `meals`, `accommodation`, and `attractions`.
   - **Example**:
     - Both traces structure the output as per the required format.

### Edge Cases

1. **Missing or Conflicting Reference Data**:
   - **Description**: When reference data is missing (e.g., no flights, no restaurants in departure
   city), the reasoning struggles to fill gaps.
   - **Example**:
     - Trace #1: Skips lunch on Day 1 because no St. Petersburg restaurants are listed.
     - Trace #2: Incorrectly assumes no flights are available (may be due to incomplete reference data).

2. **High-Cost Items**:
   - **Description**: When meal or accommodation costs are high, the reasoning may exceed the budget or
   make suboptimal choices.
   - **Example**:
     - Trace #2: Selects expensive restaurants (e.g., Blue Point Grill at $75) without proper budget
     checks.
```

```
3. **Travel Timing Conflicts**:
  - **Description**: Meal or attraction planning becomes ambiguous when travel times conflict with
   typical meal times.
   - **Example**:
     - Trace #1: Skips lunch on Day 1 due to flight timing.
     - Trace #2: Struggles with meal planning on the return day (Day 3).

### Success Recipe

1. **Budget-Centric Prioritization**:
   - **Pattern**: Successful traces prioritize budget-friendly options first (transportation →
   accommodation → meals → attractions).
   - **Example**:
     - Trace #1: Starts with flights, then picks the cheapest accommodation and affordable meals.

2. **Explicit Reference Adherence**:
   - **Pattern**: Strictly using reference data for all decisions (e.g., restaurant names, flight
   numbers) ensures correctness.
   - **Example**:
     - Trace #1: Uses exact flight numbers and restaurant names from the reference.

3. **Conservative Meal Planning**:
  - **Pattern**: Skipping meals or choosing low-cost options when timing or budget is tight avoids
   overspending.
   - **Example**:
     - Trace #1: Skips lunch on Day 1 and selects affordable restaurants for other meals.

4. **Continuous Budget Updates**:
  - **Pattern**: Regularly updating the remaining budget after each decision prevents overspending.
   - **Example**:
     - Trace #1: Tracks expenses after flights, accommodation, and meals.

### Observations

1. **Incorrect Verdict in Trace #2**:
   - The reasoning in Trace #2 is marked incorrect, likely due to:
     - Overlooking flight options (assuming none are available).
    - Poor meal cost estimation (selecting high-cost restaurants like Blue Point Grill without budget
     checks).
     - Incomplete budget tracking (total expenses are underestimated).

2. **Ambiguity Handling**:
  - Successful traces handle ambiguity (e.g., missing data) by skipping non-critical items (e.g.,
   meals) or choosing defaults (e.g., cheapest options).
  - Unsuccessful traces fail to handle ambiguity gracefully (e.g., Trace #2 assumes no flights without
   proper verification).

### Examples of Patterns

1. **Transportation Selection**:
  - Trace #1: "The total flight cost is $820. Since self-driving and taxi options have no valid info,
   flights are the only viable option."
   - Trace #2: "Self-driving total is 71 + 72 = $143, which fits well within the budget."

2. **Budget Tracking**:
   - Trace #1: "Total so far: $820 + $210 = $1,030. Remaining budget: $1,700 - $1,030 = $670."
   - Trace #2: "Total so far: 143 + 148 + 132 = $423. That leaves plenty of budget room."

3. **Edge Case (Missing Data)**:
  - Trace #1: "Lunch would be in St. Pete before departure, but the user's reference info only includes
   Rockford restaurants. So maybe skip lunch on day 1."

### Deviations/Anomalies

1. **Trace #2's Incorrect Flight Assumption**:
  - The reasoning incorrectly assumes no flights are available, leading to suboptimal self-driving
   selection. This may stem from incomplete reference data or oversight.
```

```
2. **Trace #2's High-Cost Meals**:
   - The reasoning includes high-cost restaurants (e.g., Blue Point Grill at $75) without ensuring
   budget compliance, unlike Trace #1's conservative approach
```

Rule Extractor

```
Here are the extracted heuristics based on the identified patterns:

---

### **1. Constraint-First Initialization Rule**
- **Rule Description**: Always start by explicitly listing all constraints (budget, duration,
preferences) and verifying available options from reference data before making any decisions.
- **Supporting Patterns**: Initial Setup and Constraints Check (100% frequency).
- **Example Application**:
  - *Input*: Budget = $1,700, Duration = 3 days.
  - *Action*: First check transportation options (flight/taxi/self-driving), then accommodation and
  meal costs.

---

### **2. Transportation Cost-Priority Rule**
- **Rule Description**: Prioritize transportation options in this order:
  1. **Flight** (if available and total cost <= 40\% of budget).
  2. **Self-driving** (if flight unavailable or too expensive, and cost <= 25\% of budget).
  3. **Taxi** (only if no other options fit).
- **Conditions**: Skip if no reference data exists for any option.
- **Supporting Patterns**: Transportation Selection (100% frequency), Edge Case #1 (missing flights).
- **Example Application**:
  - *Budget $900*: Self-driving ($143) chosen because flights are missing/unaffordable.

---

### **3. Greedy Accommodation Selection Rule**
- **Rule Description**: Select the cheapest viable accommodation that:
  - Fits the remaining budget (after transportation).
  - Matches the required nights.
- **Conditions**: If no options exist, mark as "unavailable" and proceed.
- **Supporting Patterns**: Accommodation Selection (100% frequency).
- **Example Application**:
  - *Remaining budget $670*: Choose "$105/night Private Room" over costlier options.

---

### **4. Conservative Meal Planning Rule**
- **Rule Description**: For each meal:
  1. Skip if travel timing conflicts (e.g., flights during lunch).
  2. Select the lowest-cost restaurant that fits cuisine preferences (if specified).
  3. Cap meal cost at 15% of remaining daily budget.
- **Supporting Patterns**: Meal Planning (100% frequency), Success Recipe #3 (conservative planning).
- **Example Application**:
  - *Day 1 Lunch*: Skipped due to flight conflict.
  - *Day 2 Dinner*: Choose "$20 avg cost" restaurant over "$75" option.

