HYPERPLAN: MODELING HIGH-ORDER TASK RELATIONS WITH CONSTRAINT HYPERGRAPHS FOR LLM PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have demonstrated strong performance on various reasoning tasks but still struggle with complex planning problems, such as travel planning, which require long-horizon reasoning under multiple hard constraints. Existing planning methods primarily focus on effectively decomposing a complex planning objective into simpler subtasks and solving them step by step. However, they often overlook high-order relations induced by shared constraints across multiple subtasks, which constitute a fundamental challenge that distinguishes complex planning from traditional reasoning tasks. To bridge this gap, we propose **HyperPlan**, a novel LLM planning framework that explicitly models high-order relations among tasks using a constraint hypergraph. First, HyperPlan induces a query-specific constraint hypergraph from the parsed query semantics. Decision-oriented task nodes represent planning subtasks, while constraint hyperedges with normalized types capture complete constraint scopes through hierarchical scope binding. Second, HyperPlan serializes the hypergraph in a constraint-centric form and performs constraint-aligned task reasoning under intra-node and inter-node consistency guidance. The coordinated task solutions are then composed into the final plan. We further theoretically characterize when explicit constraint scopes reduce LLM planning uncertainty. Experiments demonstrate that HyperPlan achieves state-of-the-art performance among planning methods with LLMs as reasoners, improving success rates by up to 28.9% over strong planning baselines across diverse planning settings.

1 INTRODUCTION

Large language models (LLMs) have demonstrated strong performance on a wide range of reasoning tasks, including mathematical reasoning, question answering, and tool-assisted problem solving Wei et al. (2022); Yao et al. (2023b); Gui et al. (2025). However, they still struggle with complex planning tasks, such as itinerary scheduling and project planning. Unlike traditional reasoning tasks primarily seeking a correct answer, planning requires the model to yield a sequence of coordinated decisions. This often involves decomposing a long-horizon planning objective into heterogeneous subtasks coupled by multiple constraints. For example, a travel plan can be decomposed into transportation, accommodation, meal, and attraction planning, whose decisions must remain consistent in dates, locations, and budget, posing substantial difficulty for LLMs in generating a feasible plan.

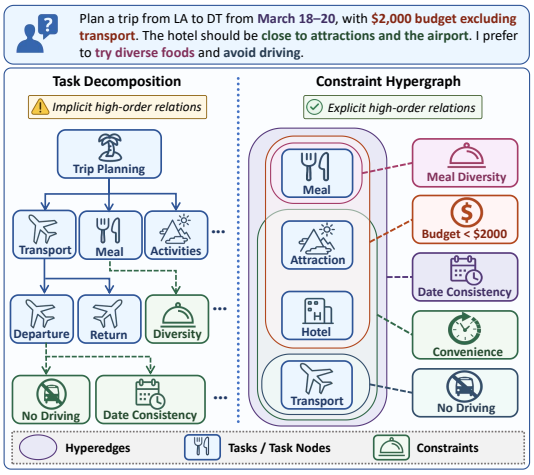


Figure 1: Task decomposition leaves shared high-order relations implicit, whereas constraint hypergraph modeling captures them explicitly.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing studies mainly focus on decomposing a complex objective into simpler subtasks to improve the planning capabilities of LLMs. Structured reasoning methods organize intermediate reasoning through chains Wei et al. (2022), trees Yao et al. (2023a), or graphs Besta et al. (2024), while tool-assisted approaches exploit knowledge retrieval Yao et al. (2023b), environmental feedback Zhou et al. (2024), and symbolic planners Liu et al. (2023). Planning-oriented approaches further employ explicit task decomposition Wang et al. (2023) and hierarchical planning outlines Gui et al. (2025). Collectively, these studies demonstrate the importance of task decomposition in reducing the reasoning burden of LLMs and improving their ability to solve long-horizon planning problems.

Despite the success of decomposition-based methods, they often overlook high-order task relations induced by shared constraints in complex planning. Consequently, task solutions may fail to jointly satisfy these constraints, compromising global plan feasibility. For example, as illustrated in Figure 1, task decomposition can incorporate constraints such as meal diversity and a preference to avoid driving within branches. This local treatment, however, does not capture the complete scope of the shared budget constraint governing accommodation, meals, and attractions. Even if each task solution is locally reasonable, their combined cost may exceed the budget. Furthermore, these tasks may participate in overlapping constraints with different scopes, such as convenience and date consistency, compounding the difficulty of ensuring plan feasibility. Such multi-task constraint scopes align naturally with a hypergraph, whose nodes denote individual tasks while each hyperedge connects the exact task nodes governed by its corresponding constraint. Exposing this structure to LLMs makes constraint-induced task relations explicit, providing the structural guidance needed to coordinate task solutions into a globally feasible plan.

Translating this structural insight into an effective LLM planning framework raises the following question. *How can LLMs perceive and exploit the hypergraph structure implicit in a planning query to generate globally feasible plans?* Answering it involves two major challenges. (1) How to reliably induce the hypergraph from the original planning query? The task nodes, constraint hyperedges, and incidence relations of the constraint hypergraph are not explicitly annotated and must be recovered from natural language. (2) How to make the induced hypergraph accessible and actionable for plan generation? The induced hypergraph has a non-sequential structure that must be serialized, while overlapping hyperedges require consistent reasoning across coupled task solutions. Therefore, an effective framework requires faithful hypergraph induction and constraint-consistent reasoning.

To address these challenges, we propose **HyperPlan**, an LLM planning framework that models high-order relations among planning tasks as a constraint hypergraph and conditions plan generation on its structure. HyperPlan first extracts planning-relevant semantics from the original query. Based on these semantics, it constructs decision-oriented task nodes with canonical types and query-specific descriptions. HyperPlan then constructs constraint hyperedges by normalizing constraint-related semantics into query-specific types, assigning each constraint hyperedge an appropriate scope level, and identifying the relevant task nodes. Together, the constructed task nodes and constraint hyperedges form the final constraint hypergraph. Second, HyperPlan leverages the induced hypergraph to coordinate task reasoning and generate the final plan. It serializes the hypergraph into a sequential representation and performs constraint-aligned task reasoning with intra-node and inter-node consistency guidance. This helps to ensure that each task solution satisfies its incident constraints and that tasks sharing the same hyperedge remain mutually consistent. Finally, the coordinated task solutions are composed into a final plan.

Our main contributions are summarized as follows:

- We identify an underexplored limitation of existing LLM reasoning methods, which overlook high-order relations when decomposing planning objectives into subtasks, leading to globally infeasible plans.
- We formulate high-order relations as a constraint hypergraph and derive a sufficient condition for reducing LLM planning uncertainty through explicit constraint scopes.
- We propose HyperPlan, an LLM planning framework that induces constraint hypergraphs from planning queries and exploits their structure through constraint-centric serialization and constraint-aligned task reasoning.
- Extensive experiments demonstrate that HyperPlan achieves state-of-the-art performance among planning methods with LLMs as reasoners, improving success rates by up to 28.9% over strong baselines while maintaining inference cost efficiency.

2 Related Work

LLM Planning. Existing approaches improve LLM planning by structuring problem decomposition, reasoning trajectories, and task execution. Chain-of-Thought (CoT) and Plan-and-Solve Wei et al. (2022); Wang et al. (2023) introduce intermediate reasoning and decomposition, while Tree of Thoughts (ToT), Reasoning via Planning (RAP), and Graph of Thoughts (GoT) Yao et al. (2023a); Hao et al. (2023); Besta et al. (2024) organize reasoning through tree search, world models, or graph structures. ReAct and Language Agent Tree Search (LATS) Yao et al. (2023b); Zhou et al. (2024) incorporate tool interactions. Among planning-specific methods, EvoAgent, HyperTree Planning (HTP), ATLAS, and GraphPlan Yuan et al. (2025); Gui et al. (2025); Choi et al. (2026); Zhuang et al. (2027) use multi-agent collaboration, hypertree decomposition, constraint management, and executable task graphs, respectively. While the preceding methods retain the LLM as the final reasoner, LLM+P Liu et al. (2023) and Hao et al.'s framework Hao et al. (2025) use LLMs to formalize problems for external solvers to produce the plans. Despite these differences, the former methods primarily organize task decomposition, search, and execution without explicitly representing the complete scopes of shared constraints, leaving the induced high-order task relations implicit.

Hypergraphs for High-Order Relation Modeling. Hypergraphs naturally represent high-order relations by connecting multiple nodes within each hyperedge. Early spectral methods and subsequent neural architectures support hypergraph representation learning through incidence structures and message passing Zhou et al. (2006); Feng et al. (2019); Yadati et al. (2019); Huang & Yang (2021); Chien et al. (2022). Recent studies integrate hypergraphs with LLMs for structural alignment and representation learning Xu et al. (2025); Li et al. (2025), hypergraph understanding Lei et al. (2026); Wei et al. (2026), and knowledge organization and retrieval Feng et al. (2026); Luo et al. (2025). In contrast, HyperPlan represents the complete scopes of constraints shared across planning tasks and uses the induced hypergraph to coordinate task solutions.

3 Preliminary

3.1 LLM Planning

Given a planning query q, LLM planning aims to generate a complete plan $\mathcal{P}$ that fulfills the planning objective. Since complex planning queries typically involve multiple decisions, a common strategy is to decompose the overall objective into manageable subtasks and generate their corresponding solutions Wei et al. (2022); Yao et al. (2023a). The resulting subtasks form the planning **task** set $\mathcal{T}_q = \{t_i\}_{i=1}^N$, with the corresponding **task solution** set $\mathcal{A}_q = \{a_i\}_{i=1}^N$ subsequently composed into the final plan $\mathcal{P}$. Requirements that regulate these solutions, such as budget restrictions, form the **constraint** set $\mathcal{C}_q = \{c_j\}_{j=1}^M$. LLM planning can therefore be generally formulated as

$$\begin{aligned} (\mathcal{T}_q, \mathcal{C}_q) &= f_{\text{extract}}(q), \\ \mathcal{A}_q &= f_{\text{solve}}(q, \mathcal{T}_q, \mathcal{C}_q), \\ \mathcal{P} &= f_{\text{compose}}(q, \mathcal{A}_q, \mathcal{C}_q). \end{aligned} \tag{1}$$

Here, f_{extract} decomposes the planning query into tasks and constraints, f_{solve} generates the corresponding task solutions, and f_{compose} organizes these solutions into the final plan. These functions are typically performed by LLMs through corresponding prompting instructions.

Under this formulation, the planning objective is defined as

$$\mathcal{P}^* = \arg\max_{\mathcal{P}} \sum_{j=1}^{M} \text{Sat}(\mathcal{P}, c_j), \tag{2}$$

where $\text{Sat}(\mathcal{P}, c_j) \in \{0, 1\}$ indicates whether $\mathcal{P}$ satisfies c_j, and $\mathcal{A}_q^*$ denotes the task-solution set composing $\mathcal{P}^*$. The plan $\mathcal{P}^*$ is feasible if $\text{Sat}(\mathcal{P}^*, c_j) = 1$ for all $c_j \in \mathcal{C}_q$.

3.2 Problem Formulation

We first give the definition of a text-attributed hypergraph, which provides the structural basis for modeling high-order relations.

