REASON, EXPLORE, VERIFY: A NEURO-SYMBOLIC ARCHITECTURE FOR CONSTRAINED LONG-HORIZON PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) can interpret natural-language goals and generate plausible action sequences, but remain unreliable for constrained long-horizon planning. Errors can accumulate across decisions, locally plausible actions can eliminate feasible continuations, and generated plans can violate numerical, logical, or temporal requirements. We introduce *Reason–Explore–Verify* (REV), a neuro-symbolic architecture that combines hierarchical problem formulation, explicit trajectory exploration, and formal verification of candidate decisions. We instantiate REV as *Hierarchical Monte Carlo Tree Search with SMT Gating* (HMCTS-SMT), which integrates hierarchical LLM agents, LLM-guided Monte Carlo tree search, and Satisfiability Modulo Theories solving. Each search node maintains an agentic state and a branch-specific symbolic constraint state. A selected action creates a successor only if its symbolic update preserves satisfiability, making symbolic verification a mandatory transition condition rather than an optional tool or post hoc check. We evaluate HMCTS-SMT on supply-chain optimization and travel planning, covering optimization-oriented and feasibility-oriented settings. Across these tasks, HMCTS-SMT improves plan validity and solution quality over direct generation approach and search-based LLM baselines. These results demonstrate the effectiveness of combining hierarchical decomposition, explicit exploration, and transition-level symbolic verification for constrained long-horizon planning.

1 INTRODUCTION

Planning is a foundational problem in artificial intelligence (Fikes & Nilsson, 1971). Classical methods systematically construct action sequences but require formal domain specifications, often expressed in PDDL (Ghallab et al., 1998), which are costly to build from natural-language requirements. Large Language Models (LLMs) can interpret goals, decompose tasks, and propose actions (Brown et al., 2020), with reasoning-oriented models further improving multi-step inference (DeepSeek-AI, 2025). However, standalone LLMs remain unreliable for constrained, long-horizon planning. Early errors may violate numerical, logical, temporal, or domain constraints, eliminate feasible continuations, or yield suboptimal plans. Search, tools, and decomposition allow reconsideration, validation, and reduced complexity, but search may expand invalid states, subproblems may conflict, and terminal verification detects infeasibility only after computation. We instead propose a neuro-symbolic architecture in which formal constraints are incorporated inside state-space exploration.

1.1 LIMITATIONS OF CURRENT LLM PLANNING

Linguistically plausible trajectories are not necessarily valid plans. Long-horizon planning requires state tracking, coordinated decisions, and constraint satisfaction throughout execution. A locally suitable action may consume future resources or remove all feasible continuations while the trajectory remains coherent (Kambhampati et al., 2024).

Search-based agents preserve alternatives, but learned evaluations, simulations, and environment feedback do not establish formal feasibility. Constrained planning requires an environment-facing

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.

state, branch-specific constraints, and a distinction between feasibility and utility. Terminal verification cannot prevent exploration below inconsistent prefixes. Therefore, symbolic satisfiability should determine whether a successor exists, while neural priors and search values rank admissible successors.

1.2 Solution and Contributions

We introduce *Reason–Explore–Verify* (REV), a neuro-symbolic architecture coordinating three functions. *Reason* converts natural-language requirements into a structured model and hierarchical subgoals; *Explore* searches alternative trajectories; and *Verify* checks candidate decisions before admitting them to the search tree. Thus, neural models interpret, search explores, and symbolic solvers enforce constraints.

We instantiate REV as *Hierarchical Monte Carlo Tree Search with SMT Gating* (HMCTS-SMT). A global LLM builds the symbolic model and decomposition, while a local LLM proposes actions that guide PUCT-based search. Each node contains an environment-facing state, active subgoal, and branch-local symbolic representation. For action a, a successor exists only if the accumulation of constraints is satisfiable. Consequently, an inconsistent action cannot be admitted by a high prior, value, or exploration bonus. MCTS ranks admissible actions by progress and objective quality. Lazy verification limits checks to selected candidates, while bounded model revision addresses systematic modeling or decomposition errors.
Our contributions are:

1. A joint planning state combining environment state, hierarchical subgoal, and branch-local symbolic constraints.
2. Verifier-authoritative transitions separating SMT-based admissibility from LLM- and MCTS-based utility.
3. HMCTS-SMT framework integrating symbolic modeling, hierarchical decomposition, local action generation, PUCT, incremental SMT checking, and terminal constrained optimization.

2 Related Work

2.1 LLM Planning and Search

LLM agents support goal interpretation, decomposition, action generation, and feedback-driven refinement. Chain-of-Thought elicits intermediate reasoning (Wei et al., 2023); ReAct interleaves reasoning, actions, and observations (Yao et al., 2023b); Reflexion uses verbal feedback across attempts (Shinn et al., 2023); and Voyager learns reusable skills through environment interaction (Wang et al., 2023). These methods improve adaptation but do not provide any formal feasibility guarantee. The Tree of Thoughts and The Graph of Thoughts introduce branching and structured refinement (Yao et al., 2023a; Besta et al., 2024). LATS combines MCTS with LLM actions, feedback, evaluation, and reflection (Zhou et al., 2024), while SPIRAL assigns planning, simulation, and criticism to specialized agents within MCTS (Zhang et al., 2026). Their intermediate signals estimate trajectory quality but do not prove consistency with branch-local constraints. HMCTS-SMT instead rejects every UNSAT action independently of its estimated value and organizes planning through decision-oriented subgoals.

2.2 LLMs and Symbolic Solvers

LLMs generate formalizations and candidates, while symbolic solvers provide deterministic reasoning and optimization. Logic-LM follows a formalize-then-solve pipeline (Pan et al., 2023), and neuro-symbolic compliance systems use SMT to verify constraints and diagnose inconsistencies (Hsia et al., 2026). These methods generally verify a generated representation or completed candidate rather than maintaining evolving theories across search branches.

Relation to verifier-guided tree search. VerMCTS is the closest algorithmic predecessor (Brandfonbrener et al., 2024). It uses formal verification to reject invalid partial Dafny or Coq programs

during MCTS. Accordingly, we do not claim the first verifier-guided tree search. HMCTS-SMT instead searches planning trajectories whose nodes combine an environment state, subgoal, and branch-local theory. A natural-language action induces new constraints and the satisfiability of the past accumulated constraints with the new one determines whether its successor exists. This introduces planning-specific problems, including environment-symbol correspondence, update fidelity, hierarchical progress, and optimization among feasible trajectories. The SMT solver makes a binary, non-compensatory admissibility decision, while LLM priors and MCTS values rank admissible actions. Terminal optimization is separate because feasible plans may differ in quality. The contribution is thus the extension of verifier-guided MCTS to hierarchical constrained planning over coupled agentic and symbolic states, where satisfiability defines the feasible edge set and neural search operates within it.

3 PRELIMINARIES

3.1 MONTE CARLO TREE SEARCH

Monte Carlo Tree Search (MCTS) incrementally explores sequential decision problems by representing states as nodes and actions as edges (Kocsis & Szepesvári, 2006). Each iteration selects a path that balances exploitation and exploration, expands a candidate action, evaluates the resulting state, and backpropagates its value through the selected path. When action priors are available, Predictor UCT (PUCT) selects actions according to

$$a^* = \arg\max_{a \in \mathcal{A}(s)} \left[Q(s,a) + c_{\text{puct}}\, P(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)} \right], \tag{1}$$

where $Q(s,a)$ is the estimated action value, $P(s,a)$ is its prior, $N(s)$ is the parent visit count, and $N(s,a)$ is the action visit count (Rosin, 2011; Silver et al., 2016). In HMCTS-SMT, the local LLM agent provides $P(s,a)$, while search outcomes determine $Q(s,a)$.