---

### **5. Attraction Inclusion Rule**
- **Rule Description**: Include 1-2 free/low-cost attractions per day, prioritizing:
  - Proximity to current city.
  - Alignment with remaining budget (cost <= 10% of daily budget).
- **Supporting Patterns**: Attraction Selection (100% frequency).
- **Example Application**:
  - *Day 1*: Visit free "Burpee Museum" instead of paid options.

---

### **6. Dynamic Budget Tracking Rule**
- **Rule Description**: After each decision (transportation → accommodation → meals → attractions):
  1. Subtract cost from total budget.
 2. If remaining budget < 0, backtrack and replace the most expensive item with a cheaper alternative.
- **Supporting Patterns**: Budget Tracking (100% frequency), Edge Case #2 (high-cost items).
- **Example Application**:
  - *After flights + accommodation*: "$1,700 → $670 remaining".
  - *If meals exceed $670*: Swap "$75 restaurant" for "$20 option".

---

### **7. Structured Output Rule**
- **Rule Description**: Format output as daywise dictionaries with keys:
  `current_city`, `transportation`, `meals`, `accommodation`, `attractions`.
- **Conditions**: Omit keys with no data (e.g., missing meals).
- **Supporting Patterns**: Output Structuring (100% frequency).

---

### **Edge Case Handling Heuristics**
1. **Missing Data Fallback**:
  - If reference data is missing (e.g., no flights), default to next viable option (self-driving) and
   log a warning.
   - *Example*: Trace #2 should have flagged "no flight data" instead of assuming unavailability.

2. **High-Cost Item Mitigation**:
  - If an item (meal/accommodation) exceeds 30% of remaining budget, reject it and search for cheaper
   alternatives.
   - *Example*: Trace #2's "$75 restaurant" should trigger a re-search.

3. **Travel Conflict Resolution**:
   - If travel overlaps with meal times, skip the meal and reallocate budget elsewhere.

---

### **Efficiency Evaluation**
- **Optimization**: These rules reduce backtracking by prioritizing high-impact decisions
(transportation/accommodation) first.
- **Robustness**: Explicit budget tracking and fallbacks handle edge cases gracefully.
- **Interpretability**: Each rule is condition-based and mirrors human-like prioritization
(e.g., "cheapest first").

---

Let me know if you'd like to refine any rule further!
```

Self Corrector

```
### **Analysis of Reasoning Traces and Self-Corrections**

#### **1. Correction Detection**
- **Sample #1**: The reasoning process involves multiple adjustments to meal planning and budgeting,
particularly around flight timings and restaurant availability. Corrections occur when recalculating
meal costs and aligning flight schedules with meal times.
- **Sample #2**: The initial assumption about flight availability is corrected when the reference shows
no flights, leading to a shift to self-driving. Later, the reasoning adjusts meal selections to fit the
budget better, though the final verdict is incorrect due to incomplete adherence to reference
constraints.

#### **2. Error Categorization**
- **Logical Inconsistencies** (Sample #2): Assuming flights exist when they don't, leading to an
initial incorrect plan.
- **Incorrect Assumptions** (Both Samples): Assuming meals can be skipped or adjusted without strict
adherence to reference data (e.g., skipping lunch in Sample #1 when no St. Pete restaurants are listed).
- **Miscalculations** (Sample #1): Recalculating meal costs multiple times due to misalignment with
flight schedules.
- **Reference Misalignment** (Sample #2): Failing to strictly use reference data for all details
(e.g., restaurant names, attraction listings).

#### **3. Correction Strategies**
```

```
- **Revisiting Assumptions**: Checking flight availability first before proceeding (Sample #2).
- **Recalculating Costs**: Adjusting meal plans iteratively to fit the budget (Sample #1).
- **Realigning with Reference**: Ensuring all restaurants, accommodations, and attractions are
strictly from the reference (Sample #2 correction needed).
- **Temporal Alignment**: Matching meal times with transportation schedules (Sample #1).

#### **4. Common Errors**
1. **Assuming Unavailable Options**: Assuming flights or restaurants exist without verifying the
reference.
2. **Budget Miscalculations**: Underestimating or overestimating costs due to incomplete calculations.
3. **Ignoring Reference Constraints**: Including restaurants or accommodations not listed in the
reference.
4. **Temporal Misalignment**: Not accounting for travel times when scheduling meals or activities.

#### **5. Correction Strategies**
- **Preemptive Verification**: Always check the reference for transportation, meals, and
accommodations before planning.
- **Iterative Budgeting**: Recalculate costs after each addition to ensure budget compliance.
- **Strict Adherence**: Use only reference-listed options for all details (restaurants, attractions,
etc.).
- **Schedule Validation**: Ensure meal and activity times align with transportation schedules.

#### **6. Heuristic Refinements**
- **Transportation First**: Always verify transportation options before proceeding with the itinerary.
- **Budget Buffering**: Leave a buffer in the budget for unexpected costs or adjustments.
- **Reference Cross-Check**: Double-check every detail (restaurant names, attraction spellings)
against the reference.
- **Temporal Mapping**: Create a timeline of travel and activities to avoid scheduling conflicts.

### **Output Format**
#### **Common Errors**
1. Assuming unavailable flights or restaurants.
2. Miscalculating total costs due to incomplete iterations.
3. Including non-reference options in the plan.
4. Misaligning meal/activity times with travel.

#### **Correction Strategies**
1. Verify all options against the reference first.
2. Recalculate costs iteratively after each addition.
3. Use strict reference matching for all details.
4. Map out a timeline to align travel, meals, and activities.

#### **Heuristic Refinements**
1. **Transportation Priority**: Confirm transportation before other planning.
2. **Budget Tracking**: Maintain a running total of expenses.
3. **Reference Rigor**: Enforce strict use of reference data.
4. **Schedule Synchronization**: Use a timeline to ensure logical flow.