Definition 1 (Text-attributed Hypergraph). *A text-attributed hypergraph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_i\}_{i=1}^{N}$ and $\mathcal{E} = \{e_j\}_{j=1}^{M}$ denote its node and hyperedge sets. Each v_i and e_j is associated with a textual description t_i and c_j, respectively. Its incidence matrix $\mathbf{H} \in \{0,1\}^{N \times M}$ satisfies $\mathbf{H}_{ij} = 1$ if v_i is incident to e_j, and $\mathbf{H}_{ij} = 0$ otherwise.*

Based on this definition, a constraint hypergraph represents the constraint-induced high-order relations among planning tasks.

Definition 2 (Constraint Hypergraph). *Given a planning query q with task set $\mathcal{T}_q = \{t_i\}_{i=1}^{N}$ and constraint set $\mathcal{C}_q = \{c_j\}_{j=1}^{M}$, its constraint hypergraph is defined as $\mathcal{G}_q = (\mathcal{V}_q, \mathcal{E}_q)$. The **task node** set $\mathcal{V}_q = \{v_i\}_{i=1}^{N}$ associates each v_i with a task description $t_i \in \mathcal{T}_q$, while the **constraint hyperedge** set $\mathcal{E}_q = \{e_j\}_{j=1}^{M}$ associates each e_j with a constraint description $c_j \in \mathcal{C}_q$. The incidence matrix $\mathbf{H}_q \in \{0,1\}^{N \times M}$ satisfies $(\mathbf{H}_q)_{ij} = 1$ if c_j governs t_i, and $(\mathbf{H}_q)_{ij} = 0$ otherwise. Its j-th column therefore represents the **constraint scope** of c_j.*

Here, a constraint scope specifies the exact task nodes governed by a constraint. Accordingly, the problem is to induce $\mathcal{G}_q$ from q, with task nodes corresponding to planning decisions and constraint hyperedges representing constraint scopes. The induced hypergraph then guides the LLM to generate task solutions consistent with the constraints, which are composed into $\mathcal{P}^*$ defined in Eq. 2.

4 METHODOLOGY

Overview of HyperPlan. HyperPlan transforms a planning query into a feasible plan through the two stages illustrated in Figure 2. It first constructs task nodes and constraint hyperedges from the query, inducing a constraint hypergraph that captures high-order task relations. The induced hypergraph is then serialized to guide constraint-aligned task reasoning, after which the coordinated solutions are composed into the final plan. The complete algorithm is summarized in Appendix B.

4.1 CONSTRAINT HYPERGRAPH INDUCTION

Inducing a constraint hypergraph from a planning query is nontrivial, since neither task nodes nor constraint hyperedges have explicit ground-truth annotations. HyperPlan therefore induces the constraint hypergraph progressively through query semantic parsing, task node construction, and constraint hyperedge construction.

4.1.1 QUERY SEMANTIC PARSING

Directly mapping a raw planning query to task nodes and constraint hyperedges is unreliable, as it forces the model to segment the query and determine the structural role of each semantic unit simultaneously. Incorrect segmentation may subsequently produce missing, redundant, or conflicting tasks and constraints, thereby weakening the coordination among task solutions. HyperPlan therefore parses the query into planning-relevant semantic units before inducing the hypergraph. Specifically, it identifies semantic categories together with their corresponding values. The parsed semantics are represented as $\mathcal{S}_q = \{(r_k, x_k)\}_{k=1}^{K}$, where r_k denotes a semantic category and x_k denotes its value extracted from q.

4.1.2 TASK NODE CONSTRUCTION

Guided by the parsed semantics $\mathcal{S}_q$, HyperPlan first constructs the task node set $\mathcal{V}_q$. A task node represents a basic planning decision and thus corresponds to a planning subtask to be solved. To construct task nodes with consistent semantics and granularity, HyperPlan follows three principles.

- **Decision-oriented.** Each node corresponds to a concrete planning decision rather than a raw entity, such as *select accommodation in Detroit* instead of *Detroit*.
- **Canonical typing.** Each task node should be assigned a fixed canonical type y_i^t from a predefined domain-specific task type library $\mathcal{L}^{\mathcal{T}}$. Semantically equivalent task expressions share the same type. For example, *hotel* and *lodging* are both categorized under the predefined type *accommodation*.

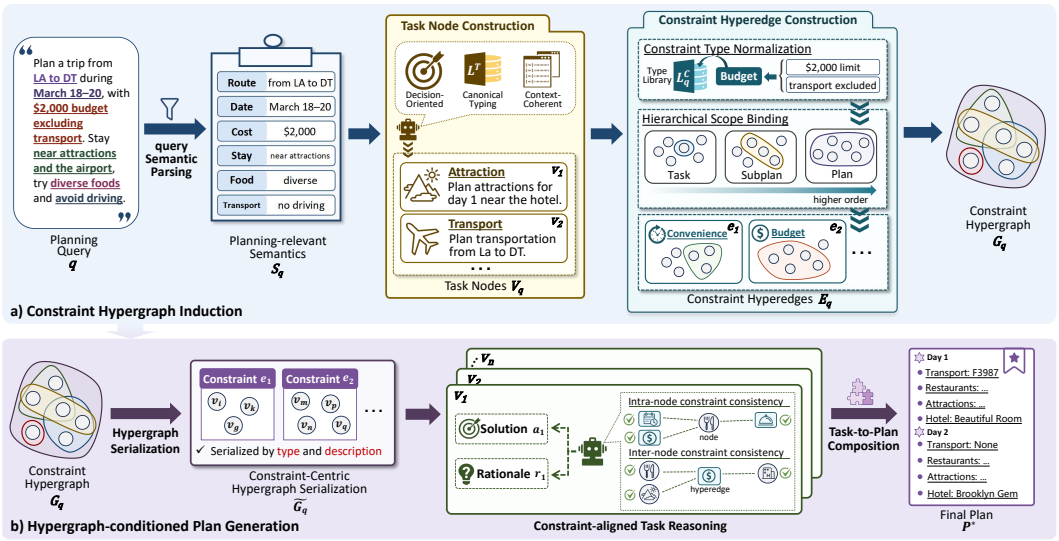


Figure 2: Pipeline of **HyperPlan**, consisting of (a) constraint hypergraph induction and (b) hypergraph-conditioned plan generation.

- **Context-coherent.** Contextual information associated with the same task node should be consolidated into a unified task description. For example, the location and duration associated with an *accommodation* task should be combined into the description *select accommodation in Detroit for two nights*.

Together, these principles yield the task node set $\mathcal{V}_q = \{v_i\}_{i=1}^{N}$, where each v_i has a canonical type $y_i^t \in \mathcal{L}^{\mathcal{T}}$ and a query-specific task description $t_i \in \mathcal{T}_q$.

4.1.3 Constraint Hyperedge Construction

Constraint hyperedges must capture high-order task relations for plan generation. HyperPlan constructs these hyperedges through constraint type normalization and hierarchical scope binding.

Constraint Type Normalization. Semantically related expressions may vary in form, causing a shared constraint to be fragmented into multiple hyperedges. To avoid this issue, the LLM normalizes the constraint-related semantics in $\mathcal{S}_q$ into a query-specific constraint type library $\mathcal{L}_q^{\mathcal{C}}$. Each type $y^c \in \mathcal{L}_q^{\mathcal{C}}$ denotes a functional category rather than a concrete constraint instance. For instance, *depart on March 18* and *return on March 20* are normalized under the same constraint type *date consistency*. Accordingly, each constraint hyperedge e_j is associated with a normalized type $y_j^c \in \mathcal{L}_q^{\mathcal{C}}$.

Hierarchical Scope Binding. Directly identifying each constraint scope requires searching over arbitrary task combinations, making it challenging for LLMs to accurately capture high-order relations. HyperPlan therefore introduces a hierarchical scope prior to guide scope binding. Each constraint hyperedge e_j is first assigned a scope level $\ell_j \in \{\text{task}, \text{subplan}, \text{plan}\}$. The corresponding task nodes are then identified within the assigned level.

The three scope levels are defined as follows:

- **Task level.** A constraint governs an individual task node, such as *hotel rating* for an accommodation task.
- **Subplan level.** A constraint governs a related subset of task nodes, such as *maximum daily travel time* for tasks within a specific day.
- **Plan level.** A constraint governs the entire set of task nodes, such as *total budget* for the complete plan.

For each constraint hyperedge e_j with normalized type y_j^c, HyperPlan identifies its incident task nodes based on the assigned level ℓ_j, thereby determining its constraint scope $\mathcal{V}_j \subseteq \mathcal{V}_q$. Using the corresponding constraint semantics in $\mathcal{S}_q$, it formulates a description c_j specifying how the constraint applies to $\mathcal{V}_j$. Together, these steps yield the constraint hyperedge set $\mathcal{E}_q = \{e_j\}_{j=1}^M$.

Together with the previously constructed task nodes $\mathcal{V}_q$, the constructed hyperedges determine the incidence matrix $\mathbf{H}_q$ and thereby complete the induction of $\mathcal{G}_q = (\mathcal{V}_q, \mathcal{E}_q)$. The incidence complexity is analyzed in Appendix D. The induced hypergraph is subsequently used for plan generation.

4.2 Hypergraph-Conditioned Plan Generation

After inducing $\mathcal{G}_q$, HyperPlan generates the final plan through constraint-centric serialization, constraint-aligned task reasoning, and task-to-plan composition.

4.2.1 Constraint-Centric Hypergraph Serialization

LLMs cannot directly process non-sequential structures, requiring the hypergraph to be serialized. HyperPlan therefore constructs a constraint-centric block $[e_j; \mathcal{V}_j]$ for each hyperedge and serializes the complete hypergraph as

$$\widetilde{\mathcal{G}}_q = [e_1; \mathcal{V}_1] \parallel \cdots \parallel [e_M; \mathcal{V}_M], \tag{3}$$

where $\parallel$ denotes block concatenation. Each block groups a constraint hyperedge with its incident task nodes, with both serialized by type and description. This serialization directly exposes each constraint scope while preserving overlaps through repeated nodes.

4.2.2 Constraint-Aligned Task Reasoning

Next, the LLM takes the complete serialized hypergraph $\widetilde{\mathcal{G}}_q$ as input to reason over all task nodes. $\widetilde{\mathcal{G}}_q$ provides only structural guidance rather than explicit enforcement of joint constraint satisfaction. A task solution may satisfy some incident constraints while violating others, and individually plausible solutions may jointly violate a shared constraint. HyperPlan therefore introduces two forms of consistency into task reasoning.

Intra-node constraint consistency. Each solution a_i should simultaneously satisfy all constraints represented by the hyperedges incident to v_i.

Inter-node constraint consistency. For each hyperedge e_j, the incident solution set $\mathcal{A}_q^{(e_j)} = \{a_i \mid v_i \in \mathcal{V}_j\}$ should be mutually compatible and jointly satisfy the shared constraint c_j.

To account for both forms of consistency, HyperPlan jointly generates a task solution a_i and a corresponding rationale r_i for each node:

$$\{(a_i, r_i)\}_{i=1}^N = f_{\text{solve}}\left(q, \widetilde{\mathcal{G}}_q\right). \tag{4}$$

Each rationale r_i explains how a_i maintains both forms of consistency. More importantly, it turns intra-node and inter-node consistency from implicit generation objectives into explicit alignment justifications, prompting the LLM to identify potential conflicts before incorporating the task solution into the final plan.