3.2 SATISFIABILITY MODULO THEORIES

Satisfiability Modulo Theories (SMT) extends Boolean satisfiability to formulas over theories such as arithmetic, arrays, and uninterpreted functions (Kroening & Strichman, 2016; Barrett et al., 2021). Modern SMT solvers combine Boolean search with theory-specific consistency checking, commonly through the DPLL(T) architecture (Bofill et al., 2008). They are widely used in formal verification, program analysis, and model checking (Beyer et al., 2018; Ponce-de-León et al., 2020; Cordeiro et al., 2009). In HMCTS-SMT, the solver determines whether a candidate transition is consistent with the symbolic constraints accumulated along its search branch.

4 REASON–EXPLORE–VERIFY ARCHITECTURE

We introduce REASON–EXPLORE–VERIFY (REV), a neuro-symbolic architecture for constrained long-horizon planning. REV separates semantic interpretation, trajectory exploration, and formal admissibility, rather than assigning problem formulation, action generation, search, and feasibility assessment to a single autoregressive model.

4.1 PROBLEM SETTING

We define constrained planning instances as $x = (\mathcal{D}, q, E)$ where $\mathcal{D}$ is a natural-language domain description, q is the planning request, and E optionally provides grounded entities, observations, actions, and domain information. The objective is to construct a plan: $\pi = (a_0, a_1, \ldots, a_{T-1})$ with environment-facing transitions $e_{t+1} = \mathcal{T}_E(e_t, a_t)$. The instance is associated with a symbolic model $\mathcal{M}_0 = (\mathcal{V}, \mathcal{Y}, \mathcal{C}_{\text{base}}, f, \eta)$ where $\mathcal{V}$ contains symbolic variables, $\mathcal{Y}$ defines their types and domains, $\mathcal{C}_{\text{base}}$ contains base constraints, f is an optional objective, and $\eta \in \{\min, \max, \varnothing\}$ specifies its direction or absence. All verification claims are relative to the coverage and fidelity of $\mathcal{M}_0$ and of the symbolic action updates introduced below.

4.2 REV ARCHITECTURE

REV coordinates three functions Reason, explore and verify. The architecture is implementation-independent. Reason may use one or more language models, Explore may use tree or graph search, and Verify may use SAT, SMT, constraint programming, theorem proving, or a domain-specific checker.

Reason: Constructs the symbolic model and subgoal hierarchy, then proposes candidate actions. It constructs $\mathcal{M}_0$ by extracting variables, domains, constraints, and objectives from x. For example, a budget requirement may be encoded as: $\sum_{t=0}^{T-1} \mathrm{cost}(a_t) \leq B$

Constraints may retain provenance to their natural-language requirements, supporting inspection and revision. Reason also constructs an ordered blueprint: $\mathcal{B} = (b_1, b_2, \ldots, b_L)$ where each b_j is a decision-oriented subgoal. The blueprint limits the local decision context, preserves dependencies, and defines planning progress. Verification and optimization remain responsibilities of Verify.

For a planning state s and active subgoal b_j, Reason proposes at most k actions:

$$\mathcal{A}_\theta(s, b_j, x) = \{a^1, a^2, \ldots, a^k\}. \tag{2}$$

Each candidate is represented as

$$a^i = (d^i, \Delta^i, p^i, m^i), \tag{3}$$

where d^i is its natural-language description, Δ^i is its symbolic update, $p^i \in [0, 1]$ is a heuristic prior, and m^i contains optional grounded metadata. Reason estimates relevance, but it does not determine formal admissibility.

Explore: Branch-Specific Planning States

Explore searches over joint agentic and symbolic states

$$s_t = (e_t, \mathcal{C}_t, \mathcal{B}, j_t, \mathcal{H}_t, \mu_t), \tag{4}$$

where e_t is the environment-facing state, $\mathcal{C}_t$ contains the symbolic updates committed on the branch, j_t identifies the active subgoal, $\mathcal{H}_t$ is the action history, and μ_t is an optional satisfying assignment. The corresponding branch theory is

$$\Phi(s_t) = \left(\bigwedge_{c \in \mathcal{C}_{\text{base}}} c \right) \wedge \left(\bigwedge_{\delta \in \mathcal{C}_t} \delta \right). \tag{5}$$

Base constraints remain fixed within a search epoch, while each branch accumulates its own updates. Explore selects a state and a candidate, but only Verify may admit the resulting successor. Alternative branches allow search to redirect computation when a decision is infeasible or leads to low utility.

Verify: Transition Admissibility

For state s_t and candidate a_t, a transition gate is defined $G(s_t, a_t) = \mathbb{I}\big[\mathrm{SAT}\big(\Phi(s_t) \wedge \Delta(a_t)\big)\big]$.
If $G(s_t, a_t) = 1$, the branch is updated as $\mathcal{C}_{t+1} = \mathcal{C}_t \cup \{\Delta(a_t)\}$ and $\mathcal{H}_{t+1} = \mathcal{H}_t \,\|\, a_t$. and a successor is created: $s_{t+1} = (e_{t+1}, \mathcal{C}_{t+1}, \mathcal{B}, j_{t+1}, \mathcal{H}_{t+1}, \mu_{t+1})$. If $G(s_t, a_t) = 0$, the action is rejected and no successor exists. Thus, verification defines the feasible edge set rather than serving only as a terminal test.

The gate is distinct from search utility. Let $U(s, a)$ denote an estimate of progress or objective quality. The gate determines reachability, whereas Explore uses U, action priors, and search statistics to rank admissible alternatives. Consequently, high estimated utility cannot compensate for an UNSAT transition, while SAT alone does not imply goal completion or high solution quality.

This guarantee is conditional on action-update fidelity. In particular, $\Delta(a_t)$ must faithfully represent the relevant consequences of $\mathcal{T}_E(e_t, a_t)$. REV guarantees consistency with the supplied symbolic update, but does not by itself prove correspondence between the symbolic and environment-facing transitions.

Terminal evaluation. A state s_T is terminal when all subgoals are complete. Feasibility-oriented tasks require $\mathrm{SAT}\big(\Phi(s_T)\big)$. For optimization tasks, Verify computes

$$f^\star(s_T) = \begin{cases} \min\limits_{\mathbf{v}\in\mathcal{V}} f(\mathbf{v}), & \eta = \min, \\ \max\limits_{\mathbf{v}\in\mathcal{V}} f(\mathbf{v}), & \eta = \max, \end{cases} \quad \text{subject to } \Phi(s_T). \tag{6}$$

This value is exact for the terminal branch theory, but does not imply global optimality over trajectories omitted by decomposition, candidate generation, or finite search.

4.3 Feedback and Bounded Revision

A local verification failure rejects only its transition. Repeated rejection of credible candidates for a required subgoal may instead indicate an incorrect encoding, missing variable, unsuitable decomposition, or incomplete environment information. Explore and Verify then return a structured report to Reason containing the failed subgoal, rejected updates, active constraints, relevant observations, and an unsatisfiable core when available. Reason may revise the symbolic model, blueprint, or candidate-generation strategy.

Revisions are bounded, and explicit user requirements remain immutable, although their encodings may be repaired. Because revision changes the active model, search restarts in a new epoch and retained plans must be revalidated before reuse or comparison.