### **Final Notes**
- **Sample #1** succeeds due to iterative corrections and strict reference adherence.
- **Sample #2** fails because of initial incorrect assumptions and incomplete reference alignment. The
heuristic refinements would prevent such errors in future iterations.
```

D.2 TimeArena-Static

We include comprehensive meta agent responses generated by LRPLAN with GPT-4o and o3-mini.

Pattern Recognizer

```
## Analysis of Reasoning Traces

### Segment Analysis
1. **Task Breakdown**: Each reasoning trace begins by listing the tasks and their dependencies,
identifying which actions are Type 1 (occupy until completion) and Type 2 (allowing other
actions).
2. **Dependency Mapping**: The traces identify dependencies between tasks, such as needing to wash a
dish before adding ingredients or picking ingredients before cooking.
3. **Action Sequencing**: The traces attempt to sequence actions to minimize total time, often by
```

```
interleaving Type 2 actions with Type 1 actions.
4. **Time Management**: Each trace involves calculating the time steps required for each action and
ensuring actions are performed in the correct order.

### Pattern Detection
1. **Initial Setup**: Each trace starts with washing the dish, a common initial step due to its
dependency requirement.
2. **Parallel Processing**: Attempts to run Type 2 actions in parallel with Type 1 actions to optimize
time.
3. **Sequential Dependencies**: Ensuring that actions dependent on previous steps (e.g., adding
ingredients to a dish) are correctly sequenced.
4. **Time Step Calculation**: Consistent use of time steps to manage and track the duration of actions.

### Comparison
- **Similarities**: All traces follow a similar structure of listing tasks, identifying dependencies,
and attempting to interleave actions to optimize time.
- **Differences**: Variations occur in how effectively actions are interleaved, with some traces
struggling to manage overlapping actions due to fryer or pot usage.

### Abstraction
1. **Heuristic for Efficiency**: Start with actions that have dependencies (e.g., washing dishes) and
interleave Type 2 actions with Type 1 actions to maximize efficiency.
2. **Dependency First**: Always address dependencies first to avoid blocking subsequent actions.
3. **Time Step Management**: Use time steps to track action durations and ensure no overlaps in
resource usage (e.g., fryer).

### Edge Case Identification
1. **Resource Conflicts**: Struggles occur when multiple actions require the same resource (e.g.,
fryer) simultaneously.
2. **Action Overlaps**: Difficulty arises in managing overlapping Type 1 actions due to their blocking
nature.
3. **Complex Dependencies**: Scenarios with multiple dependencies and interleaving requirements can
lead to inefficient sequencing.

### Success and Failure Analysis
- **Success Recipe**: Efficient traces correctly interleave Type 2 actions with Type 1 actions, manage
dependencies upfront, and avoid resource conflicts.
- **Failure Points**: Inefficient traces often mismanage time steps, leading to resource conflicts or
incorrect sequencing of dependent actions.

### Examples
- **Efficient Interleaving**: In Sample Reasoning Trace #2, the agent successfully interleaves cooking
and preparation tasks, minimizing idle time.
- **Resource Conflict**: In Sample Reasoning Trace #4, the agent struggles with overlapping fryer
usage, leading to inefficiencies.

### Observations
- **Common Errors**: Mismanagement of time steps and resource conflicts are frequent issues.
- **Optimization Challenges**: Balancing parallel processing with sequential dependencies is a
recurring challenge.
- **Complexity Management**: As tasks increase in complexity, maintaining an efficient sequence
becomes more difficult.

Overall, the reasoning traces reveal a consistent approach to task management, with common challenges
in resource allocation and dependency management. The key to success lies in effectively
interleaving actions and managing time steps to avoid conflicts.
```

Rule Extractor

```
### Heuristic 1: Dependency-First Approach
- **Rule Description**: Always address tasks with dependencies first to prevent blocking subsequent
actions.
- **Supporting Patterns**: Dependency Mapping, Initial Setup, Dependency First.
- **Example Application**: Before starting to cook, ensure all necessary ingredients are prepared and
the dish is washed, as these are prerequisites for the cooking process.

### Heuristic 2: Interleave Type 2 Actions with Type 1 Actions
- **Rule Description**: To optimize time, interleave Type 2 actions (non-blocking) with Type 1 actions
```

```
(blocking) whenever possible.
- **Supporting Patterns**: Action Sequencing, Parallel Processing, Efficient Interleaving.
- **Example Application**: While a dish is baking (Type 1), prepare the salad (Type 2) to utilize the waiting time effectively.

### Heuristic 3: Time Step Management
- **Rule Description**: Use time steps to track action durations and ensure no overlaps in resource usage.
- **Supporting Patterns**: Time Management, Time Step Calculation, Success Recipe.
- **Example Application**: Allocate specific time slots for each task and monitor them to prevent two tasks from requiring the same resource simultaneously.

### Heuristic 4: Resource Conflict Avoidance
- **Rule Description**: Plan tasks to avoid simultaneous use of the same resource, especially for Type 1 actions.
- **Supporting Patterns**: Resource Conflicts, Action Overlaps, Failure Points.
- **Example Application**: If two dishes require the fryer, schedule them sequentially or use an alternative cooking method for one.

### Heuristic 5: Complexity Management
- **Rule Description**: Break down complex tasks into simpler, manageable steps and address each dependency systematically.
- **Supporting Patterns**: Task Breakdown, Complex Dependencies, Optimization Challenges.
- **Example Application**: For a multi-course meal, list all tasks, identify dependencies, and sequence them to ensure each course is ready at the right time.

### Efficiency Evaluation
- **Dependency-First Approach**: Optimizes workflow by preventing bottlenecks.
- **Interleave Actions**: Reduces idle time, increasing overall efficiency.
- **Time Step Management**: Ensures smooth operation without resource conflicts.
- **Resource Conflict Avoidance**: Prevents delays caused by resource contention.
- **Complexity Management**: Simplifies task execution, reducing errors and inefficiencies.

These heuristics provide a structured approach to managing tasks with dependencies and resource constraints, enhancing decision-making efficiency in complex scenarios.
```

Self Corrector

```
## **Analysis of Reasoning Traces**

### **Correction Detection**
1. **Sample Reasoning Trace #1**: The model initially miscalculates the timing of actions, particularly the overlap of Type 1 and Type 2 actions. It corrects itself by realizing that Type 2 actions allow for parallel processing.
2. **Sample Reasoning Trace #2**: The model struggles with the timing of Type 1 actions, initially thinking they can overlap with Type 2 actions. It corrects by ensuring Type 1 actions are completed before starting another.
3. **Sample Reasoning Trace #3**: The model initially mismanages the sequence of actions, particularly the overlap of frying and adding to the dish. It corrects by adjusting the sequence to avoid conflicts.
4. **Sample Reasoning Trace #4**: The model attempts to interleave tasks but struggles with the complexity of managing two dishes. It corrects by trying to sequence tasks more linearly.