4.2.3 Task-to-Plan Composition

Finally, HyperPlan composes the coordinated task solutions $\mathcal{A}_q^* = \{a_i\}_{i=1}^N$ into the final plan according to the required planning structure. The original query q and constraint set $\mathcal{C}_q$ are jointly considered to preserve the constraint-aligned solutions established during task reasoning:

$$\mathcal{P}^* = f_{\text{compose}}\left(q, \mathcal{A}_q^*, \mathcal{C}_q\right). \tag{5}$$

4.3 Theoretical Analysis

In this section, we establish a sufficient condition for explicit constraint scopes in HyperPlan to reduce LLM planning uncertainty. We further show that hierarchical scope binding tightens the uncertainty bound. Across planning queries, let $\mathcal{S}_q^\star = \{\mathcal{V}_j^\star\}_{j=1}^M$ be the correct constraint-scope family,

Z_D the task-decomposition representation provided to the LLM, and Y a feasible plan satisfying all constraints; $H(\cdot \mid \cdot)$ denotes conditional entropy. Let $\epsilon = H(\mathcal{S}_q^\star \mid \widetilde{\mathcal{G}}_q)$ denote *Scope Uncertainty*, the residual uncertainty about the correct scopes, and $\delta = H(Y \mid Z_D) - H(Y \mid Z_D, \mathcal{S}_q^\star)$ denote *Scope Informativeness*, the planning uncertainty reduced by the correct scopes. Let Ω_H denote all possible combinations of constraint scopes remaining after hierarchical scope binding. The proof of the following theorem is in Appendix C.

Theorem 1. *If $\delta > \epsilon$, then $H(Y \mid Z_D, \widetilde{\mathcal{G}}_q) < H(Y \mid Z_D)$. Moreover, HyperPlan can represent scopes absent from Z_D, hence $\delta > 0$ when such scopes affect Y. Without hierarchical scope binding, $\epsilon \leq M \log(2^N - 1)$. With $\mathcal{S}_q^\star \in \Omega_H$, binding tightens this bound to $\epsilon \leq \log |\Omega_H| < M \log(2^N - 1)$.*

5 EXPERIMENTS

Tasks and Evaluation. We evaluate HyperPlan on four planning tasks from TravelPlanner Xie et al. (2024), Natural Plan Zheng et al. (2024), and PlanBench Valmeekam et al. (2023). We report Delivery Rate (DR), Commonsense Constraint Pass Rate (CPR), Hard Constraint Pass Rate (HCPR), and Success Rate (SR) for TravelPlanner, SR for the other tasks, and API cost for efficiency analysis. Dataset descriptions, statistics, and evaluation metrics are provided in Appendices F.1 and F.3.

Baselines and Implementation. We compare HyperPlan with four categories of representative baselines: (i) direct prompting with One-shot; (ii) general-purpose methods, including CoT Wei et al. (2022), ToT Yao et al. (2023a), and RAP Hao et al. (2023); (iii) planning-specific methods, including EvoAgent Yuan et al. (2025) and HTP Gui et al. (2025); and (iv) standalone open- and closed-source LLMs, including o3 OpenAI (2025), Claude Sonnet 5 Anthropic (2026), Gemini 2.5 Pro Gemini Team (2025), and DeepSeek-V4-Pro DeepSeek-AI (2026). We evaluate HyperPlan with GPT-4o OpenAI (2024), GPT-5-mini OpenAI (2026), and Gemini 2.5 Flash Gemini Team (2025), setting the temperature to 0 for all API calls. Baseline details and additional results with Qwen3 Qwen Team (2025) are provided in Appendices F.2 and E.1.

5.1 MAIN RESULTS

Table 1 compares HyperPlan with representative planning baselines across four planning tasks and three LLM backbones. The results yield three main findings.

HyperPlan consistently achieves the best performance across all evaluated LLM backbones and planning benchmarks. Specifically, it ranks first in every result, with SR gains ranging from 3.20% to 28.88% over the strongest baseline. Compared with HTP Gui et al. (2025), the strongest planning baseline in our experiments, HyperPlan improves the SR by 14.46%, demonstrating broad effectiveness across backbones and planning tasks.

HyperPlan delivers particularly substantial gains on complex planning tasks involving densely coupled constraints. On TravelPlanner, characterized by diverse and tightly coupled real-world constraints, HyperPlan achieves an average SR of 52.59%, compared with 27.96% for HTP, representing the largest improvement across the four tasks. This result highlights the benefit of modeling high-order task relations when interdependent decisions must satisfy multiple constraints.

HyperPlan effectively complements the capabilities of LLM backbones. Across GPT-4o, Gemini 2.5 Flash, and GPT-5-mini, HyperPlan yields relative improvements of 25.2%, 27.1%, and 27.8% over HTP, respectively, demonstrating stable gains across backbones with different levels of planning capability. HyperPlan therefore provides consistent structural guidance across models, leading to more reliable planning outcomes.

5.2 ABLATION STUDY

We compare HyperPlan with five ablated variants across four planning tasks and two backbones. Details are provided in Appendix F.4. Table 2 shows the results and yields three findings.

Consistency guidance is critical to coordinated task reasoning. Its removal causes the largest average performance degradation of 20.48%. Without explicit consistency guidance, locally plausible task solutions frequently conflict under shared constraints and fail to form a globally coherent plan.

Table 1: Main results (%) across four planning tasks and three LLM backbones. **Bold** and underlined values indicate the best and second-best results. Trip., Meet., and Mys. denote Trip Planning, Meeting Planning, and Mystery Blocksworld, respectively.

Backbone	Method	TravelPlanner						Trip.	Meet.	Mys.
		DR	CPR		HCPR		SR	SR	SR	SR
			Micro	Macro	Micro	Macro				
GPT-4o	One-shot	100	82.50	17.78	32.38	18.89	2.20	7.00	44.60	0.60
	CoT	100	84.17	20.00	36.19	20.00	2.70	9.44	45.10	1.20
	ToT	100	73.56	15.38	46.30	18.89	3.89	18.50	43.80	2.80
	RAP	100	86.41	26.11	43.46	20.56	4.44	18.75	46.20	4.40
	EvoAgent	100	80.95	19.44	45.16	23.89	8.33	12.25	47.70	2.60
	HTP	100	84.02	37.78	43.33	27.78	17.22	36.90	50.30	6.80
	HyperPlan	100	**91.18**	**51.67**	**81.42**	**62.22**	**33.89**	**40.25**	**53.50**	**11.60**
Gemini 2.5 Flash	One-shot	100	82.50	13.33	75.23	55.56	8.89	59.94	76.50	42.20
	CoT	100	85.83	26.67	72.38	40.00	17.22	63.13	81.20	46.60
	ToT	100	72.69	25.00	74.58	48.15	19.44	70.06	79.50	58.60
	RAP	100	83.52	36.67	55.55	36.67	18.33	68.13	77.60	61.00
	EvoAgent	100	88.07	35.56	60.43	34.44	21.67	66.38	80.90	57.40
	HTP	100	88.47	55.56	76.19	50.00	31.11	75.31	83.00	63.20
	HyperPlan	100	**92.78**	**62.22**	**84.29**	**73.89**	**59.44**	**88.81**	**91.70**	**81.20**
GPT-5-mini	One-shot	100	81.39	21.11	49.52	31.11	14.44	57.50	83.20	52.80
	CoT	100	81.66	30.00	45.24	22.22	16.67	63.25	79.70	67.10
	ToT	100	76.25	25.56	52.62	36.67	20.00	65.19	80.50	63.40
	RAP	100	87.50	37.22	60.32	41.11	23.80	67.25	86.60	80.10
	EvoAgent	100	86.66	35.56	69.93	50.00	25.00	71.19	84.50	71.20
	HTP	100	89.52	60.00	60.95	44.44	35.56	79.06	89.10	72.80
	HyperPlan	100	**95.83**	**71.11**	**83.33**	**68.89**	**64.44**	**98.69**	**94.70**	**95.60**

Table 2: Ablation results in SR (%) across four tasks and two backbones.

Variant	GPT-4o				Gemini 2.5 Flash			
	Travel.	Trip.	Meet.	Mys.	Travel.	Trip.	Meet.	Mys.
HyperPlan	**33.89**	**40.25**	**53.50**	**11.60**	**59.44**	**88.81**	**91.70**	**81.20**
w/o Consistency Guidance	11.11	27.50	42.70	3.50	22.22	48.75	71.40	69.40
w/o Incidence Relations	25.56	27.75	41.50	4.40	50.56	77.50	64.90	76.20
w/o Constraint Hyperedges	21.67	26.25	40.60	2.00	36.67	61.25	63.10	70.80
w/o Type Guidance	13.33	37.50	46.40	3.00	25.56	57.50	72.50	61.60
w/o Hypergraph Induction	13.33	31.25	47.10	0.60	37.78	66.63	67.70	73.80

Performance progressively declines as incidence relations and constraint hyperedges are removed. The variant without hyperedges consistently underperforms that without incidence relations, showing that losing constraint semantics is more detrimental than losing explicit scopes. This progressive decline confirms that HyperPlan depends on a complete hypergraph representation.

Constraint Hypergraph Induction is critical to the effectiveness of HyperPlan. Without this mechanism, hypergraphs may incompletely or incorrectly represent task dependencies, while removing its type-guidance component leads to inconsistent task granularity and constraint semantics. Appendix E.2 further examines the relation between induction correctness and planning success.

5.3 Further Analysis

Constraint Scope Analysis. To examine how scope-level determination affects hypergraph quality, we compare hierarchical scope binding with variants restricted to task, subplan, or plan scope levels, as shown in Table 3. Detailed settings are provided in Appendix F.6. Although all variants use the same constraints, hierarchical scope binding consistently performs best, showing that performance depends on how scope levels are assigned to constraints. This finding agrees with Theorem 1, since assigning each constraint to its appropriate level narrows its possible scope candidates, thereby tightening the uncertainty bound. A fixed level instead leaves broader constraint scopes incomplete or attaches local constraints to irrelevant tasks.

Hypergraph Serialization Analysis. Figure 3A reports serialization effects. Eq. 3 defines the default constraint-centric local expanded view by grouping each constraint with incident tasks in

Table 3: SR (%) under different levels of constraint scopes.