5 REV instantiation: Hierarchical Monte Carlo Tree Search

We instantiate REV as *Hierarchical Monte Carlo Tree Search with SMT Gating* (HMCTS-SMT) with: Reason as **global and local LLM agents**, Explore as **PUCT-guided tree search**, and Verify as **SMT-gated transitions and terminal solving**.

The global agent constructs the symbolic model and subgoal hierarchy, the local agent proposes actions for the active subgoal, and MCTS allocates search among them. The SMT solver defines admissibility, so neural priors and search values cannot override symbolic infeasibility. Figure 1 summarizes the procedure, Appendix A provides the complete HMCTS-SMT Algorithm.

5.1 Model, State, and Action Generation

Given $x = (\mathcal{D}, q, E)$, the global agent produces $\mathcal{M}_0 = (\mathcal{V}, \mathcal{Y}, \mathcal{C}_{\text{base}}, f, \eta)$ and $\mathcal{B} = (b_1, \ldots, b_L)$ where $\mathcal{M}_0$ is the symbolic model and $\mathcal{B}$ is an ordered subgoal blueprint. A search node is $s_t = (e_t, \mathcal{C}_t, j_t, \mathcal{H}_t, \mu_t)$ where e_t is the environment-facing state, $\mathcal{C}_t$ contains branch-specific updates, j_t identifies the active subgoal, $\mathcal{H}_t$ is the action history, and μ_t is an optional satisfying assignment. Its symbolic theory is

$$\Phi(s_t) = \left(\bigwedge_{c\in\mathcal{C}_{\text{base}}} c \right) \wedge \left(\bigwedge_{\delta\in\mathcal{C}_t} \delta \right). \tag{7}$$

Conditioned on b_{j_t}, $\Phi(s_t)$, $\mathcal{H}_t$, e_t, and μ_t, the local agent proposes at most k candidates with $a_t^i = (d_t^i, \Delta_t^i, p_t^i, m_t^i)$ containing a description, symbolic update, neural prior, and optional grounded metadata. The prior estimates relevance, not feasibility.

5.2 PUCT Search with SMT-Gated Transitions

HMCTS-SMT selects among unrejected actions using

$$a^* = \arg\max_{a\in\mathcal{A}_{\text{live}}(s)} \left[Q(s,a) + c_{\text{puct}} P(a \mid s, b_j, x) \frac{\sqrt{N(s)}}{1+N(s,a)} \right], \tag{8}$$

where Q is the empirical value, P is the local-agent prior, and N denotes visit counts. A selected action is admitted only if $G(s_t, a_t) = \mathbb{I}[\mathrm{SAT}(\Phi(s_t) \wedge \Delta(a_t))] = 1$. An admitted action creates $s_{t+1} = (e_{t+1},\, \mathcal{C}_t \cup \{\Delta(a_t)\},\, j_{t+1},\, \mathcal{H}_t \,\|\, a_t,\, \mu_{t+1})$. If the query is UNSAT, no child is created and

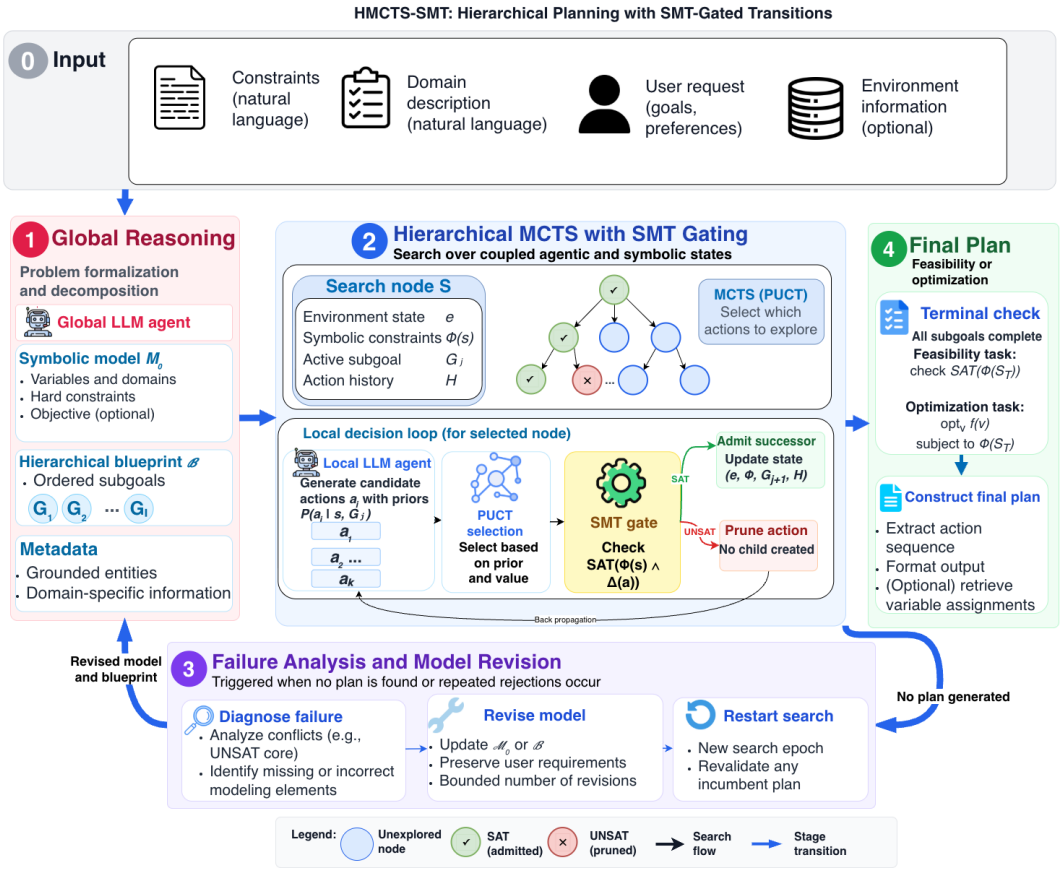

Figure 1: Overview of HMCTS-SMT. The global agent formalizes the planning instance into a symbolic model and subgoal blueprint. The local agent proposes candidate actions for each subgoal, each candidate is admitted to the search tree only if its symbolic update $\Delta(a)$ preserves satisfiability of the branch theory $\Phi(s)$. PUCT-guided MCTS allocates search effort among admissible trajectories, and the SMT solver performs terminal constrained optimization when all subgoals are complete.

the action is removed from $\mathcal{A}_{\text{live}}(s_t)$. Thus, G defines the feasible edge set, while P and Q rank only admissible trajectories.

Verification is lazy: candidates are generated and ranked together, but only a candidate selected for expansion is submitted to the solver. This preserves mandatory verification of committed transitions while avoiding checks for candidates never explored.

5.3 Terminal Solving and Revision

A state is terminal when all blueprint subgoals are complete. Feasibility tasks require $\text{SAT}(\Phi(s_T))$. For optimization tasks, the solver computes

$$f^\star(s_T) = \underset{\mathbf{v}\in\mathcal{V}}{\text{opt}_\eta}\, f(\mathbf{v}) \quad \text{subject to } \Phi(s_T), \tag{9}$$

where $\text{opt}_\eta \in \{\min, \max\}$. This value is exact for the terminal branch theory, but not necessarily across trajectories omitted by decomposition, candidate generation, or finite search.