### **Error Categorization**
- **Logical Inconsistencies**: Misunderstanding the nature of Type 1 and Type 2 actions, leading to incorrect assumptions about parallel processing.
- **Incorrect Assumptions**: Assuming actions can overlap when they cannot, particularly with Type 1 actions.
- **Miscalculations**: Errors in calculating the end times of actions, especially when transitioning between tasks.

### **Correction Strategies**
- **Revisiting Assumptions**: The model frequently revisits its assumptions about action types and dependencies to correct errors.
- **Adjusting Calculations**: It recalculates the timing of actions to ensure they fit within the constraints.
- **Re-evaluating Premises**: The model re-evaluates the sequence of actions to find a more efficient order.
```

```
### **Cumulative Learning**
- **Common Errors**: Misunderstanding of Type 1 and Type 2 actions, incorrect sequencing of tasks, and
miscalculations of action durations.
- **Correction Strategies**: Reassessing action dependencies, recalculating action timings, and
reordering tasks to avoid conflicts.

### **Refinement Mechanism**
- **Heuristic Refinements**:
  - **Action Type Clarification**: Clearly define and differentiate between Type 1 and Type 2 actions
    to prevent overlap errors.
  - **Dependency Mapping**: Create a visual map of dependencies to better understand the sequence of
    actions.
  - **Time Management**: Implement a time-tracking mechanism to accurately calculate the start and end
    times of actions.
  - **Parallel Processing Awareness**: Develop a heuristic to identify when parallel processing is
    possible and when it is not.

### **Output Format**
- **Common Errors**:
  - Misunderstanding of action types.
  - Incorrect sequencing of tasks.
  - Miscalculations of action durations.

- **Correction Strategies**:
  - Reassessing action dependencies.
  - Recalculating action timings.
  - Reordering tasks to avoid conflicts.

- **Heuristic Refinements**:
  - Clarify action types.
  - Map dependencies visually.
  - Implement time-tracking.
  - Develop parallel processing awareness.