Scope Setting	GPT-4o		Gemini 2.5 Flash	
	Travel.	Trip.	Travel.	Trip.
Hierarchical Scope Binding	**33.89**	**40.25**	**59.44**	**88.81**
Only Task-Level Scope	26.67	35.31	56.11	73.06
Only Plan-Level Scope	24.44	36.06	55.00	72.94
Only Subplan-Level Scope	23.89	34.94	53.33	71.81

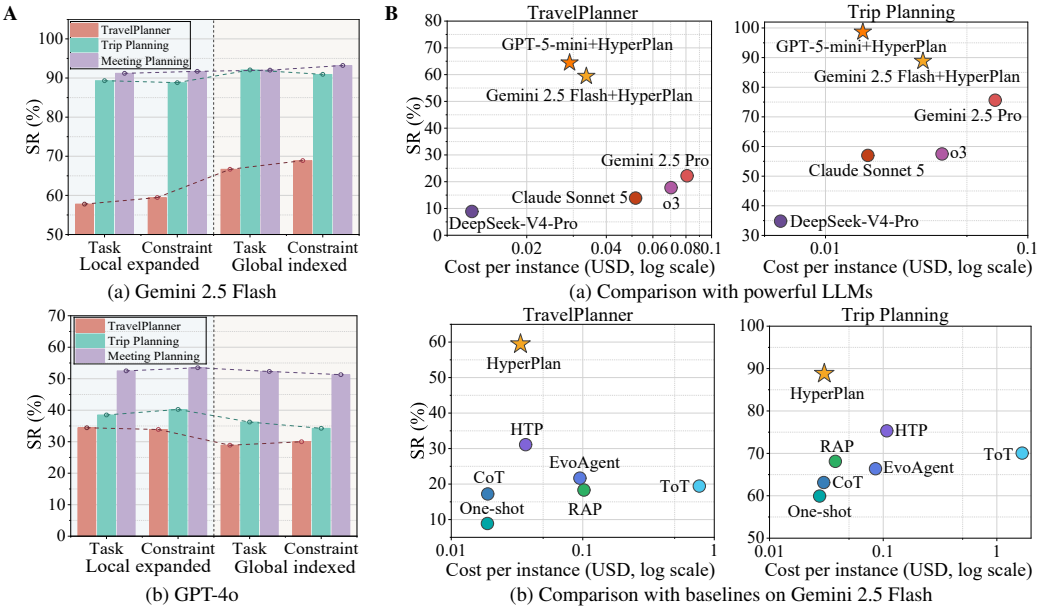


Figure 3: **(A)** SR (%) for schemes varying in serialization unit (task- or constraint-centric) and context layout (local expanded or global indexed). **(B)** SR (%) versus inference cost of HyperPlan, baselines and powerful LLMs. Complete results are reported in Appendix E.3.

one block. Varying its serialization unit and context layout gives three alternatives. A task-centric unit groups each task with its incident constraints, while the global indexed layout places tasks and constraints in separate blocks linked by indices. Details appear in Appendix F.5. Context layout affects SR more than serialization unit, but preferences depend on the backbone, with GPT-4o favoring local expansion and Gemini 2.5 Flash global indexing. Local expansion exposes incidence relations directly, whereas global indexing requires reconstructing task-constraint associations from indices. Thus, serialization should account for how each LLM processes relational information.

Cost-Effectiveness Analysis. Figure 3B plots success rate against average per-instance inference cost for HyperPlan, planning baselines, and powerful LLMs, with points closer to the upper-left indicating a better trade-off. Among planning baselines, HyperPlan is more costly only than One-shot and CoT, indicating modest overhead. When instantiated with Gemini 2.5 Flash or GPT-5-mini, HyperPlan achieves markedly higher success rates than powerful LLMs at costs lower than or comparable to those of closed-source models, while DeepSeek-V4-Pro incurs the lowest cost but exhibits the weakest performance. Collectively, these results demonstrate that structured constraint reasoning achieves a more favorable balance between planning performance and inference cost than increasing reasoning complexity or relying on model scale alone.

6 Conclusion

In this paper, we propose HyperPlan, an LLM planning framework that models high-order relations among planning tasks with a constraint hypergraph. HyperPlan induces the constraint hypergraph by constructing task nodes and constraint hyperedges and coordinates task solutions through serialization and constraint-aligned task reasoning. Theoretical analysis supports planning uncertainty reduction, while experiments demonstrate HyperPlan's effectiveness and inference cost efficiency.

AI USE STATEMENT

In this work, we used generative AI tools to assist with translation, interpret results, and assist in the writing of proofs. We have not used generative AI tools for other tasks with required disclosure. Additionally, we used generative AI tools to summarize or analyse existing literature, source or search for information, identify relevant literature, and format references. We reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Section 5 specifies the evaluated benchmarks and baselines, LLM backbones, decoding temperature, and evaluation metrics. The appendices provide the complete algorithm and theoretical proof, detailed benchmark and baseline descriptions, evaluation protocols, exact ablation and serialization settings, complete numerical results, and core prompt templates. The supplementary material also includes an anonymized implementation of HyperPlan.

REFERENCES

Anthropic. Introducing Claude Sonnet 5, 2026. URL https://www.anthropic.com/news/claude-sonnet-5.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving elaborate problems with large language models. In *AAAI*, pp. 17682–17690. AAAI Press, 2024.

Eli Chien, Chao Pan, Jianhao Peng, and Olgica Milenkovic. You are AllSet: A multiset function framework for hypergraph neural networks. In *ICLR*. OpenReview.net, 2022.

Jihye Choi, Jinsung Yoon, Jiefeng Chen, Somesh Jha, and Tomas Pfister. ATLAS: Constraints-aware multi-agent collaboration for real-world travel planning. In *ICLR*. OpenReview.net, 2026.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *CoRR*, abs/2606.19348, 2026.

Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. Hypergraph neural networks. In *AAAI*, pp. 3558–3565. AAAI Press, 2019.

Yifan Feng, Hao Hu, Shihui Ying, Xingliang Hou, Shiquan Liu, Mingyuan Yang, Junchang Li, Shaoyi Du, Nanning Zheng, Han Hu, et al. Hyper-RAG: Combating LLM hallucinations using hypergraph-driven retrieval-augmented generation. *Nature Communications*, 2026.

Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *CoRR*, abs/2507.06261, 2025.

Runquan Gui, Zhihai Wang, Jie Wang, Chi Ma, Huiling Zhen, Mingxuan Yuan, Jianye Hao, Defu Lian, Enhong Chen, and Feng Wu. HyperTree Planning: Enhancing LLM reasoning via hierarchical thinking. In *ICML*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *EMNLP*, pp. 8154–8173. Association for Computational Linguistics, 2023.

Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. Large language models can solve real-world planning rigorously with formal verification tools. In *NAACL (Long Papers)*, pp. 3434–3483. Association for Computational Linguistics, 2025.

Jing Huang and Jie Yang. UniGNN: a unified framework for graph and hypergraph neural networks. In *IJCAI*, pp. 2563–2569. ijcai.org, 2021.

Mengqi Lei, Guohuan Xie, Shihui Ying, Shaoyi Du, Jun-Hai Yong, Si-Qi Li, and Yue Gao. Hypergraph as language. *CoRR*, abs/2605.21858, 2026.

Ruochang Li, Xiao Luo, Zhiping Xiao, Wei Ju, and Ming Zhang. HEAL: hybrid enhancement with LLM-based agents for text-attributed hypergraph self-supervised representation learning. In *EMNLP (Findings)*, pp. 6815–6829. Association for Computational Linguistics, 2025.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: empowering large language models with optimal planning proficiency. *CoRR*, abs/2304.11477, 2023.

Haoran Luo, Haihong E, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and Anh Tuan Luu. HyperGraphRAG: Retrieval-augmented generation via hypergraph-structured knowledge representation. In *NeurIPS*, 2025.

OpenAI. GPT-4o system card. *CoRR*, abs/2410.21276, 2024.

OpenAI. OpenAI o3 and o4-mini system card, 2025. URL `https://openai.com/index/o3-o4-mini-system-card/`.

OpenAI. OpenAI GPT-5 system card. *CoRR*, abs/2601.03267, 2026.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo Hernandez, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *NeurIPS*, 2023.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve prompting: Improving zero-shot Chain-of-Thought reasoning by large language models. In *ACL (1)*, pp. 2609–2634. Association for Computational Linguistics, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Yanbin Wei, Chun Kang, Siwei Li, Haoxuan Che, Yang Chen, Hua Liu, Jian Liu, Zhuang Liu, Can Ouyang, Fei Xing, Lei Sha, Rui Liu, Yu Zhang, and James T. Kwok. HyperGVL: Benchmarking and improving large vision-language models in hypergraph understanding and reasoning. *CoRR*, abs/2604.15648, 2026.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54590–54613. PMLR / OpenReview.net, 2024.

Yao Xu, Shizhu He, Jiabei Chen, Xiangrong Zeng, Bingning Wang, Jun Zhao, and Kang Liu. LLaSA: Large language and structured data assistant. In *NAACL (Long Papers)*, pp. 1935–1946. Association for Computational Linguistics, 2025.

Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha P. Talukdar. HyperGCN: A new method for training graph convolutional networks on hypergraphs. In *NeurIPS*, pp. 1509–1520, 2019.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In *NeurIPS*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*. OpenReview.net, 2023b.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Dongsheng Li, and Deqing Yang. EvoAgent: Towards automatic multi-agent generation via evolutionary algorithms. In *NAACL (Long Papers)*, pp. 6192–6217. Association for Computational Linguistics, 2025.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. NATURAL PLAN: benchmarking LLMs on natural language planning. *CoRR*, abs/2406.04520, 2024.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *ICML*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR / OpenReview.net, 2024.

Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In *NIPS*, pp. 1601–1608. MIT Press, 2006.

Yiheng Zhuang, Yueying Liu, Wenlong Hou, Weidong Zhao, and Xianhui Liu. Planning as graphs: Adaptive agentic planning for LLM agents. *Expert Systems with Applications*, 333:133777, 2027.

A Future Work

Several directions could further improve HyperPlan in complex and interactive planning settings. Future work could explore iterative hypergraph induction, allowing task nodes, constraint hyperedges, and incidence relations to be refined through intermediate reasoning rather than fixed before solution generation. This could improve robustness as query complexity increases. HyperPlan could also incorporate ReAct-style tool use Yao et al. (2023b), enabling the LLM to query external sources and integrate retrieved information into hypergraph induction and task reasoning. This extension would broaden its applicability to planning problems requiring extensive information search.

B Algorithm of HyperPlan

Algorithm 1 presents the complete procedure of HyperPlan, including constraint hypergraph induction and hypergraph-conditioned plan generation.

Algorithm 1 Overall Procedure of HyperPlan

Input: Planning query q, task type library $\mathcal{L}^{\mathcal{T}}$

Extract planning semantics $\mathcal{S}_q$ from q
Construct $\mathcal{V}_q = \{v_i\}_{i=1}^{N}$ from $\mathcal{S}_q$ using $\mathcal{L}^{\mathcal{T}}$, associating each v_i with y_i^t and t_i
Normalize constraint semantics in $\mathcal{S}_q$ into $\mathcal{L}_q^{\mathcal{C}}$
for each concrete constraint instance j in $\mathcal{S}_q$ **do**
    Determine $y_j^c \in \mathcal{L}_q^{\mathcal{C}}$, $\ell_j \in \{\text{task}, \text{subplan}, \text{plan}\}$, and $\mathcal{V}_j \subseteq \mathcal{V}_q$
    Formulate c_j from $\mathcal{S}_q$ and $\mathcal{V}_j$
end for
Construct $\mathcal{E}_q = \{e_j\}_{j=1}^{M}$
Set $(\mathbf{H}_q)_{ij} \leftarrow \mathbf{1}[v_i \in \mathcal{V}_j]$ for all i, j and construct $\mathcal{G}_q \leftarrow (\mathcal{V}_q, \mathcal{E}_q)$
Serialize $\mathcal{G}_q$ by type and description into $\widetilde{\mathcal{G}}_q = [e_1; \mathcal{V}_1] \,\|\, \cdots \,\|\, [e_M; \mathcal{V}_M]$
Generate $\{(a_i, r_i)\}_{i=1}^{N} \leftarrow f_{\text{solve}}(q, \widetilde{\mathcal{G}}_q)$ under intra-node and inter-node consistency
Compose $\mathcal{P}^* \leftarrow f_{\text{compose}}\left(q, \{a_i\}_{i=1}^{N}, \{c_j\}_{j=1}^{M}\right)$

Output: Final plan $\mathcal{P}^*$

C Proof of Theorem 1

Definition 3 (Task-Decomposition Tree)**.** *A task-decomposition tree over $\mathcal{V}_q$ is a rooted tree $\mathcal{D}_q = (\mathcal{U}_q, u_q, \mathcal{F}_q)$, where $\mathcal{U}_q$ is the node set, $u_q \in \mathcal{U}_q$ is the root representing the planning objective, and $\mathcal{F}_q \subseteq \mathcal{U}_q \times \mathcal{U}_q$ is the set of parent-child relations. Each non-leaf node represents a planning unit decomposed into its children. The leaf set is exactly $\mathcal{V}_q$, with each task node appearing once.*

Definition 4 (Crossing Constraint Scopes)**.** *Two constraint scopes $\mathcal{V}_a, \mathcal{V}_b \subseteq \mathcal{V}_q$ are crossing if*

$$\mathcal{V}_a \cap \mathcal{V}_b \neq \emptyset, \qquad \mathcal{V}_a \not\subseteq \mathcal{V}_b, \qquad \mathcal{V}_b \not\subseteq \mathcal{V}_a.$$

Proof of Theorem 1. All variables are treated as discrete random variables over the planning-query distribution, and all logarithms use the same base as the entropy.