Repeated rejection at a required subgoal may indicate an incorrect formalization, unsuitable decomposition, or missing environment information. The global agent may then revise $(\mathcal{M}_0, \mathcal{B})$. Revisions are bounded, share the search budget, and start a new search epoch. Any incumbent retained across epochs must be revalidated under the revised model. More details appear in Appendix C.

5.4 Representation-Relative Invariant

Proposition 1: Prefix satisfiability. Within an epoch with fixed model $\mathcal{M}_0^{(e)}$, suppose the root theory is satisfiable and the SMT solver is sound. Then every state reachable through HMCTS-SMT transitions has a satisfiable branch theory.

Proof sketch. The root is satisfiable by assumption. A child is created only if $\Phi(s_t) \wedge \Delta(a_t)$, which is its branch theory, is satisfiable. Induction over search depth proves the result. Appendix B provides the full proof.

This proposition records an invariant of the transition rule rather than the primary algorithmic contribution. It guarantees consistency only with the active encoding. It does not establish that $\Delta(a_t)$ faithfully captures the environment transition, that all natural-language requirements are encoded, or that finite search finds a globally optimal plan.

6 Domain Instantiations

We evaluate HMCTS-SMT in two complementary settings. OptiGuide tests integer-linear supply-chain optimization, while ChinaTravel tests open-ended itinerary planning under grounded constraints.

OptiGuide. We adapt Coffee ($n = 273$, seven infeasible) and Facility ($n = 165$) from solver-code generation to direct plan construction (Li et al., 2023). An independent verifier evaluates validity, optimality, and infeasibility detection. Details appear in Appendix G.1.

ChinaTravel. We use the Human ($n = 154$) and Medium ($n = 150$) splits (Shao et al., 2026). The evaluator measures structural, schema, commonsense, and hard-logic compliance, as well as joint all-pass performance. Definitions appear in Appendix G.2.

7 Experiments

We compare HMCTS-SMT with ReAct and Tree of Thoughts (ToT) on OptiGuide, and with ReAct and LLM-Modulo on ChinaTravel (Kambhampati et al., 2024). Experiments use Anthropic Haiku and Sonnet. Implementation details, uncertainty estimates, and costs appear in Appendix F.

7.1 Main Result

In Table 1 HMCTS-SMT substantially improves validity and optimality. With Sonnet, it raises Coffee optimality from 19.4% to 82.4% over ReAct and validity from 61.2% to 94.1%, achieving the highest validity rate in the experiment, the same pattern is observed for the Facility benchmark. With Haiku, HMCTS-SMT still achieves the best result, improving both validity and optimality in both Coffee and Facility. The complete results appear in Table 8 and Table 9 in Appendix H.1.

Table 1: OptiGuide results. Gap is the median relative objective gap over valid plans. Facility HMCTS-SMT Sonnet uses the UNSAT-feedback retry variant.

		Coffee ($n = 273$)			**Facility** ($n = 165$)		
Method	**Model**	**Valid**	**Optimal**	**Gap**	**Valid**	**Optimal**	**Gap**
ReAct	Haiku	90 (33.0%)	42 (15.4%)	3.31%	45 (27.3%)	2 (1.2%)	7.41%
	Sonnet	167 (61.2%)	53 (19.4%)	2.57%	73 (44.2%)	2 (1.2%)	5.56%
ToT	Haiku	33 (12.2%)	16 (5.9%)	3.31%	35 (21.2%)	3 (1.8%)	6.87%
	Sonnet	28 (10.3%)	6 (2.2%)	5.34%	48 (29.1%)	4 (2.4%)	5.32%
HMCTS-SMT	Haiku	**223 (81.7%)**	**176 (64.5%)**	**1.53%**	**99 (60.0%)**	**95 (57.6%)**	**0.44%**
	Sonnet	**257 (94.1%)**	**225 (82.4%)**	**0.42%**	**143 (86.7%)**	**111 (67.3%)**	**0.33%**

In Table 2 HMCTS-SMT with Sonnet achieves 20.1% all pass on Human and 4.0% on Medium, outperforming both baselines. The low Medium result reflects missing entity and transportation

attributes in the symbolic model, illustrating that solver guarantees remain relative to the encoding. Full results appear in Table 10 and Table 11 Appendix H.2.

Table 2: ChinaTravel results. All pass requires satisfaction of every evaluator dimension.

		Human ($n = 154$)			**Medium** ($n = 150$)		
Method	**Model**	**Valid**	**Common.**	**All pass**	**Valid**	**Common.**	**All pass**
ReAct	Haiku	100.0%	0.0%	0.0%	66.7%	0.0%	0.0%
	Sonnet	99.4%	0.0%	0.0%	100.0%	0.0%	0.0%
LLM-Modulo	Haiku	8.4%	1.9%	1.3%	4.7%	1.3%	0.0%
	Sonnet	27.9%	3.9%	1.3%	30.7%	6.7%	0.7%
HMCTS-SMT	Haiku	64.9%	26.0%	**16.2%**	68.7%	27.3%	**2.7%**
	Sonnet	79.2%	**49.4%**	**20.1%**	86.0%	**64.7%**	**4.0%**

On the Human split, HMCTS-SMT with SONNET achieves the highest all-pass rate at 20.1%, compared with 1.3% for LLM-Modulo and 0% for ReAct. Its main advantage is in commonsense satisfaction, where it reaches 49.4%, despite ReAct producing more complete plans. On the Medium split, HMCTS-SMT with SONNET achieves the highest all-pass rate at 4.0%, but not the highest hard-logic score. These results show that plan completion does not imply correctness and that SMT consistency cannot enforce external requirements absent from the symbolic encoding.

7.2 Ablation Study

To isolate the contribution of each component, we ablate the three core mechanisms of HMCTS-SMT, the SMT transition gate, the hierarchical decomposition, and the MCTS exploration policy, then compare each variant with the full system on the Coffee benchmark, Table 3,reports all results. Complete ablations appear in Table 14 in Appendix K.

Table 3: Coffee ablation with SONNET ($n = 273$).

Variant	**Valid**	**Optimal**	**Gap**	**Calls**
Full	**94.1%**	**82.4%**	**0.42%**	21.1
−Gate	85.0%	70.7%	0.95%	30.5
−Hierarchy	91.2%	26.0%	4.07%	3.5
Greedy	88.3%	25.3%	6.52%	6.4

As a result, all three components are essential and non-substitutable, the gate filters infeasible branches early, the hierarchy structures the search space so the solver can find optimal assignments, and MCTS explores alternatives instead of committing greedily.

7.3 Efficiency and Failure Modes

HMCTS-SMT is expensive. On ChinaTravel Human, SONNET averages 146.8 MCTS iterations, 304.0 solver calls, and 582 seconds per query. Its main failures arise from missing grounding and inaccurate symbolic updates. Detailed cost and failure analyses appear in Table 12 and Table 13 Appendices I and J.

8 Discussion and Scope

Our HMCTS-SMT model separates admissibility from utility. SMT satisfiability determines whether a successor exists, while LLM priors and MCTS values rank admissible trajectories. The results of the ablation study show that the verification improves feasibility, whereas hierarchy and search improve optimization. Assuming a satisfiable root and sound solver, every reachable node within a revision epoch has a satisfiable branch formal representation. This invariant does not ensure that the encoding captures the request, that $\Delta(a)$ faithfully represents the action, or that finite

search finds a globally optimal plan. Revised models start new epochs, and retained plans must be revalidated. Formal details appear in Appendix B.