By integrating these insights, future reasoning processes can be refined to minimize errors and
improve efficiency in task completion.
```

E Evaluation Metrics for TimeArena-Static

- **Full Completion Rate (FCR)** The rate of completely correct samples:

$$\text{FCR} = \frac{\sum_{s \in S} \text{complete}(s, p_s)}{|S|} \quad (1)$$

  where S is the set of all samples, each sample being a set of tasks and complete(.) returns 1 when all tasks in the sample s are completed by the given plan p_s corresponding to the sample.

- **Task Completion Rate (TCR)** The rate of completely correct tasks:

$$\text{TCR} = \frac{\sum_{s \in S} \sum_{t \in s} \text{complete}(\{t\}, p_s)}{\sum_{s \in S} |s|} \quad (2)$$

- **Average Progress (AP)** The average of progress percentage achieved per task:

$$\text{AP} = \frac{\sum_{s \in S} \sum_{t \in s} \text{progress}(t, p_s)}{\sum_{s \in S} |s|} \quad (3)$$

  where progress returns a progress score out of 100 for the percentage of task completed successfully weighted in terms of duration of successfully completed subtasks

- **Average Completion Time (ACT)** The average time taken for completing tasks successfully:

$$\text{ACT} = \frac{\sum_{i \in C_t} T_i}{|C_t|} \quad (4)$$

  where C_t is the set of completed tasks and T_i is the time at which task i was completed.

- **Progress Speed (PS)** Sum of highest progress scores achieved by each task divided by the sum of time taken to achieve them.

$$\text{PS} = \frac{\sum_{s \in S} \sum_{t \in s} \text{progress}(t, p_s)}{\sum_{s \in S} \sum_{t \in s} T_t} \quad (5)$$

- **Idle Violation Rate** Total number of times an action was initiated but was invalid, because another non-idle task was underway, across all plans.

- **Dependency Violation Rate** Total number of times an action was initiated but was invalid, because its prerequisite sub-task(s) was(were) not completed, across all plans.

F Error Analysis

We conducted a detailed error analysis of LRPLAN across both the TravelPlanner and TimeArena-Static domains (see Tables 8 and 9. Also, Table 10 shows evaluation across difficulty levels for TimeArena.

F.1 TravelPlanner

In the TravelPlanner domain, the most frequent error involved violations of the “minimum nights stay” constraint, where the generated plans did not always ensure that each accommodation met the required minimum stay. Additionally, LRPLAN occasionally failed to enforce commonsense constraints regarding transportation modes, sometimes including both self-driving and flight segments within a single itinerary which is an impractical scenario since a self-driven vehicle cannot be abandoned mid-trip. The system also exhibited difficulties in consistently honoring traveler cuisine preferences at restaurants, particularly when balancing these preferences with budget constraints.

Both variants of LRPLAN (GPT-4o + o3-mini and DeepSeek-V3 + DeepSeek-R1) demonstrate balanced performance across commonsense and hard constraints. This suggests that the multi-agent collaboration and LLM-LRM synergy effectively generalize across constraint types, unlike other strategies that tend to skew toward one. Strategies like Direct Prompting and Chain of Thought with GPT-4o achieve high scores on commonsense constraints (e.g., CI, DA, DR), but their hard constraint performance is weak, especially on Budget (B), Cuisine (C), and Room Type (RT). This indicates that while these models can reason well with general knowledge, they lack precision in satisfying rigid, domain-specific rules.

AutoBuild with o3-mini and DeepSeek-R1 shows relatively better performance on hard constraints, particularly Budget and Room Rule. However, it lags in commonsense reasoning, suggesting that its structured planning may come at the cost of flexibility or contextual understanding.

Despite high scores on some commonsense constraints (e.g., DA, RCR, WCC), Multi-Agent Debate strategies show very poor hard constraint adherence and low pass rates. This suggests that while debate mechanisms may surface diverse ideas, they struggle to converge on coherent, constraint-satisfying plans.

The Pass Rate metric, which reflects overall plan

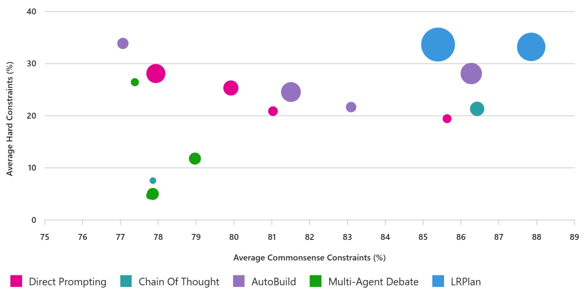


Figure 3: TravelPlanner: Trade-off between commonsense and hard constraint performance, with bubble size representing the final pass rate

validity, is highest for LRPLAN, reinforcing that its architecture not only performs well on individual constraints but also produces consistently valid plans. Notably, even when individual constraint scores are close, LRPLAN's integration and refinement steps give it a clear edge in final outcomes.

F.2 TimeArena-Static

For the TimeArena-Static domain, our analysis revealed that while LRPLAN achieves strong results in the Cooking subdomain and performs reasonably well in Laboratory tasks, its efficiency in the Household subdomain remains comparable to baseline methods and is suboptimal. Also, there is room for improvement in reducing the average time required for successful task completion.

Cooking domain is the hardest for baselines. Most methods, especially GPT-4o and DeepSeek-V3, show very low performance in Cooking, suggesting that this domain requires more nuanced reasoning and constraint satisfaction. LRPLAN's strong performance here suggests its meta-agent-driven refinement is particularly effective in recovering from initial plan errors. Despite being a widely used prompting strategy, Chain of Thought performs poorly, especially in FCR (e.g., <14% across domains). This highlights that reasoning traces alone are insufficient without structured correction mechanisms like those in LRPLAN.