Planning uncertainty reduction. Recall that

$$\epsilon = H(\mathcal{S}_q^\star \mid \widetilde{\mathcal{G}}_q)$$

is *Scope Uncertainty*, measuring the uncertainty about the correct scopes remaining after observing the serialized hypergraph. Similarly,

$$\delta = H(Y \mid Z_D) - H(Y \mid Z_D, \mathcal{S}_q^\star)$$

is *Scope Informativeness*, measuring the planning uncertainty reduced by the correct scopes. By the chain rule and the fact that conditioning cannot increase entropy,

$$\begin{aligned} H(Y \mid Z_D, \widetilde{\mathcal{G}}_q) &\le H(Y, \mathcal{S}_q^\star \mid Z_D, \widetilde{\mathcal{G}}_q) \\ &= H(\mathcal{S}_q^\star \mid Z_D, \widetilde{\mathcal{G}}_q) + H(Y \mid Z_D, \widetilde{\mathcal{G}}_q, \mathcal{S}_q^\star) \\ &\le H(\mathcal{S}_q^\star \mid \widetilde{\mathcal{G}}_q) + H(Y \mid Z_D, \mathcal{S}_q^\star) \\ &= H(Y \mid Z_D) - \delta + \epsilon. \end{aligned}$$

Therefore, $\delta > \epsilon$ implies

$$H(Y \mid Z_D, \widetilde{\mathcal{G}}_q) < H(Y \mid Z_D).$$

Positive scope informativeness. Consider a query whose correct scope family contains two crossing scopes $\mathcal{V}_a$ and $\mathcal{V}_b$. For each node $u \in \mathcal{U}_q$, let

$$\mathrm{Desc}_{\mathcal{D}_q}(u) = \{v_i \in \mathcal{V}_q \mid v_i \text{ is a leaf in the subtree rooted at } u\}.$$

A decomposition unit identifies task membership through its subtree, so its explicitly represented scope is $\mathrm{Desc}_{\mathcal{D}_q}(u)$.

Suppose that the crossing scopes are represented by nodes u_a and u_b. Since $\mathcal{V}_a \cap \mathcal{V}_b \neq \emptyset$, some task leaf is a descendant of both nodes. The nodes therefore lie on the same root-to-leaf path, so one must be an ancestor of the other. If u_a is an ancestor of u_b, then

$$\mathcal{V}_b = \mathrm{Desc}_{\mathcal{D}_q}(u_b) \subseteq \mathrm{Desc}_{\mathcal{D}_q}(u_a) = \mathcal{V}_a,$$

contradicting $\mathcal{V}_b \not\subseteq \mathcal{V}_a$. Conversely, if u_b is an ancestor of u_a, then

$$\mathcal{V}_a = \mathrm{Desc}_{\mathcal{D}_q}(u_a) \subseteq \mathrm{Desc}_{\mathcal{D}_q}(u_b) = \mathcal{V}_b,$$

contradicting $\mathcal{V}_a \not\subseteq \mathcal{V}_b$. Thus, task decomposition cannot represent both crossing scopes simultaneously. A constraint hypergraph imposes no nesting requirement on its hyperedges, so HyperPlan represents both scopes through distinct hyperedges. It can therefore provide scope information absent from Z_D.

When this additional scope information affects Y,

$$P_{Y \mid Z_D, \mathcal{S}_q^\star} \neq P_{Y \mid Z_D}$$

on a set of positive probability, where $P_{Y|\cdot}$ denotes the conditional distribution of Y. This condition states that the scope information affects Y after Z_D is given.

By the conditional mutual-information identity,

$$I(Y; \mathcal{S}_q^\star \mid Z_D) = \mathbb{E}\left[D_{\mathrm{KL}}\left(P_{Y \mid Z_D, \mathcal{S}_q^\star} \,\middle\|\, P_{Y \mid Z_D} \right) \right],$$

where $\mathbb{E}$ denotes expectation and $D_{\mathrm{KL}}(\cdot\|\cdot)$ denotes KL divergence. Since the two conditional distributions differ on a set of positive probability, Gibbs' inequality gives

$$I(Y; \mathcal{S}_q^\star \mid Z_D) > 0.$$

Consequently,

$$\begin{aligned} \delta &= H(Y \mid Z_D) - H(Y \mid Z_D, \mathcal{S}_q^\star) \\ &= I(Y; \mathcal{S}_q^\star \mid Z_D) \\ &> 0. \end{aligned}$$

Scope uncertainty bounds. Without hierarchical scope binding, each of the M constraints may govern any nonempty subset of the N task nodes. Define the unrestricted candidate space as

$$\Omega_0 = \left(2^{\mathcal{V}_q} \setminus \{\emptyset\}\right)^M,$$

where $2^{\mathcal{V}_q}$ denotes the power set of $\mathcal{V}_q$. Since one constraint has $2^N - 1$ possible nonempty scopes,

$$|\Omega_0| = (2^N - 1)^M.$$

The entropy of a discrete variable cannot exceed the logarithm of its support size. Therefore,

$$\epsilon \leq \log |\Omega_0| = M \log(2^N - 1).$$

Under hierarchical scope binding, let Ω_{ℓ_j} denote the candidate constraint scopes permitted for constraint hyperedge e_j at its assigned level ℓ_j, and define

$$\Omega_H = \prod_{j=1}^{M} \Omega_{\ell_j}.$$

Thus, Ω_H contains all scope-family candidates remaining after binding, while $\mathcal{S}_q^\star \in \Omega_H$ ensures that the correct scope family is retained.

Each hierarchical level excludes task subsets inconsistent with its scope granularity. Hence,

$$\Omega_{\ell_j} \subsetneq 2^{\mathcal{V}_q} \setminus \{\emptyset\},$$

which gives

$$|\Omega_H| = \prod_{j=1}^{M} |\Omega_{\ell_j}| < (2^N - 1)^M.$$

Applying the maximum-entropy bound to the restricted candidate space yields

$$\epsilon \leq \log |\Omega_H| < M \log(2^N - 1).$$

Therefore, hierarchical scope binding tightens the scope-uncertainty bound while retaining the correct scope family. □

D Incidence Complexity Analysis of the Constraint Hypergraph

The structural scale of a constraint hypergraph can be characterized by its number of node-hyperedge incidences. For a constraint hypergraph $\mathcal{G}_q$ with N task nodes and M constraint hyperedges, the incidence count is

$$I_q = \sum_{j=1}^{M} |\mathcal{V}_j| = \sum_{j=1}^{M} \sum_{i=1}^{N} (\mathbf{H}_q)_{ij}.$$

Under constraint-centric serialization, each incidence causes one task node to be serialized within its corresponding hyperedge block. The total number of serialized task-node occurrences is therefore I_q. Since every hyperedge is serialized once, the total number of serialized task-node and hyperedge occurrences is $M + I_q$. Although the exact token count also depends on the lengths of their textual descriptions, I_q directly characterizes the structural repetition introduced during serialization.

A task-level strategy connects each constraint to one task node and therefore produces $I_{\text{task}} = M$ incidences. Although structurally compact, it cannot preserve a shared constraint as a complete scope over multiple tasks. A plan-level strategy connects every constraint to all task nodes, producing $I_{\text{plan}} = MN$ incidences. This exhaustive expansion preserves shared relations but repeatedly introduces task nodes unrelated to individual constraints. A subplan-level strategy connects each constraint to an arbitrary subset of task nodes. With an average subset size K, its incidence count is $I_{\text{sub}} = KM$, where $1 < K < N$. However, it requires the appropriate subset to be identified directly from arbitrary task combinations for every constraint.

Hierarchical scope binding first assigns each constraint to the task, subplan, or plan level and then identifies its incident task nodes within the assigned level. Let M_t, M_s, and M_p denote the numbers of constraints assigned to these levels, where $M_t + M_s + M_p = M$. A task-level constraint contributes one incidence, a subplan-level constraint contributes K incidences on average, and a plan-level constraint contributes N incidences. The resulting incidence count is

$$I_{\text{HSB}} = M_t + KM_s + NM_p,$$

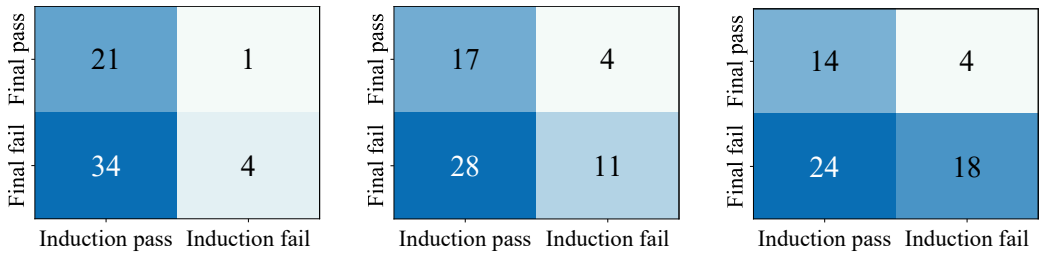
where $1 < K < N$. It consequently satisfies

$$M \leq I_{\text{HSB}} \leq MN.$$

Hierarchical scope binding therefore adjusts the incidence scale according to the scope level assigned to each constraint. Compared with task-level binding, it preserves the required high-order constraint relations. Compared with plan-level binding, it avoids repeatedly serializing every task node under every constraint, thereby reducing redundant relational context.

Table 4: Results (%) with Qwen3-235B-A22B across four planning tasks.

Method	TravelPlanner						Trip.	Meet.	Mys.
	DR	CPR		HCPR		SR	SR	SR	SR
		Micro	Macro	Micro	Macro				
CoT	100	84.17	20.00	44.76	26.67	6.67	24.56	39.80	7.80
ToT	100	77.79	23.89	47.84	21.11	7.22	18.06	40.20	8.20
One-shot	100	83.89	13.33	51.58	15.56	4.45	17.94	33.90	3.80
RAP	100	82.24	15.79	61.43	42.63	7.22	29.13	41.70	11.20
EvoAgent	100	83.35	25.55	49.12	28.33	8.89	25.31	39.30	11.00
HTP	100	84.79	37.78	41.19	26.67	15.00	33.06	43.50	11.00
HyperPlan	100	**89.44**	**43.33**	**71.19**	**56.11**	**25.00**	**46.88**	**48.50**	**13.90**



(a) Easy (b) Medium (c) Hard

Figure 4: Joint outcomes of hypergraph induction correctness and final planning success on TravelPlanner with GPT-4o across three difficulty levels. Each level contains 60 queries.