9 Conclusion

We introduced *Reason–Explore–Verify* architecture and the HMCTS-SMT model, which extends verifier-guided search to hierarchical planning over agentic and branch-local symbolic states. SMT defines transition admissibility, while MCTS explores among admissible trajectories. HMCTS-SMT improves the validity and optimization quality across supply-chain and travel-planning tasks. Its main limitations are inference cost, incomplete grounding, and action-update fidelity. but still outperform other baselines mainly regarding optimality, Future work will address these issues and evaluate matched-compute baselines across additional domains.

Ethics Statement

The experiments use established benchmarks without human subjects or personal data. The system may inherit errors or biases from models, data, and symbolic encodings. Satisfiability does not establish safety, fairness, or correctness beyond the encoded constraints.

Reproducibility Statement

The appendices provide algorithms, prompts, implementation settings, benchmark transformations, evaluation procedures, and complete results.
Code and data availability: all baseline implementations, evaluation scripts, and result files are publicly available. The ChinaTravel benchmark is available at Shao et al. (2026), the OptiGuide benchmark is available at Li et al. (2023).
Randomness and repeated runs: All experiments use a single fixed seed (`seed = 0`). The LLM API is called with temperature $= 0.2$, introducing residual stochasticity in model outputs. Each query is run once; results represent single-run performance rather than averaged estimates. We report absolute counts in the main tables, the primary claims (HMCTS-SMT vs. ReAct/ToT) are large-margin improvements (e.g., 94.1% vs. 61.2% Coffee validity) that are unlikely to reverse under re-runs.

AI Use Statement

Claude (Anthropic) was used to assist with code implementation and Python debugging. Perplexity AI was used to identify potentially relevant literature and support documentation tasks. All retrieved sources were independently checked, and all AI-generated content was reviewed and validated by the authors. AI tools were not used to formulate the scientific claims, determine the experimental design, interpret the results, or draw the conclusions of this work.

References

Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (eds.), *Handbook of Satisfiability, Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 33, pp. 825–885. IOS Press, February 2021. URL http://theory.stanford.edu/~barrett/pubs/BSST21.pdf.

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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024. ISSN 2159-5399. doi: 10.1609/aaai.v38i16.29720. URL http://dx.doi.org/10.1609/aaai.v38i16.29720.

Dirk Beyer, Matthias Dangl, and Philipp Wendler. A Unifying View on SMT-Based Software Verification. *Journal of Automated Reasoning*, 60(3):299–335, March 2018. ISSN 1573-0670. doi: 10.1007/s10817-017-9432-6.

Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. The Barcelogic SMT Solver. In *Proceedings of the 20th International Conference on Computer Aided Verification*, CAV '08, pp. 294–298, Berlin, Heidelberg, July 2008. Springer-Verlag. ISBN 978-3-540-70543-7. doi: 10.1007/978-3-540-70545-1_27.

David Brandfonbrener, Simon Henniger, Sibi Raja, Tarun Prasad, Chloe Loughridge, Federico Cassano, Sabrina Ruixin Hu, Jianang Yang, William E. Byrd, Robert Zinkov, and Nada Amin. VerMCTS: Synthesizing Multi-Step Programs using a Verifier, a Large Language Model, and Tree Search, May 2024.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020.

Lucas Cordeiro, Bernd Fischer, and Joao Marques-Silva. SMT-Based Bounded Model Checking for Embedded ANSI-C Software, July 2009.

DeepSeek-AI. Deepseek-r1. *arXiv:2501.12948*, 2025.

Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3-4):189–208, December 1971. ISSN 00043702. doi: 10.1016/0004-3702(71)90010-5.

Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun, and Daniel Weld. PDDL - The Planning Domain Definition Language. August 1998.

Yung-Shen Hsia, Fang Yu, and Jie-Hong Roland Jiang. Neuro-Symbolic Compliance: Integrating LLMs and SMT Solvers for Automated Financial Legal Analysis, January 2026.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks, June 2024.

Levente Kocsis and Csaba Szepesvári. Bandit Based Monte-Carlo Planning. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Gerhard Weikum, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (eds.), *Machine Learning: ECML 2006*, volume 4212, pp. 282–293. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-45375-8 978-3-540-46056-5. doi: 10.1007/11871842_29.

Daniel Kroening and Ofer Strichman. *Decision Procedures: An Algorithmic Point of View*. Texts in Theoretical Computer Science. An EATCS Series. Springer Berlin Heidelberg, Berlin, Heidelberg, 2. 2nd ed. 2016 edition, 2016. ISBN 978-3-662-50496-3 978-3-662-50497-0. doi: 10.1007/978-3-662-50497-0.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. Large Language Models for Supply Chain Optimization, July 2023.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3806–3824, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.248.

Hernán Ponce-de-León, Florian Furbach, Keijo Heljanko, and Roland Meyer. Dartagnan: Bounded Model Checking for Weak Memory Models (Competition Contribution). In Armin Biere and David Parker (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, volume 12079, pp. 378–382. Springer International Publishing, Cham, 2020. ISBN 978-3-030-45236-0 978-3-030-45237-7. doi: 10.1007/978-3-030-45237-7_24.

Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61:203–230, 2011. URL https://api.semanticscholar.org/CorpusID:207081359.

Jie-Jing Shao, Bo-Wen Zhang, Xiao-Wen Yang, Baizhi Chen, Si-Yu Han, Jinghao Pang, Wen-Da Wei, Guohao Cai, Zhenhua Dong, Lan-Zhe Guo, and Yu-Feng Li. ChinaTravel: An Open-Ended Travel Planning Benchmark with Compositional Constraint Validation for Language Agents, April 2026.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

Shunyu Yao, Dian Yu, Jeffrey Zhao, et al. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, et al. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.

Yifan Zhang, Giridhar Ganapavarapu, Srideepika Jayaraman, Bhavna Agrawal, Dhaval Patel, and Achille Fokoue. SPIRAL: Symbolic LLM Planning via Grounded and Reflective Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(43):36527–36535, March 2026. ISSN 2374-3468. doi: 10.1609/aaai.v40i43.40975.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, et al. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning (ICML)*, 2024.

A Complete HMCTS-SMT Algorithm

Algorithm 1 HMCTS-SMT: Hierarchical Monte Carlo Tree Search with SMT Gating

Require: Instance $x = (\mathcal{D}, q, E)$; global agent Θ_G; local agent Θ_L; SMT oracle φ; budget N; patience P; restarts R; c_{puct}
Ensure: Plan π^* or FAILURE