Table 10 shows evaluation across difficulty levels for TimeArena. LRPLAN maintains high TCR and FCR across Easy, Medium, and Hard tasks, unlike other strategies that show steep performance drops as difficulty increases.

G Analyzing AutoBuild

AutoBuild is a multi-agent system where agents are peers. Let us consider two of the agents as first agent and final agent. First agent is the agent which is the first agent which generates a plan draft. Final agent is the one that outputs the final plan. Tables 12 and 13 show performance of the first agent and final agent output using AutoBuild with various LLMs and LRMs for TravelPlanner and TimeArena-Static respectively.

Table 12 shows that for GPT-4o, the final agent does not improve over the first. Commonsense slightly drops, and hard constraint scores remain flat or marginally improve. This suggests limited benefit from iterative refinement in this setup. DeepSeek-R1, however, shows a notable jump in final agent performance, especially in macro-level constraint satisfaction and final pass rate. This implies that iterative reasoning or planning helps DeepSeek-R1 course-correct and generalize better.

Table 13 also shows that GPT-4o's final agent shows a regression in both completion rates (Task CR drops by ∼5%, Full CR by ∼8%) and average progress (from 73.59 to 68.07), despite a slight improvement in completion time (CT). This suggests that the final agent may be over-optimizing for speed, sacrificing task completeness and constraint adherence. DeepSeek-V3 shows virtually no change across all metrics, suggesting that the final agent adds little value, possibly due to limited internal state evolution or lack of effective feedback incorporation. DeepSeek-R1 shows mixed signals: while completion rates improve slightly, violations increase, and progress metrics decline. This may indicate instability in refinement, where the final agent introduces new errors while trying to optimize.

H Analyzing Multi-Agent Debate

Multi-Agent Debate is also a multi-agent system where agents are peers. Tables 14 and 15 show performance of the first agent and final agent output using Multi-Agent Debate with various LLMs and LRMs for TravelPlanner and TimeArena-Static respectively.

Unlike Autobuild, Table 14 shows that Multi-Agent Debate leads to significant improvements across metrics from first to final agent across models. GPT-4o shows modest improvements across all metrics, with steady gains in commonsense reasoning, constraint satisfaction, and final task success. o3-mini exhibits the most dramatic gains, especially in satisfying hard constraints, while also improving in commonsense reasoning and overall success. DeepSeek-V3 demonstrates balanced progress, with strong improvements in both com-

Strategy	LLM	Commonsense Constraints ↑							Hard Constraints ↑					Final
		CI	DA	DR	MNS	NCT	RCR	WCC	B	C	RR	RT	T	Pass Rate
Direct Prompting	GPT-4o	88.33	100.00	95.00	60.00	73.33	84.44	98.33	30.00	9.44	10.00	26.11	21.67	3.89
Direct Prompting	o3-mini	98.89	99.44	48.89	49.44	73.33	94.44	95.00	41.67	16.67	27.22	21.67	19.44	8.33
Chain of thought	GPT-4o	98.33	100.00	96.67	57.22	73.33	81.11	98.33	33.33	17.22	10.56	25.00	20.56	7.78
AutoBuild	GPT-4o	93.89	97.78	91.67	56.67	72.22	93.89	97.78	46.67	18.33	19.44	30.00	26.11	12.78
AutoBuild	o3-mini	96.67	96.67	36.67	41.11	72.22	96.67	99.44	56.67	21.11	34.44	31.67	25.56	5.56
Multi-Agent Debate	GPT-4o	16.11	99.44	90.00	66.11	73.33	99.44	100.00	7.78	3.89	4.44	3.33	3.89	2.78
Multi-Agent Debate	o3-mini	87.22	96.11	16.67	74.44	72.22	96.67	98.33	45.56	14.44	25.56	26.67	20.00	3.33
ExpeL	GPT-4o	82.50	84.17	82.50	60.83	70.00	80.00	84.17	35.83	45.45	36.73	68.18	70.59	10.00
ExpeL	o3-mini	56.67	58.33	50.00	28.33	52.50	55.83	57.50	38.33	45.45	59.18	68.18	61.76	8.89
LRPLAN (ours)	GPT-4o + o3-mini	98.33	99.44	86.67	66.11	74.44	90.56	99.44	50.56	22.78	33.89	31.67	27.22	17.78
Direct Prompting	DeepSeek-V3	97.78	86.11	80.00	35.56	72.22	97.22	98.33	32.22	12.22	18.33	21.11	20.56	4.44
Direct Prompting	DeepSeek-R1	94.44	95.00	57.22	38.89	70.00	95.00	95.00	53.33	15.56	25.00	24.44	22.22	11.11
Chain of thought	DeepSeek-V3	97.78	100.00	81.67	47.22	71.11	99.44	47.78	14.44	6.67	2.22	8.89	5.56	2.22
AutoBuild	DeepSeek-V3	97.78	92.78	76.67	38.33	81.67	95.00	99.44	38.33	11.67	16.67	21.11	20.56	5.00
AutoBuild	DeepSeek-R1	85.00	98.33	61.11	54.44	76.11	97.22	98.33	55.56	15.56	20.00	17.78	13.89	11.67
Multi-Agent Debate	DeepSeek-V3	48.89	100.00	90.56	43.89	71.11	98.33	100.00	18.89	11.67	8.89	8.89	10.56	6.11
Multi-Agent Debate	DeepSeek-R1	27.22	97.78	83.33	72.22	72.22	94.44	97.78	11.67	2.22	5.00	2.78	3.33	6.11
ExpeL	DeepSeek-V3	88.33	88.33	83.33	46.67	86.67	88.33	88.33	35.00	35.29	62.50	86.36	94.12	4.44
LRPLAN (ours)	DeepSeek-V3 + DeepSeek-R1	97.22	99.44	83.33	45.00	73.89	99.44	99.44	74.44	20.56	24.44	25.00	23.89	21.67

Table 8: Evaluation of different strategies across constraint categories. **Abbreviations:** CI: Complete Information, DA: Diverse Attractions, DR: Diverse Restaurants, MNS: Minimum Nights Stay, NCT: Non-conf. Transportation, RCR: Reasonable City Route, WCC: Within Current City, B: Budget, C: Cuisine, RR: Room Rule, RT: Room Type, T: Transportation. All constraint values are reported as pass rates (%).

Strategy	LLM	Household		Cooking		Laboratory		Total	
		TCR ↑	FCR ↑	TCR ↑	FCR ↑	TCR ↑	FCR ↑	TCR ↑	FCR ↑
Direct Prompting	GPT-4o	47.62	28.57	4.76	4.76	59.52	38.10	37.30	23.81