E Additional Results

E.1 Additional Results with Qwen3

Table 4 provides additional results using the open-source Qwen3-235B-A22B as the backbone. HyperPlan consistently outperforms the planning baselines across all four tasks, further demonstrating that its performance gains generalize to an open-source backbone.

E.2 Hypergraph Induction Quality Analysis

We manually inspect the hypergraphs induced by HyperPlan with GPT-4o on all 180 TravelPlanner queries. Final planning success follows the TravelPlanner criterion used to compute SR, while induction correctness is evaluated using four criteria. (i) The task nodes cover all required activities across all days. (ii) The constraint hyperedges capture all requirements in the query. (iii) The incidence relations associate each constraint with the appropriate tasks according to the query and commonsense. (iv) The types and descriptions of all tasks and constraints are complete and accurate. Figure 4 reports the joint outcomes across the three difficulty levels.

Both induction accuracy and final SR decrease with query difficulty. From easy to hard, induction accuracy decreases from 91.67% to 75.00% and 63.33%, while SR decreases from 36.67% to 35.00% and 30.00%. Despite this decline, the overall induction accuracy reaches 76.67%, compared with an overall SR of 33.89%, indicating that the induction procedure remains reliable even when downstream planning fails. Moreover, 78.57% of queries with failed induction also fail in final planning, while successful induction yields an SR of 37.68%, compared with 21.43% under failed induction. These results show a clear positive association between induction correctness and planning success, supporting hypergraph induction as an important component of HyperPlan, although correct induction alone does not guarantee a successful plan.

E.3 Complete Results for Figure 3

Table 5 reports the complete serialization results underlying Figure 3A. Each row corresponds to one combination of serialization unit and context layout defined in Appendix F.5.

Table 5: Complete success rates (%) for the serialization schemes in Figure 3A.

Serialization Scheme	Gemini 2.5 Flash			GPT-4o		
	Travel.	Trip.	Meet.	Travel.	Trip.	Meet.
Task-Centric Local Expanded	57.78	89.31	91.20	**34.44**	38.50	52.50
Constraint-Centric Local Expanded	59.44	88.81	91.70	33.89	**40.25**	**53.50**
Task-Centric Global Indexed	66.67	**92.06**	92.00	28.89	36.25	52.30
Constraint-Centric Global Indexed	**68.89**	90.94	**93.20**	30.00	34.25	51.30

Table 6: Success rates (%) and average per-instance inference costs (USD) in Figure 3B.

Method	TravelPlanner		Trip Planning	
	SR	Cost	SR	Cost
Comparison with powerful LLMs				
Gemini 2.5 Flash + HyperPlan	59.44	0.03360	88.81	0.03032
GPT-5-mini + HyperPlan	64.44	0.02905	98.69	0.01530
o3	17.78	0.07028	57.50	0.03768
Claude Sonnet 5	13.89	0.05160	57.00	0.01619
Gemini 2.5 Pro	22.22	0.08079	75.63	0.06899
DeepSeek-V4-Pro	8.89	0.01240	34.81	0.00600
Comparison with planning baselines on Gemini 2.5 Flash				
HyperPlan	59.44	0.03360	88.81	0.03032
CoT	17.22	0.01893	63.13	0.02998
ToT	19.44	0.76971	70.06	1.67104
One-shot	8.89	0.01883	59.94	0.02757
RAP	18.33	0.10208	68.13	0.03794
EvoAgent	21.67	0.09515	66.38	0.08554
HTP	31.11	0.03678	75.31	0.10763

Table 6 gives the complete success rates and average per-instance inference costs underlying Figure 3B. The first panel compares two HyperPlan configurations with standalone LLMs. The second compares HyperPlan with planning baselines using Gemini 2.5 Flash.

F Experimental Settings

F.1 Details of Benchmarks

We provide detailed descriptions of the three benchmarks used in our experiments, spanning real-world itinerary construction, natural-language planning, and symbolic action-sequence generation.

- **TravelPlanner** Xie et al. (2024) is designed for complex travel planning with long horizons, multiple constraints, and fine-grained evaluation metrics. Its instances are divided into nine groups by trip duration (3, 5, and 7 days) and difficulty level (easy, medium, and hard). We evaluate on the full validation set of 180 examples, consisting of 20 examples from each group.

- **Natural Plan** Zheng et al. (2024) evaluates natural-language planning through multiple tasks, from which we select Trip Planning and Meeting Planning. Trip Planning constructs multi-city itineraries and includes 1,600 examples across eight city-count levels. Meeting Planning schedules meetings across locations to maximize the number of people met and includes 1,000 examples across ten participant-count levels.

- **PlanBench** Valmeekam et al. (2023) evaluates classical planning over domains from the International Planning Competition. We focus on its obfuscated Blocksworld setting, which requires valid action sequences under misleading action and predicate names, and use 500 instances with at least four blocks.

F.2 Details of Baselines

We provide detailed descriptions of the planning baselines used in our experiments, covering direct prompting, general-purpose problem-solving methods, and planning-specific methods.

- **One-shot Prompting** is a direct prompting baseline that provides one in-context example and asks the LLM to generate the final plan directly, without explicit decomposition, search, or iterative refinement.
- **CoT** Wei et al. (2022) elicits a linear sequence of intermediate steps before the final output, enabling the LLM to construct a plan through step-by-step generation.
- **ToT** Yao et al. (2023a) organizes intermediate thoughts into a search tree, where multiple candidates are generated and evaluated through lookahead and backtracking to identify a promising solution path.
- **RAP** Hao et al. (2023) employs the LLM as both an agent and a world model, and uses Monte Carlo Tree Search with task-specific rewards to explore and evaluate candidate trajectories.
- **EvoAgent** Yuan et al. (2025) automatically extends a specialized agent into a multi-agent system through evolutionary operations, including mutation, crossover, and selection, to generate agents with diverse configurations.
- **HTP** Gui et al. (2025) constructs hypertree-structured planning outlines to hierarchically decompose complex tasks, and iteratively expands and refines them to coordinate multiple constraints and subtasks.

F.3 Details of Evaluation Metrics

We follow the official evaluation protocols and use benchmark-specific success criteria.

TravelPlanner. TravelPlanner is evaluated using four metrics.

- **Delivery Rate** measures the proportion of plans completed within the maximum number of steps, which is set to 30. Failures include dead loops, repeated unsuccessful attempts, and exceeding the step limit.
- **Commonsense Constraint Pass Rate** measures the satisfaction of commonsense requirements in the generated plans.
- **Hard Constraint Pass Rate** measures the satisfaction of explicit constraints specified in the planning query.
- **Success Rate** measures the proportion of complete and viable plans satisfying all evaluation criteria.

Commonsense Constraint Pass Rate and Hard Constraint Pass Rate are reported under both micro and macro evaluation. Micro evaluation computes the proportion of individual constraints satisfied, whereas macro evaluation computes the proportion of plans satisfying all constraints in the corresponding category.

Other Planning Tasks. Trip Planning, Meeting Planning, and Mystery Blocksworld are evaluated using Success Rate, defined as the proportion of instances meeting the corresponding task-specific success criterion. Trip Planning and Meeting Planning require the unique target solution. Mystery Blocksworld plans are validated by VAL and succeed upon reaching the goal, irrespective of optimality.

API Cost. API cost is computed from the input and output tokens consumed using the official pricing of each provider. It provides a more stable efficiency measure than wall-clock time, which is affected by network and provider-side conditions.

F.4 Ablation Settings

We evaluate five ablated variants of HyperPlan, each removing a specific source of structural or reasoning guidance while retaining the remaining planning procedure.

- **w/o Consistency Guidance.** The explicit intra-node and inter-node consistency instructions, including the rationale-based justification of constraint alignment, are removed from Constraint-Aligned Task Reasoning. The LLM still receives the complete serialized hypergraph and generates all task solutions, isolating the contribution of explicit consistency guidance.
- **w/o Incidence Relations.** The LLM retains all task-node contents and constraint-hyperedge descriptions, while the incidence relations specifying which task nodes are governed by each constraint are omitted during plan generation.
- **w/o Constraint Hyperedges.** Only the task-node contents are retained during Hypergraph-Conditioned Plan Generation. Constraint types, constraint descriptions, and their associated incidence relations are all removed.
- **w/o Type Guidance.** The domain-specific task type library is removed from Task Node Construction, and Constraint Type Normalization is omitted from Constraint Hyperedge Construction. Task nodes are constructed without canonical task types, while constraint hyperedges are constructed without normalized constraint types.
- **w/o Hypergraph Induction.** The explicit Constraint Hypergraph Induction procedure is removed. The LLM directly maps the planning query to a hypergraph under the same output schema, without explicit guidance for query semantic parsing, task node construction, constraint type normalization, or hierarchical scope binding. The resulting hypergraph is subsequently serialized and used for the same task reasoning procedure.

F.5 Hypergraph Serialization Schemes

Hypergraph serialization varies along two dimensions that determine how graph elements and their incidence relations are organized in a sequential representation.

- **Serialization Unit.** This dimension determines the anchor of incidence representation. A task-centric view enumerates the constraints incident to each task, whereas a constraint-centric view enumerates the tasks incident to each constraint.
- **Context Layout.** This dimension determines how graph elements are grouped. Local expansion treats each serialization unit as a block and embeds its incident elements within it, whereas global indexing groups task nodes and hyperedges by element type and represents their incidence through indices.

Combining the two serialization units with the two context layouts yields four views, as described below. The default serialization defined in the methodology corresponds to the Constraint-Centric Local Expanded View.

- **Task-Centric Local Expanded View.** Each task node is serialized with the complete contents of its incident constraint hyperedges. This view directly presents all constraints connected to each task, while shared constraints are repeated across the corresponding task-centric blocks.
- **Task-Centric Global Indexed View.** All constraint hyperedges are listed first, followed by the task nodes and the indices of their incident hyperedges. Each index j refers to the corresponding hyperedge in the preceding global sequence, preserving task-centric relations without repeating constraint contents.
- **Constraint-Centric Local Expanded View.** Each constraint hyperedge is serialized with the complete contents of its incident task nodes. This view directly presents all tasks connected by each constraint, while shared task nodes are repeated across the corresponding constraint-centric blocks.

- **Constraint-Centric Global Indexed View.** All task nodes are listed first, followed by the constraint hyperedges and the indices of their incident nodes. Each index i refers to the corresponding task node in the preceding global sequence, preserving constraint-centric relations without repeating task contents.

All four views encode the same task nodes, constraint hyperedges, and incidence relations, differing only in their sequential organization and relation presentation.

F.6 Constraint-Scope Variants

We evaluate three scope-restricted variants that preserve the same constraint content as HyperPlan while confining constraint binding to a single granularity. The variants differ only in scope representation and binding granularity.

- **Only Task-Level Scope.** During Constraint Hypergraph Induction, every constraint is assigned a task-level scope and bound to exactly one task node. Constraint applicability is therefore represented exclusively through node-specific associations.
- **Only Plan-Level Scope.** Every constraint is assigned a plan-level scope and treated as a global requirement governing the complete plan. No constraint is associated with an individual node or a selected subset of nodes.
- **Only Subplan-Level Scope.** Every constraint is assigned a subplan-level scope and bound to a group of related task nodes. Such groups may consist of tasks within the same day or tasks sharing the same semantic type, representing constraint applicability through group-level relationships.