$\mathcal{M}_0, \mathcal{B} \leftarrow \Theta_G$.TRANSLATE($x$); **assert** φ.CHECK($\mathcal{M}_0$) = SAT
$s_0 \leftarrow (e_0, \emptyset, \mathcal{B}, j{=}0, \emptyset, \mu_0)$; root $\leftarrow$ NODE(s_0); best $\leftarrow$ **null**; stall $\leftarrow 0$; epoch $\leftarrow 0$; rc $\leftarrow 0$
for $it = 1, \ldots, N$ **do**
  $node \leftarrow$ root
  while $node.expanded \wedge |node.children| \geq pw_c \cdot \sqrt{node.visits} \wedge node.untried = \emptyset$ **do**
    $live \leftarrow \{ch \in node.children : \neg ch.dead\}$
    if $live = \emptyset$ **then**
      $node.dead \leftarrow$ **true**; **break**
    end if
    $node \leftarrow \arg\max_{ch \in live} \mathrm{PUCT}(ch, c_{\text{puct}})$
  end while
  if $node.dead$ **then**
    continue
  end if
  $b_j \leftarrow node.state$.ACTIVESUBGOAL()
  if $\neg\, node.expanded$ **then**
    Attempt $A \leftarrow \Theta_L$.PROPOSE($node.state, b_j, x$)
      on SLOTINFEASIBLE(r): **if** rc $< R$, call Θ_G.REPAIR; restart epoch, rc += 1; **continue**
      on BRANCHDEADEND: $node.dead \leftarrow$ **true**; **continue**
    $node.untried \leftarrow$ SORTBYPRIOR(A); $node.expanded \leftarrow$ **true**
  end if
  if $node.untried \neq \emptyset$ **then**
    $a \leftarrow node.untried$.POPBEST(); $s' \leftarrow node.state$.COMMIT(a)
    if φ.CHECK($\Phi(s')$) = SAT **then**
      $child \leftarrow$ NODE($s', node, a.prior$); $node \leftarrow child$
    else
      $core \leftarrow \varphi$.LASTCORE(); $A_r \leftarrow \Theta_L$.RETRYWITHFEEDBACK($node.state, a, core, x$)
      if some $a' \in A_r$ satisfies φ **then**
        Create child from a'; $node \leftarrow child$
      else if $node.untried = \emptyset \wedge b_j.drop_tier \in \{\text{free}, \text{soft}\} \wedge \neg\, node.skip_tried$ **then**
        $node.untried \leftarrow$ [SKIPACTION($node.state, b_j$)]; $node.skip_tried \leftarrow$ **true**; **continue**
      end if
    end if
  else if $\neg\, node.state$.TERMINAL() **then**
    $node.dead \leftarrow$ **true**
    if $|node.state.committed| \leq 1 \wedge$ rc $< R$ **then**
      Call Θ_G.REPAIR($x, \emptyset, b_j$, "all rejected"); restart epoch
    end if
    continue
  end if
  if $node.state$.TERMINAL() **then**
    $f^* \leftarrow \varphi$.OPTIMIZE($\Phi(s_T), \mathcal{M}_0.obj$)
    $r \leftarrow$ TERMINALREWARD(f^*, best)
    if best = **null** **or** f^* improves best.$cost$ **then**
      best $\leftarrow$ (ASSEMBLE($node$), f^*); stall $\leftarrow 0$
    end if
  else
    $r \leftarrow$ INTERMEDIATEREWARD($node.state$)
  end if
  BACKPROPAGATE($node, r$); stall += 1; **if** stall $\geq P$: **break**
end for
if best $\neq$ **null** $\wedge$ epoch > 0 **then**
  if φ.CHECK(best.$state.\Phi(s_T)$) = SAT **then**
    return best.π
  else
    best $\leftarrow$ **null**
  end if
end if
return best.π **if** best $\neq$ **null** **else** FAILURE

Notes.

- **PUCT score**: $Q(s,a) + c \cdot P(a|s,b_j) \cdot \sqrt{N(s)}/(1+N(s,a))$, where Q is the empirical mean backed-up reward and $P(a)$ is the local agent's score field.
- **Progressive widening** (ChinaTravel only): a new child is spawned only if $|children| < pw_c \cdot \sqrt{visits}$ ($pw_c = 0.5$). OptiGuide uses the full untried list without this bound.
- **Reward functions** differ by domain. *OptiGuide*: UNSAT $\to 0$; SAT non-terminal: $r = 0.5 \cdot \text{coverage}$; SAT terminal: $r = 0.5 + \text{slack}$, where $\text{slack} = \max\big(0, (\text{incumbent} - \text{cost})/\text{incumbent}\big)$ measures improvement over the current best solution. *ChinaTravel*: UNSAT $\to 0$; SAT non-terminal: $r = \text{coverage}^2 + 0.1 \cdot \text{slack}$, where squaring coverage rewards depth commitment more steeply than linear reward, and slack is the SMT budget headroom; SAT terminal: same with $\text{coverage} = 1$.
- **Bounded revision**: when the restart budget is exhausted, further SLOTINFEASIBLE exceptions prune the local node rather than triggering a global restart.

B Formal Definitions and Proofs

B.1 Revision Epochs

A revision epoch is a maximal interval of search under a fixed symbolic model $\mathcal{M}_0^{(e)}$. A revision creates a new epoch and root.

B.2 Prefix Satisfiability

Proposition 1. *Let e be a revision epoch with satisfiable root formula $\Phi_0^{(e)}$. Assume that the SMT solver is sound and that a successor is created only when*

$$\text{SAT}(\Phi(s_t) \wedge \Delta(a_t)) .$$

Every state reachable during epoch e then has a satisfiable active formula.

Proof. The root is satisfiable by assumption. If s_t is reachable, a successor s_{t+1} is created only after the solver returns SAT for $\Phi(s_t) \wedge \Delta(a_t)$. By construction,

$$\Phi(s_{t+1}) = \Phi(s_t) \wedge \Delta(a_t),$$

so the successor formula is satisfiable. Induction over search depth proves the claim. □

B.3 Scope and Non-Guarantees

The proposition depends on solver soundness and applies only to the active symbolic encoding. It does not guarantee:

- equivalence between the symbolic model and the natural-language request;
- correctness or completeness of $\Delta(a)$;
- correspondence between symbolic and environment transitions;
- complete candidate generation or search;
- global optimality beyond the explored trajectories.

A stronger end-to-end guarantee would additionally require a correspondence condition

$$\text{Corr}(e_t, a_t, e_{t+1}, \Delta(a_t)),$$

asserting that the symbolic update faithfully represents the relevant environment transition. HMCTS-SMT does not currently prove this condition.

C Additional Architectural Discussion

Admissibility and Utility. HMCTS-SMT assigns distinct responsibilities to its components. The local-agent prior $P(s, a)$ measures semantic relevance, while $Q(s, a)$ records empirical search utility. The SMT gate defines the feasible outgoing edge set:

$$\mathcal{A}_{\text{legal}}(s) = \{a \in \mathcal{A}(s) : \text{SAT}(\Phi(s) \wedge \Delta(a))\}.$$

PUCT ranks actions within this set. A high prior, value estimate, or exploration bonus cannot compensate for symbolic inconsistency.

Search and Verification. Search alone increases candidate diversity but does not guarantee preservation of balance, capacity, integrality, or temporal constraints. Verification alone can reject inconsistent decisions but does not necessarily recover from early feasible yet suboptimal commitments. HMCTS-SMT combines these functions by using formal checks for reachability and MCTS for allocation of computation among reachable trajectories.

Lazy Verification. Candidates are generated and ranked at expansion time, but a solver query is issued only when a candidate is selected. The procedure is therefore lazy with respect to the generated candidate set and strict with respect to committed transitions. This reduces unnecessary solver calls without permitting an unchecked action to enter the tree.

Formalization Fidelity. The solver can enforce only constraints included in the active model. Moreover, its result depends on whether $\Delta(a)$ correctly captures the consequences of the proposed action. Missing entity attributes, incorrect grounding, or incomplete updates can produce an internally satisfiable plan that fails an external evaluator. Formalization coverage, provenance, grounding, and update validation are therefore first-class components of end-to-end reliability.

Model Revision. Repeated rejection at a mandatory subgoal may indicate an incorrect encoding, missing variable, unsuitable decomposition, or incomplete environment information. HMCTS-SMT then returns a structured report containing the failed subgoal, rejected updates, active constraints, relevant observations, and an unsatisfiable core when available. The global agent may revise the model or blueprint under a bounded restart budget. Explicit user requirements remain immutable, although their symbolic encodings may be repaired. Every revision starts a new epoch, and any incumbent retained across epochs must be revalidated.