Direct Prompting	o3-mini	64.29	38.10	30.95	42.86	90.48	85.71	61.90	55.56
Chain of thought	GPT-4o	21.43	9.52	2.38	0.00	35.71	19.05	19.84	9.52
AutoBuild	GPT-4o	25.64	19.05	0.00	0.00	69.05	47.62	31.71	22.22
AutoBuild	o3-mini	69.05	47.62	12.2	14.29	95.24	95.24	59.20	52.38
Multi-Agent Debate	GPT-4o	47.62	23.81	4.76	0.00	57.14	38.10	36.51	20.63
Multi-Agent Debate	o3-mini	80.95	61.90	28.57	33.33	95.24	90.48	68.25	61.90
ExpeL	GPT-4o	40.48	23.81	0.00	0.00	38.10	19.05	26.19	14.29
ExpeL	o3-mini	78.57	57.14	66.67	66.67	97.62	95.24	80.95	73.01
LRPLAN (ours)	GPT-4o + o3-mini	71.43	52.38	88.10	85.71	90.48	85.71	83.33	74.60
Direct Prompting	DeepSeek-V3	11.90	19.05	7.14	9.52	28.57	14.29	15.87	14.29
Direct Prompting	DeepSeek-R1	40.74	9.52	38.71	28.57	50.00	38.10	43.75	25.40
Chain of thought	DeepSeek-V3	28.57	14.29	21.43	14.29	45.24	23.81	31.75	17.46
AutoBuild	DeepSeek-V3	21.43	23.81	16.67	14.29	50.00	38.10	29.37	25.40
AutoBuild	DeepSeek-R1	33.33	9.52	33.33	23.81	78.57	71.43	48.41	34.92
Multi-Agent Debate	DeepSeek-V3	11.90	14.29	11.90	14.29	21.43	19.05	15.08	15.87
Multi-Agent Debate	DeepSeek-R1	25.64	9.52	38.24	23.81	58.97	42.86	41.07	25.40
ExpeL	DeepSeek-V3	59.52	42.86	28.57	19.05	61.90	33.33	50.00	31.75
LRPLAN (ours)	DeepSeek-V3 + DeepSeek-R1	76.19	57.14	54.76	52.38	73.81	52.38	68.25	53.97

Table 9: Evaluation of different task types in TimeArena-Static **Abbreviations:** FCR: Full Completion Rate, TCR: Task Completion Rate. All values are reported as pass rates (%).

monsense and constraint metrics, leading to a notable increase in final task success. DeepSeek-R1 also shows solid gains, particularly in final success rate and constraint handling, with consistent improvements in commonsense reasoning. Overall, all models benefit from the multi-agent debate process, with o3-mini and DeepSeek-V3 standing out for their substantial advancements.

Table 15 shows that GPT-4o has an increase in idle and departure violations, with slight improvements in completion time and progress speed, but a decrease in average progress and completion rates. o3-mini exhibits a reduction in both idle and departure violations, with slight improvements in completion time and progress speed, and notable increases in average progress and completion rates. DeepSeek-V3 shows a slight increase in idle violations and a decrease in departure violations, with significant improvements in completion time and progress speed, but a drop in average progress and task completion rate. DeepSeek-R1 experiences increases in both idle and departure violations, with minor gains in completion time and progress speed, but declines in average progress and completion rates. Overall, o3-mini demonstrates the most consistent and substantial improvements, while the other models show mixed outcomes with both gains and regressions.

I TimeArena Case Study

Fig. 4 shows an example from TimeArena-Static dataset. Baselines like AutoBuild and Multi-Agent

Strategy	LLM	Easy		Medium		Hard		Total	
		TCR ↑	FCR ↑	TCR ↑	FCR ↑	TCR ↑	FCR ↑	TCR ↑	FCR ↑
Direct Prompting	GPT-4o	42.86	42.86	40.48	23.81	33.33	04.76	37.30	23.81
Chain of thought	GPT-4o	19.05	19.05	26.19	09.52	15.87	00.00	19.84	09.52
AutoBuild	GPT-4o	33.33	33.33	35.71	19.05	28.33	14.28	31.71	22.22
Multi-Agent Debate	GPT-4o	33.33	33.33	38.10	19.05	36.51	09.52	36.51	20.63
ExpeL	GPT-4o	28.57	28.57	28.57	14.29	23.81	0.00	26.19	14.29
Direct Prompting	o3-mini	95.24	95.24	59.52	47.62	52.38	23.81	61.90	55.56
AutoBuild	o3-mini	75.00	71.43	61.90	52.38	52.38	33.33	59.20	52.38
Multi-Agent Debate	o3-mini	85.71	85.71	64.28	57.14	65.08	42.86	68.25	61.90
ExpeL	o3-mini	95.24	95.24	76.19	66.67	79.37	57.14	80.95	73.01
LRPLAN (ours)	GPT-4o + o3-mini	90.48	90.48	83.33	71.43	80.95	61.90	83.33	74.60
Direct Prompting	DeepSeek-V3	42.86	42.86	14.28	00.00	07.94	00.00	15.87	14.29
Chain of thought	DeepSeek-V3	23.81	23.81	42.86	19.05	26.98	09.52	31.75	17.46
AutoBuild	DeepSeek-V3	42.86	42.86	03.33	19.05	22.22	14.28	29.37	25.40
Multi-Agent Debate	DeepSeek-V3	42.86	42.86	14.28	04.76	06.35	00.00	15.08	15.87
ExpeL	DeepSeek-V3	42.86	42.86	61.90	38.10	44.44	14.29	50.00	31.75
Direct Prompting	DeepSeek-R1	57.89	52.38	38.46	09.52	41.18	14.28	43.75	25.40
AutoBuild	DeepSeek-R1	52.38	52.38	50.00	28.57	46.03	23.81	48.41	34.92
Multi-Agent Debate	DeepSeek-R1	47.62	47.62	50.00	23.81	25.40	04.76	41.07	25.40
LRPLAN (ours)	DeepSeek-V3 + R1	80.95	80.95	71.43	52.38	61.90	28.57	68.25	53.97

Table 10: Evaluation of TimeArena-Static across difficulty levels. TCR = Task Completion Rate, FCR = Full Completion Rate (all in %).

Strategy	LLM	Easy Micro		Medium Micro		Hard Micro		Full Micro
		Commonsense ↑	Hard ↑	Commonsense ↑	Hard ↑	Commonsense ↑	Hard ↑	Pass Rate ↑
Direct Prompting	GPT-4o	84.38	45.00	82.08	35.83	88.33	43.75	3.89
Chain of thought	GPT-4o	84.58	46.67	83.33	45.00	87.92	45.83	7.78
AutoBuild	GPT-4o	84.58	60.00	85.42	54.17	90.42	63.33	12.78
Multi-Agent Debate	GPT-4o	78.75	16.67	76.88	6.25	82.71	11.25	2.78
ExpeL	GPT-4o	78.75	37.50	80.00	52.50	73.44	45.62	10.00
Direct Prompting	o3-mini	80.21	68.33	76.46	49.17	79.79	53.33	8.33
AutoBuild	o3-mini	78.12	83.33	76.88	63.33	81.88	74.58	5.56
Multi-Agent Debate	o3-mini	76.88	58.33	73.33	45.00	80.83	62.08	3.33
ExpeL	o3-mini	37.50	42.50	55.62	50.00	62.50	52.50	8.88
LRPLAN (ours)	GPT-4o + o3-mini	89.38	60.00	84.79	64.17	91.67	77.50	17.78
Direct Prompting	DeepSeek-V3	80.42	36.67	80.83	44.17	81.25	47.08	4.44
Chain of thought	DeepSeek-V3	70.00	23.33	73.96	20.83	70.83	12.08	2.22
AutoBuild	DeepSeek-V3	81.88	48.33	83.33	43.33	83.96	47.50	5.00