G Case Study

We compare the reasoning processes and final outcomes of HyperPlan and HTP on a representative TravelPlanner instance. This example illustrates how each method organizes interacting constraints and translates its reasoning into the final itinerary.

Original Query

Can you assist in crafting a travel itinerary for a pair of travelers originating from Milwaukee and visiting 2 cities in Michigan? The trip is 5 days long, spanning from March 21st to March 25th, 2022. Our budget is now set at $1,700, and we require accommodations that allow parties and provide private rooms. Moreover, we'd prefer if our transportation did not involve any flights.

G.1 HyperPlan

For readability, we present the induced constraint hypergraph together with the jointly generated task solutions and their rationales in a unified representation. The resulting task solutions are subsequently composed into the final plan.

Constraint Hypergraph with Joint Task Solutions

nodes:
  node_id: 1
  task_type: transportation
  content: Decide transportation from Milwaukee to the first Michigan visiting city.
  answer_list:
    self−driving, from Milwaukee to Alpena, duration: 7 hours 37 mins, distance: 749 km, cost: 37
  reason: Self−driving is the cheapest option and avoids flights, fitting the transportation preference constraint.

node_id: 2
task_type: transportation
content: Decide transportation from the first Michigan visiting city to the second Michigan visiting city.
answer_list:
self−driving, from Alpena to Kalamazoo, duration: 4 hours 33 mins, distance: 464 km, cost: 23
reason: Self−driving is consistent with the first leg's mode and is the cheapest option available.

node_id: 3
task_type: transportation
content: Decide transportation from the second Michigan visiting city back to Milwaukee.
answer_list:
self−driving, from Kalamazoo to Milwaukee, duration: 3 hours 45 mins, distance: 382 km, cost: 19
reason: Self−driving is consistent with the previous legs and is the cheapest option available.

node_id: 4
task_type: meal
content: Decide restaurants and meal choices in the first Michigan visiting city.
answer_list:
Sahni Fish Corner, Alpena, Desserts, Pizza, Bakery, Fast Food, Indian, Average Cost: 11
Jain Chawal Wale, Alpena, Desserts, Pizza, Italian, Mexican, Cafe, Average Cost: 20
Chennai Dosa Express, Alpena, Tea, Pizza, Mexican, BBQ, Bakery, American, Average Cost: 20
Cafe Coffee Day, Alpena, Desserts, Tea, French, BBQ, Bakery, American, Average Cost: 22
Shashi's China Wok, Alpena, Tea, Bakery, BBQ, Desserts, Average Cost: 29
Kylin Express, Alpena, Chinese, Bakery, BBQ, Seafood, Average Cost: 30
reason: Selected restaurants in Alpena offer a variety of cuisines at the lowest average costs, covering meals for the stay.

node_id: 5
task_type: meal
content: Decide restaurants and meal choices in the second Michigan visiting city.
answer_list:
Tamasha In Tafree, Kalamazoo, Desserts, Italian, French, Bakery, Seafood, Average Cost: 11
Kolkata Kathi Roll, Kalamazoo, Tea, Chinese, BBQ, Cafe, Average Cost: 11
Ruchi's Food Junction, Kalamazoo, Desserts, Cafe, Pizza, Fast Food, Chinese, Average Cost: 15
Six Degrees, Kalamazoo, Cafe, Italian, Bakery, Fast Food, Chinese, Average Cost: 20
Boheme Bar & Grill, Kalamazoo, Bakery, BBQ, Seafood, Average Cost: 26
Al Bake, Kalamazoo, Cafe, BBQ, Italian, Average Cost: 36
reason: Selected restaurants in Kalamazoo offer a variety of cuisines at the lowest average costs, covering meals for the stay.

node_id: 6
task_type: attraction
content: Decide attractions to visit in the first Michigan visiting city.
answer_list:
Great Lakes Maritime Heritage Center, Alpena
Besser Museum for Northeast Michigan, Alpena
Mich−e−ke−wis Park, Alpena

reason: Selected attractions in Alpena are popular and provide a variety of experiences, fitting the stay duration.

node_id: 7
task_type: attraction
content: Decide attractions to visit in the second Michigan visiting city.
answer_list:
Kalamazoo Valley Museum, Kalamazoo
Kalamazoo Institute of Arts, Kalamazoo
Milham Park, Kalamazoo
reason: Selected attractions in Kalamazoo are popular and provide a variety of experiences, fitting the stay duration.

node_id: 8
task_type: accommodation
content: Decide accommodation in the first Michigan visiting city.
answer_list:
SINGLE ROOM AVAILABLE IN JAMAICA NEAR JFK & LGA, Alpena, Private room, No visitors, Minimum nights: 1, Maximum occupancy: 2, Price: 88
reason: This accommodation in Alpena allows parties, offers a private room, and is the cheapest option that fits the constraints.

node_id: 9
task_type: accommodation
content: Decide accommodation in the second Michigan visiting city.
answer_list:
Apartment in Ridgewood/Bushwick Neighborhood, Kalamazoo, Private room, No smoking, Minimum nights: 2, Maximum occupancy: 2, Price: 152
reason: This accommodation in Kalamazoo allows parties, offers a private room, and is the cheapest option that fits the constraints.

hyperedges:
edge_id: 1
constraint_type: within_city
node_ids: 1, 2, 4, 6, 8
content: The arrival transportation, departure transportation, meals, attractions, and accommodation for the first Michigan visiting city must refer to the same city.

edge_id: 2
constraint_type: within_city
node_ids: 2, 3, 5, 7, 9
content: The arrival transportation, return transportation, meals, attractions, and accommodation for the second Michigan visiting city must refer to the same city.

edge_id: 3
constraint_type: date_and_route_consistency
node_ids: 1, 2, 3, 8, 9
content: The three transportation legs and two accommodation choices must jointly fit the March 21 to March 25, 2022 trip and a valid two−city route.

edge_id: 4
constraint_type: accommodation_requirement
node_ids: 8, 9
content: Both accommodations should be private rooms and allow parties.

edge_id: 5
constraint_type: transportation_preference

node_ids: 1, 2, 3
content: All transportation choices should avoid flights.

edge_id: 6
constraint_type: transportation_non_conflict
node_ids: 1, 2, 3
content: All transportation nodes must follow the first route leg's mode policy: if the first leg uses self−driving, every later leg must use self−driving; if the first leg uses Flight or Taxi, no later leg may use self−driving.

edge_id: 7
constraint_type: meals_number
node_ids: 1, 2, 4
content: Meal choices in the first visiting city should cover meals needed during its stay days, including breakfast, lunch, and dinner for each day of staying, considering arrival and departure transportation time.

edge_id: 8
constraint_type: meals_number
node_ids: 2, 3, 5
content: Meal choices in the second visiting city should cover meals needed during its stay days, including breakfast, lunch, and dinner for each day of staying, considering arrival and return transportation time.

edge_id: 9
constraint_type: attraction_number
node_ids: 1, 2, 6
content: Attraction choices in the first visiting city should cover its stay days, with fewer attractions allowed on heavy transportation days.

edge_id: 10
constraint_type: attraction_number
node_ids: 2, 3, 7
content: Attraction choices in the second visiting city should cover its stay days, with fewer attractions allowed on heavy transportation days.

edge_id: 11
constraint_type: minimum_nights_stay
node_ids: 1, 2, 8
content: The number of consecutive nights spent in the first visiting city must be greater than or equal to the selected accommodation's minimum nights field.

edge_id: 12
constraint_type: minimum_nights_stay
node_ids: 2, 3, 9
content: The number of consecutive nights spent in the second visiting city must be greater than or equal to the selected accommodation's minimum nights field.

edge_id: 13
constraint_type: budget
node_ids: 1, 2, 3, 4, 5, 8, 9
content: Transportation, meals, and accommodations for 2 people should fit the $1,700 budget.

Final Plan

plan:
days: 1
current_city: from Milwaukee to Alpena
transportation: self−driving, from Milwaukee to Alpena, duration: 7 hours 37 mins, distance: 749 km, cost: 37
breakfast: −
lunch: Sahni Fish Corner, Alpena
dinner: Jain Chawal Wale, Alpena
attraction: Great Lakes Maritime Heritage Center, Alpena;
accommodation: SINGLE ROOM AVAILABLE IN JAMAICA NEAR JFK & LGA, Alpena

days: 2
current_city: Alpena
transportation: −
breakfast: Chennai Dosa Express, Alpena
lunch: Cafe Coffee Day, Alpena
dinner: Shashi's China Wok, Alpena
attraction: Besser Museum for Northeast Michigan, Alpena; Mich−e−ke−wis Park, Alpena;
accommodation: SINGLE ROOM AVAILABLE IN JAMAICA NEAR JFK & LGA, Alpena

days: 3
current_city: from Alpena to Kalamazoo
transportation: self−driving, from Alpena to Kalamazoo, duration: 4 hours 33 mins, distance: 464 km, cost: 23
breakfast: Kylin Express, Alpena
lunch: Tamasha In Tafree, Kalamazoo
dinner: Kolkata Kathi Roll, Kalamazoo
attraction: Kalamazoo Valley Museum, Kalamazoo;
accommodation: Apartment in Ridgewood/Bushwick Neighborhood, Kalamazoo

days: 4
current_city: Kalamazoo
transportation: −
breakfast: Ruchi's Food Junction, Kalamazoo
lunch: Six Degrees, Kalamazoo
dinner: Boheme Bar & Grill, Kalamazoo
attraction: Kalamazoo Institute of Arts, Kalamazoo; Milham Park, Kalamazoo;
accommodation: Apartment in Ridgewood/Bushwick Neighborhood, Kalamazoo

days: 5
current_city: from Kalamazoo to Milwaukee
transportation: self−driving, from Kalamazoo to Milwaukee, duration: 3 hours 45 mins, distance: 382 km, cost: 19
breakfast: Al Bake, Kalamazoo
lunch: −
dinner: −
attraction: −
accommodation: −

G.2 HTP

Initial Thought

First, I extract some basic information from the query: 'days': 5, 'visiting_city_number': 2, 'people_number': 2, 'preference':{{'house rule': 'parties', 'cuisine': None, 'room type': 'private room', 'transportation': 'no flight'}} Break down the travel planning into four parts: transportation, attractions, dining, and accommodation.

For the transportation part:
The transportation I need to consider involves three parts: from Milwaukee to Alpena on 2022−03−21, from Alpena to Kalamazoo on 2022−03−23, and from Kalamazoo to Milwaukee on 2022−03−25.

For the transportation from Milwaukee to Alpena on 2022−03−21:
I observe that there is no flight, but self−driving and taxis are available. The user prefers no flights, so I must choose between self−driving or a taxi. I should choose the one that costs less. Calculate the cost of choosing the self−driving option: $37. Calculate the cost of choosing the taxi option: $749. I will choose the self−driving option and submit: "self−driving, from Milwaukee to Alpena, duration: 7 hours 37 mins".

For the transportation from Alpena to Kalamazoo on 2022−03−23:
I observe that there is no flight, but self−driving and taxis are available. The user prefers no flights, so I must choose between self−driving or a taxi. I should choose the one that costs less. Calculate the cost of choosing the self−driving option: $23. Calculate the cost of choosing the taxi option: $464. I will choose the self−driving option and submit: "self−driving, from Alpena to Kalamazoo, duration: 4 hours 33 mins".