D SMT Encoding Examples

D.1 OptiGuide Encoding

The Coffee supply-chain problem is a mixed-integer linear program (MILP). HMCTS-SMT encodes it natively using a `Problem` object whose constraints are compiled to Z3 QF_LIRA (quantifier-free linear integer and real arithmetic).

Variables and domains.
Suppliers: $\{s_1, s_2, s_3\}$; Roasteries: $\{r_1, r_2\}$; Cafés: $\{c_1, c_2, c_3\}$.

Table 4 presents examples of constraints variables and their domain.

Table 4: Variables and domains.

Variable	Type	Domain	Meaning
$x_{i,r}$	Int	$[0, 150]$	Beans shipped from supplier i to roastery r
$y^L_{r,c}$	Int	$[0, 1000]$	Light-roast beans from roastery r to café c
$y^D_{r,c}$	Int	$[0, 1000]$	Dark-roast beans from roastery r to café c

Objective (minimize).

$$\min \sum_{i,r} \text{ship}_{i,r}\, x_{i,r} + \sum_{r,c} (\text{roast}^L_r + \text{ship}^L_{r,c}) y^L_{r,c} + \sum_{r,c} (\text{roast}^D_r + \text{ship}^D_{r,c}) y^D_{r,c}$$

Constraints.

$$\sum_i x_{i,r} = \sum_c (y^L_{r,c} + y^D_{r,c}) \quad \forall r \qquad \text{(flow conservation)}$$

$$\sum_r x_{i,r} \le \text{cap}_i \qquad \text{(supplier capacity)}$$

$$\sum_r y^L_{r,c} \ge \text{light}_c \qquad \text{(café light demand)}$$

$$\sum_r y^D_{r,c} \ge \text{dark}_c \qquad \text{(café dark demand)}$$

Concrete data (default instance, native optimum $= 2470$).
In Table 5 all variables are declared as Z3 `Int`. Terminal solving uses an objective upper-bound binary search (the planner drives the bound; Z3 answers SAT/UNSAT). A what-if query modifies one or more data entries, regenerates the problem, and re-solves. The Facility problem encodes a capacitated facility-location ILP with binary open/close variables (`Int` $\in \{0, 1\}$), assignment variables, capacity, demand, and a cost-minimization objective.

Table 5: Example of constrains variables values assignation.

Parameter	Values
Supplier capacities	s_1: 150, s_2: 50, s_3: 100
Shipping $s \to r$ costs	(s_1, r_1): 5, (s_1, r_2): 4, (s_2, r_1): 6, (s_2, r_2): 3, (s_3, r_1): 2, (s_3, r_2): 7
Shipping $r \to c$ costs	(r_1, c_1): 5, (r_1, c_2): 3, (r_1, c_3): 6, (r_2, c_1): 4, (r_2, c_2): 5, (r_2, c_3): 2
Light demand per café	c_1: 20, c_2: 30, c_3: 40
Dark demand per café	c_1: 20, c_2: 20, c_3: 100

D.2 CHINATRAVEL ENCODING

HMCTS-SMT uses an incremental QF_LIRA theory over integer time variables (minutes 0–1,439) and real-valued cost variables.

Variables (per slot j).
In Table 6, we present an example of variables and constraints definition for China Travel

Table 6: Example of China Travel constraint variables.

Variable	Type	Meaning
$start_j$	Int	Slot start time (minutes since midnight)
end_j	Int	Slot end time
dur_j	Int	$end_j - start_j$ (auxiliary)
$cost_j$	Real	Monetary cost of activity j
run_j	Real	Cumulative cost through slot j

Base constraints $\mathcal{C}_{\text{base}}$. Seeded once per epoch from the global agent's output:

- Budget: $run_{\text{last}} \le B$
- Day bounds: $start_j \in [0, 1{,}439]$, $end_j \in [start_j, 1{,}439]$
- Meal time windows: breakfast $start_j \in [360, 539]$, lunch $[660, 839]$, dinner $[1{,}020, 1{,}199]$
- Minimum durations: $dur_j \ge 30$ (meals), ≥ 60 (attractions)
- Slot ordering: $start_{j+1} \ge end_j$ for consecutive slots
- Membership: cuisine, transport mode, or POI-type constraints encoded as integer label clauses

Per-action updates $\Delta(a)$. Appended when candidate a is selected for expansion; see Appendix D.3 for examples.

D.3 Action Updates

The following examples show how a concrete natural-language action a maps to a symbolic update $\Delta(a)$.

Example 1 — Intercity train (slot 0). *"G1234 high-speed train, Shanghai→Hangzhou, departs 08:00, arrives 09:15, cost ¥163 × 2 tickets"*

$$start_0 = 480, \quad end_0 = 555, \quad cost_0 = 326, \quad run_0 = 326$$

Example 2 — Lunch restaurant (slot 2). *"West Lake Impression, opens 11:00–21:00, ¥80/person × 2"*

$$start_2 \in [660, 839], \quad end_2 = start_2 + 30, \quad cost_2 = 160,$$
$$run_2 = run_1 + 160, \quad open \geq 660, \quad close \leq 1{,}260$$

Example 3 — Attraction (slot 3). *"West Lake scenic area, 2h, ¥0 × 2 tickets, metro 12 min ¥4 total"*

$$start_3 \geq end_2 + 12, \quad end_3 = start_3 + 120, \quad cost_3 = 4, \quad run_3 = run_2 + 4$$

Example 4 — Hotel (slot 4). *"Lakeside Inn, 1 room × 1 night, ¥380"*

$$start_4 \in [0, 1{,}439], \quad end_4 = 1{,}439, \quad cost_4 = 380, \quad run_4 = run_3 + 380$$

Example 5 — Droppable slot skip (slot 5, breakfast). *"SKIP (breakfast infeasible — arrival at 09:30 leaves no time in window)"*

$$start_5 = start_6, \quad end_5 = start_6, \quad cost_5 = 0, \quad run_5 = run_4$$