Multi-Agent Debate	DeepSeek-V3	73.96	20.00	77.29	24.17	79.79	27.08	6.11
ExpeL	DeepSeek-V3	80.00	60.00	81.88	47.50	83.75	57.50	4.44
Direct Prompting	DeepSeek-R1	78.12	68.33	73.75	53.33	80.21	61.67	11.11
AutoBuild	DeepSeek-R1	82.92	78.33	80.62	63.33	80.42	40.83	11.67
Multi-Agent Debate	DeepSeek-R1	76.88	21.67	73.54	10.83	77.50	11.88	6.11
LRPLAN (ours)	DeepSeek-V3 + R1	84.79	88.33	83.75	69.17	89.17	69.58	21.67

Table 11: Evaluation of TravelPlanner across difficulty levels (all in %).

	Commonsense ↑		Hard Constraints ↑		Final
	Micro	Macro	Micro	Macro	Pass Rate ↑
			GPT-4o		
first agent	88.13	37.78	59.52	34.44	12.78
final agent	86.81	35.56	60.24	36.67	12.78
			o3-mini		
first agent	79.86	15.56	78.33	60.00	7.22
final agent	78.96	13.33	72.62	54.44	5.56
			DeepSeek-V3		
first agent	83.61	21.11	45.00	22.78	4.44
final agent	83.06	21.11	46.43	26.11	5.00
			DeepSeek-R1		
first agent	81.32	13.33	53.81	47.78	7.78
final agent	81.32	18.89	52.62	50.56	11.67

Table 12: AutoBuild: First Agent Vs Final for TravelPlanner

	Violations ↓		Progress			Completion Rate ↑	
	#Idle Viol.	#Dep. Viol.	CT ↓	Prog. speed ↑	Avg Prog. ↑	Task CR	Full CR
				GPT-4o			
first agent	53	65	28.42	2.75	73.59	36.59	30.16
final agent	70	71	27.62	2.80	68.07	31.71	22.22
				o3-mini			
first agent	3	52	26.43	2.87	86.70	56.45	46.03
final agent	0	49	27.96	2.62	88.75	59.20	52.38
				DeepSeek-V3			
first agent	108	78	29.59	2.65	60.95	29.84	25.40
final agent	109	74	29.76	2.62	61.12	29.37	25.40
				DeepSeek-R1			
first agent	36	34	26.54	3.32	82.01	47.90	30.16
final agent	51	51	27.66	3.15	79.39	48.41	34.92

Table 13: AutoBuild: First Agent Vs Final for TimeArena-Static

Debate fail to satisfy implicit constraints like "subtask dependencies" and "non-idle state". Our proposed method, LRPLAN, generates an accurate plan satisfying all constraints with the help of synergistic collaboration between planner LLM and refiner LRM.

Task: Prepare and bake a cheese and tomato pizza.
Valid actions with time required for completion: -wash dish: 1 min -pick dough: 1 min -chop dough: 3 min -add dough to dish: 2 min -pick cheese: 2 min -chop cheese: 1 min -add cheese to dish: 4 min
-pick tomato: 3 min -chop tomato: 2 min -add tomato to dish: 1 min -bake dish in oven: 10 min

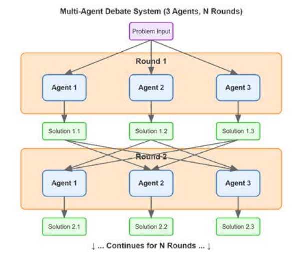
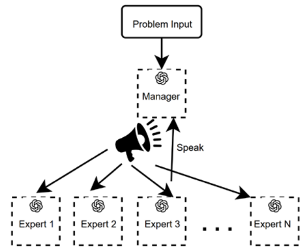
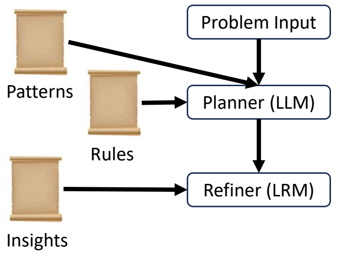
 Multi-Agent Debate System (3 Agents, N Rounds) Problem Input Round 1 Agent 1 Agent 2 Agent 3 Solution 1.1 Solution 1.2 Solution 1.3 Round 2 Agent 1 Agent 2 Agent 3 Solution 2.1 Solution 2.2 Solution 2.3 ↓ ... Continues for N Rounds ... ↓	 Problem Input Manager Speak Expert 1 Expert 2 Expert 3 . . . Expert N	 Problem Input Patterns Planner (LLM) Rules Refiner (LRM) Insights
Multi-Agent Debate	**AutoBuild**	**LRPlan (Ours)**
0: wash dish 1: pick dough 2: chop dough 3: wait 4: wait 5: add dough to dish 6: wait 7: pick cheese 8: wait 9: chop cheese 10: add cheese to dish 11: wait 12: wait 13: wait 14: pick tomato 15: wait 16: chop tomato 17: wait 18: add tomato to dish 19: bake dish in oven 20: wait 21: wait 22: wait 23: wait 24: wait 25: wait 26: wait 27: wait 28: wait 29: wait	0: wash dish 1: pick dough 2: chop dough 3: wait 4: wait 5: add dough to dish 6: pick cheese 7: wait 8: chop cheese 9: add cheese to dish 10: pick tomato 11: wait 12: wait 13: chop tomato 14: add tomato to dish 15: bake dish in oven 16: wait 17: wait 18: wait 19: wait 20: wait 21: wait 22: wait 23: wait 24: wait 25: wait	0: wash dish 1: pick dough 2: chop dough 3: wait 4: wait 5: add dough to dish 6: wait 7: pick cheese 8: wait 9: chop cheese 10: add cheese to dish 11: wait 12: wait 13: wait 14: pick tomato 15: wait 16: wait 17: chop tomato 18: wait 19: add tomato to dish 20: bake dish in oven 21: wait 22: wait 23: wait 24: wait 25: wait 26: wait 27: wait 28: wait 29: wait
Action 'chop tomato' initiated at time 16 while a non-idle action is underway ✗	Non-idle action 'add cheese to dish' started at time 9 before dependency 'chop cheese' completed. ✗	✓

Figure 4: Example from TimeArena-Static dataset: Baselines like AutoBuild and Multi-Agent Debate fail to satisfy implicit constraints like "sub-task dependencies" and "non-idle state". Our proposed method, LRPLAN, generates an accurate plan satisfying all constraints with the help of synergistic collaboration between planner LLM and refiner LRM.

	Commonsense ↑		Hard Constraints ↑		Final
	Micro	Macro	Micro	Macro	Pass Rate ↑
			GPT-4o		
first agent	75.42	3.89	4.29	2.22	1.11
final agent	79.44	7.78	10.00	6.67	2.78
			o3-mini		
first agent	69.86	1.11	0.48	1.11	1.11
final agent	77.01	7.78	56.67	40.56	3.33
			DeepSeek-V3		
first agent	70.98	3.33	7.62	4.44	1.11
final agent	77.01	14.44	25.24	15.00	6.11
			DeepSeek-R1		
first agent	69.44	2.77	2.86	2.22	1.67
final agent	75.97	8.33	10.71	11.11	6.11

Table 14: Multi-Agent Debate: First Agent Vs Final for TravelPlanner

	Violations ↓		Progress			Completion Rate ↑	
	#Idle Viol.	#Dep. Viol.	CT ↓	Prog. speed ↑	Avg Prog. ↑	Task CR	Full CR
				GPT-4o			
first agent	30	70	28.35	2.52	74.25	40.48	31.75
final agent	46	73	27.74	2.71	71.23	36.51	20.63
				o3-mini			
first agent	3	47	28.14	2.83	88.46	63.49	57.14
final agent	0	41	28.30	2.90	92.53	68.25	61.90
				DeepSeek-V3			
first agent	105	94	26.41	2.34	49.69	17.46	14.29
final agent	107	92	23.63	2.77	47.94	15.08	15.87
				DeepSeek-R1			
first agent	33	37	26.30	3.43	81.54	50.96	36.51
final agent	37	55	26.24	3.32	75.77	41.07	25.40

Table 15: Multi-Agent Debate: First Agent Vs Final for TimeArena-Static