For the transportation from Kalamazoo to Milwaukee on 2022−03−25:
I observe that there is no flight, but self−driving and taxis are available. The user prefers no flights, so I must choose between self−driving or a taxi. I should choose the one that costs less. Calculate the cost of choosing the self−driving option: $19. Calculate the cost of choosing the taxi option: $382. I will choose the self−driving option and submit: "self−driving, from Kalamazoo to Milwaukee, duration: 3 hours 45 mins".

For the attractions part:
The attractions I need to consider involve two parts: attractions in Alpena and attractions in Kalamazoo.

For the attractions in Alpena:
I am in Alpena from 2022−03−21 to 2022−03−22, so I should choose two attractions in Alpena. I will submit: "Great Lakes Maritime Heritage Center, Alpena" for day 1 and "Besser Museum for Northeast Michigan, Alpena" for day 2.

For the attractions in Kalamazoo:
I am in Kalamazoo from 2022−03−23 to 2022−03−24, so I should choose two attractions in Kalamazoo. I will submit: "Kalamazoo Valley Museum, Kalamazoo" for day 3 and "Kalamazoo Institute of Arts, Kalamazoo" for day 4.

For the dining part:
I need to ensure that dining is covered for all 5 days. The user has not mentioned any specific cuisine preferences, so I will select based on price and availability.

For the dining in Alpena from 2022−03−21 to 2022−03−22:
Since breakfast and lunch on the first day do not need to be considered, I need to account for 4 meals. Select the four lowest−cost restaurants. Submit: "Jain Chawal Wale, Alpena" for dinner on day 1 and "Sahni Fish Corner, Alpena", "Cafe Coffee Day, Alpena", "Shudh Restaurant, Alpena" for meals on day 2.

For the dining in Kalamazoo from 2022−03−23 to 2022−03−24:
Choose 6 different restaurants considering cost−effectiveness. Submit: "Tamasha In Tafree, Kalamazoo", "Kolkata Kathi Roll, Kalamazoo", "Ruchi's Food Junction, Kalamazoo" for day 3, and "Boheme Bar & Grill, Kalamazoo", "Al Bake, Kalamazoo", "Six Degrees, Kalamazoo" for day 4.

For the dining in Kalamazoo for breakfast on day 5:
Choose based on cost−effectiveness and availability. Submit: "Django, Kalamazoo".

For the accommodation part:
The accommodation I need to consider involves two parts: accommodation in Alpena and in Kalamazoo.

For the accommodation in Alpena from 2022−03−21 to 2022−03−22:
The user requests private rooms that allow parties, the accommodations that meet the criteria are: "#NAME?", and "Huge Bedroom w/direct access to backyard". Among the filtered, I should choose the one with the lowest price: "#NAME?" is priced at $839. We need 1 room, the total price is $839. "Huge Bedroom w/direct access to backyard" is priced at $635. We need 1 room, the total price is $635. Based on the calculations, I will submit: "Huge Bedroom w/direct access to backyard, Alpena".

For the accommodation in Kalamazoo from 2022−03−23 to 2022−03−24:
The user requests private rooms that allow parties. Potential accommodations include: "Clean design, One BD, Renovated!" and "Comfy Basement Studio". Among the filtered, I should choose the one with the lowest price: "Clean design, One BD, Renovated!" is priced at $1033. We need 1 room, the total price is $1033. "Comfy Basement Studio" is priced at $1118. We need 1 room, the total price is $1118. Based on the calculations, I will submit: "Clean design, One BD, Renovated!, Kalamazoo".

Final Plan

plan:
  days: 1
  current_city: from Milwaukee to Alpena
  transportation: self−driving, from Milwaukee to Alpena, duration: 7 hours 37 mins
  breakfast: −
  attraction: Great Lakes Maritime Heritage Center, Alpena
  lunch: −
  dinner: Jain Chawal Wale, Alpena
  accommodation: Huge Bedroom w/direct access to backyard, Alpena

  days: 2
  current_city: Alpena
  transportation: −
  breakfast: Sahni Fish Corner, Alpena
  attraction: Besser Museum for Northeast Michigan, Alpena
  lunch: Cafe Coffee Day, Alpena
  dinner: Shudh Restaurant, Alpena
  accommodation: Huge Bedroom w/direct access to backyard, Alpena

  days: 3
  current_city: from Alpena to Kalamazoo
  transportation: self−driving, from Alpena to Kalamazoo, duration: 4 hours 33 mins
  breakfast: Tamasha In Tafree, Kalamazoo
  attraction: Kalamazoo Valley Museum, Kalamazoo

lunch: Kolkata Kathi Roll, Kalamazoo
dinner: Ruchi's Food Junction, Kalamazoo
accommodation: Clean design, One BD, Renovated!, Kalamazoo

days: 4
current_city: Kalamazoo
transportation: –
breakfast: Boheme Bar & Grill, Kalamazoo
attraction: Kalamazoo Institute of Arts, Kalamazoo
lunch: Al Bake, Kalamazoo
dinner: Six Degrees, Kalamazoo
accommodation: Clean design, One BD, Renovated!, Kalamazoo

days: 5
current_city: from Kalamazoo to Milwaukee
transportation: self−driving, from Kalamazoo to Milwaukee, duration: 3 hours 45 mins
breakfast: Django, Kalamazoo
attraction: –
lunch: –
dinner: –
accommodation: –

Evaluation Errors

errors:
constraint_group: Commonsense Constraint
constraint_name: Minimum Nights Stay
reason: The accommodation Clean design, One BD, Renovated!, Kalamazoo do not obey the minumum nights rule.

constraint_group: Hard Constraint
constraint_name: Room Rule
reason: The house rule should be parties, but the accommodation(Huge Bedroom w/direct access to backyard, Alpena) does not meet this condition.

constraint_group: Hard Constraint
constraint_name: Room Type
reason: The room type should be private room, but the accommodation(Clean design, One BD, Renovated!, Kalamazoo) does not meet this condition.

constraint_group: Hard Constraint
constraint_name: Budget
reason: The most expensive cost is: accommodation(Huge Bedroom w/direct access to backyard, Alpena)

G.3 COMPARISON

HTP decomposes the query by planning component but handles the associated constraints largely within separate branches, resulting in violations of the party, room-type, minimum-night, and budget requirements. HyperPlan instead connects each constraint to all affected tasks through explicit hyperedges, enabling their joint consideration during task reasoning. Consequently, the composed plan satisfies the interacting constraints that HTP fails to coordinate.

H PROMPT TEMPLATES

We provide the two core prompt templates used by HyperPlan on TravelPlanner.

Constraint Hypergraph Induction Prompt

```
Build a constraint hypergraph for the travel−planning query.
Return only valid JSON. No markdown.
The graph is a task−constraint representation:
− nodes are planning decision units that require answers.
− hyperedges are constraints that restrict one or more related task nodes.
Required JSON schema:
{
  "nodes": [
    {
      "node_id": string containing only digits, e.g. "1",
      "task_type": one of "meal", "attraction", "accommodation", "transportation",
      "content": short clear task description
    }
  ],
  "hyperedges": [
    {
      "edge_id": string containing only digits, e.g. "1",
      "constraint_type": string,
      "node_ids": [node_id, ...],
      "content": short clear constraint description
    }
  ]
}
−Building Steps−
1. Query Structure Extraction.
Extract the task−structural information needed for graph construction from the current query.
− Identify the planning entities that affect later decisions: origin, destination region, candidate
visiting cities, number of visiting cities, travelers, date range, budget, route requirements, and
local preference constraints.
− Identify which information should become task−level decisions and which information
should become constraints over those decisions.
− Use the current query only.

2. Task Node Construction.
Construct task nodes as decision units in the planning process.
− Nodes should represent planning tasks that require concrete answers, not sentences, entities,
or constraint descriptions.
− node_id values must be "1", "2", "3", ... with no semantic words.
− Task_type must be one of "meal", "attraction", "accommodation", or "transportation".
− Each node should correspond to a coherent decision that will later produce an answer, such
as selecting transportation for a route leg, meals for a city stay, attractions for a city stay, or
accommodation for a city stay.
− Do not create nodes for within−city transportation.
− Put semantic meaning in task_type and content, not in node_id.
− Do not generate graph_id, answer_list, or reason.

3. Constraint Hyperedge Construction.
Organize query constraints into hyperedges and explicitly ground each constraint to the task
nodes it affects.
− Identify a query−specific set of normalized constraint types from the query semantics.
− Merge functionally equivalent requirements under the same constraint type.
− Define each constraint type narrowly enough to have one clear applicability scope.
```

– Use the normalized constraint types only as internal guidance for controlling the semantic specificity of concrete hyperedges.
– Keep constraint_type as the semantic label of each concrete hyperedge required by the JSON schema.
– Allow one constraint type to produce multiple hyperedges when the query contains distinct constraint instances.
– Instantiate concrete hyperedges from the normalized constraint types and query semantics.
– Bind each concrete hyperedge to one scope level: task, subplan, or plan.
– Task scope means the constraint governs exactly one task node.
– Subplan scope means the constraint governs a coherent subset of related task nodes, but not the complete set of task nodes.
– Plan scope means the constraint governs the complete set of task nodes and coordinates the planning objective as a whole.
– For each concrete constraint instance, connect its hyperedge to all and only the task nodes governed by that instance within the assigned level.
– Do not output the normalized type inventory or scope levels as separate intermediate results.
– Write each hyperedge content in concise, executable language that states how the constraint governs its connected nodes.
– Every hyperedge must reference existing node_id values.
– Use numeric edge_id values in sequential order.
– Put semantic meaning in constraint_type and content, not in edge_id.
– Prefer compact hyperedges while preserving all constraints required for subsequent task reasoning and final plan generation.
Query:
{query}

Hypergraph-Conditioned Plan Solving Prompt

Solve all task nodes in the constraint hypergraph using TravelPlanner reference data.
Return only valid JSON. No markdown.
Output a JSON list. Each item must have exactly:
– node_id: a node id in the graph.
– answer_list: a list of direct answers for this task node.
– reason: one short sentence explaining why this node answer is consistent with all incident hyperedge constraints.

Rules:
– Use only node_id values that appear in the graph.
– Use exact names/records from reference data for restaurants, attractions, accommodations, flights, taxi, or self−driving.
– Infer answers from the query and constraints.
– Do not invent city names, restaurant names, attraction names, accommodation names, flight numbers, dates, or prices.
– answer_list should contain direct answers only, not explanations.
– reason must be one short sentence with no line breaks and no quotation marks.
– In reason, explicitly state why this node answer does not violate the incident hyperedges connected to this node.
– If several nodes share a hyperedge, make their node answers mutually compatible under that shared constraint.
– Do not treat node answers as isolated local choices; each answer must be justified against its connected hyperedges.

Required answer content by task_type:
– meal: each selected answer should include restaurant name, city, cuisine information, and average cost if available.
– attraction: each selected answer should include attraction name and city.

− accommodation: each selected answer should include accommodation name, city, room type, house rules, minimum nights, maximum occupancy, and price if available.
− transportation: each selected answer should include transportation mode, from city, to city, and the exact reference fields available for that mode, such as flight number/time/price or taxi/self−driving duration/distance/cost.
Original query:
{query}
Constraint−Centric Local Expanded View:
{hypergraph_view_json}
Reference data:
{reference_data_json}