E Prompts

E.1 Global Agent Prompt

The global (translation) agent is invoked as a single-turn LLM completion and converts a natural-language travel request into a structured slot blueprint. The full prompt template is reproduced below; curly-brace fields are filled at runtime.

```
You are planning a trip. Read the request and output a JSON object describing
the plan at a HIGH LEVEL (do NOT choose specific trains/hotels yet).

The resulting plan is scored by a strict evaluator. Design the blueprint so it
can pass. Beyond HARD-LOGIC (total cost <= budget; correct days/people; ticket
counts match the party size), and enforces these commonsense requirements:

{commonsense}

So make the blueprint FEASIBLE -- order the slots by time of day, and write
each label with the timing/constraint intent so grounding is easy.
NEVER USE YOUR INTERNAL KNOWLEDGE TO FILL THE BLUEPRINT.
Examples of good labels:
  "intercity: morning train Shanghai->Hangzhou"
  "meal: breakfast right after arrival, at an OPEN place preferably near the
   station, don't forget to indicate how to get there and the travel time"
  "attraction: late-morning visit an OPEN attraction"
  "meal: lunch preferably near the attraction, don't forget to indicate how
   to get there and the travel time"
  "return: evening train back to Shanghai, after the last activity"

Return ONLY a JSON object with keys:
  "problem_class": one sentence naming the problem (who travels, between where,
                   how long),
  "budget": integer RMB budget (extract from the request),
  "people": integer number of travellers,
```

```
  "days": integer number of days,
  "slots": a list of steps, each with fields:
    "kind":     one of intercity, attraction, meal, accommodation, return
    "subtype":  specific variant -- ALWAYS fill:
                  meal: "breakfast", "lunch", or "dinner"
                  intercity: "outbound"
                  return: "return"
                  others: ""
    "label":    optional short NL description (system never relies on it)
    "day":      0-based day (0=first); REQUIRED for multi-day trips; 1-day -> all 0
    "cost_basis":
                  "people"  for MEALS
                  "tickets" for ATTRACTIONS and INTERCITY transport
                  "rooms"   for ACCOMMODATION
    "droppable" (MEAL/ATTRACTION only):
                  "free" (DEFAULT): slot added for a realistic day that the user
                         did NOT specifically name -- drop freely if infeasible.
                  "soft": user explicitly named this activity but it is NOT a
                         membership requirement -- keep it, drop only as last resort.
                  "no"  : keep -- use for the SINGLE representative of a category
                         the user mentions GENERICALLY; all others in same category
                         should be "free".
    "remark":   SHORT SOFT HINT for the worker, from user's explicit words ONLY.
                intercity/return: transport mode only if stated; append
                                  "({people} people)".
                meal: cuisine/type ONLY if named; append "({people} people)".
                attraction: type or name ONLY if stated.
                accommodation: room type/count ONLY if stated; append
                               "({people} people, N rooms)".
                INNER-CITY TRANSPORT: if user explicitly states how to move
                between activities (e.g. "by metro"), add
                "transport_mode:<mode>" to EVERY meal and attraction slot remark.
                Leave empty otherwise. NEVER invent preferences.
    "room_count" (accommodation ONLY): integer rooms; use 1 if unstated.
    "time_window_lo", "time_window_hi": integer clock minutes (0-1439) bounding
                the slot START time. Encode meal windows HERE, NOT in constraints:
                  breakfast: lo=360,  hi=539
                  lunch:     lo=660,  hi=839
                  dinner:    lo=1020, hi=1199
                Non-meal slots with no time constraint: set both to null.
    "membership_attr", "membership_value": hard entity requirement for THIS slot.
                Set only when user explicitly requires a specific entity:
                  cuisine: attr="cuisine" only on the specific meal(s) mentioned.
                  attraction type: attr="type" only on slots where user requested it.
                  room type: attr="numbed", value="2" on accommodation slot.
                  named POI: attr="name", value="West Lake" on that slot only.
                Empty strings if no hard requirement for this slot.
  "constraints": optional inter-slot and duration constraints. Each is
    {"slot": <index>, "var": "start"|"end"|"duration",
     "op": "<="|">="|"==", "value": <integer minutes>, "name": "<short id>"}
    Use ONLY for minimum durations on free-time slots:
      meal:       {"var": "duration", "op": ">=", "value": 30}
      attraction: {"var": "duration", "op": ">=", "value": 60}

Rules: first slot MUST be "intercity" (outbound); last MUST be "return".
Include breakfast/lunch/dinner meals as the day spans them.
For a 1-day trip include no "accommodation"; for D days include one per night.
Return ONLY the JSON, no prose.

Request:
{nl}
Known: start_city={start_city}, target_city={target_city},
       days={days}, people={people}
```

Regenerate variant (on SLOTINFEASIBLE restart): the same prompt is appended with a note explaining the failure at the failed label and asking for a revised slot list that avoids it, including timing constraints from the arrival time.

Repair variant (tail-repair, preserving committed prefix): an `=== REPAIR REQUEST ===` section is appended listing every committed slot, the failure index and reason, and the instruction to keep the prefix unchanged while revising the tail.

E.2 LOCAL AGENT PROMPT

The local agent runs a bounded mini-ReAct loop (at most `max_tool_calls` turns per node expansion). The prompt is assembled from three components injected into the loop at runtime.

Tool-use preamble (`tools` block):

```
You may CALL these WorldEnv query tools to surface concrete options.
TIP: for a POI/restaurant slot, first call the *_keys / *_types /
*_cuisine tool to learn the valid column keys and values, THEN _select
with a lambda that matches them -- guessing keys/values returns 0 rows.
IMPORTANT: attractions_id_is_open / restaurants_id_is_open only verify
one KNOWN id -- do NOT call them with guessed IDs (0, 1, 2...) to discover
options. Use attractions_select or restaurants_select first to get a list
with real IDs, then call id_is_open if you need to check hours for a
specific one.

[WorldEnv tool signatures inserted here at runtime]
```

Blueprint-following instruction (`implicit` block):

```
--- FOLLOW THE BLUEPRINT ---
Ground the ACTIVE step above by choosing a concrete option that fits its
intent (e.g. an early-opening place for a breakfast step). The symbolic
solver already enforces budget, chronological ordering, and POI open-hours
on the times it computes -- you only need to PICK sensible options; you do
not add any constraints yourself.
```

Response schema (`schema` block):

```
--- HOW TO RESPOND (reply with ONLY one JSON object) ---
To call a tool:
  {"tool": "restaurants_nearby(city='Hangzhou', point='West Lake', topk=5, dist=3)"}
To finish (rank options you have gathered, best first, at most {k}):
  {"finish": true, "candidates": [{"index": i, "score": s}]}
  (score s in [0,1]; "index" refers to the numbered option list above.)
PREFER to finish with at least one candidate whenever you have gathered any
usable option -- pick the best available even if imperfect. Only use the
stop option below as a LAST RESORT when no gathered option could possibly work:
  {"stop": true, "reason": "<why no option can work>"}
```

Retry-with-feedback prompt: on first UNSAT rejection, the planner re-prompts the local agent with the UNSAT core explanation (e.g., `budget_total`, `open_N`, `tw_breakfast`), asking for a revised candidate that avoids the violated constraints.

E.3 MODEL-REVISION PROMPT

When all candidates for a required subgoal are gate-rejected and the restart budget allows, the global agent's `repair()` method is invoked. The revision prompt appends the following suffix verbatim to the standard translation prompt:

```
=== REPAIR REQUEST ===
The planner successfully committed these slots:
  slot 0: "intercity: morning train" -> G1234 (08:00-09:15, cost 326)
  slot 1: "meal: breakfast" -> (dropped)
  ...

Slot {failed_idx} ('{failed_label}') then failed: {reason}.
The committed prefix ends at {last_end_time}.

Produce a REVISED blueprint where:
- Slots 0..{failed_idx-1} keep the SAME label, kind, subtype, day as before
  (prefix is fixed).
- Slots {failed_idx}..end are REVISED to be feasible given the committed prefix.
  You may reorder, replace, or remove tail slots to avoid the conflict.
  Be realistic about timing: the prefix ends at {last_end_time}.
  IMPORTANT -- if the failed slot was NOT specifically cited in the user's query,
  you may simply OMIT it. Only keep a slot if the user explicitly requested it or
  if removing it would leave the itinerary structurally incomplete.
```

Explicit user requirements remain immutable, the global agent may only change free/soft slots and their symbolic encodings.

E.4 OPTIGUIDE TRANSLATION PROMPT

The translation agent converts a natural-language supply-chain problem and a what-if query into a Z3-compatible JSON model. The full prompt template is reproduced below; `{context}` is the problem description and `{query}` is the what-if question.

```
You translate a natural-language optimization problem into
a formal integer-linear model for the Z3 SMT solver, expressed as JSON.

HOW TO TRANSLATE -- follow these steps:
1. VARIABLES: one integer variable per atomic decision quantity, each with finite
   bounds [lo, hi]. Name them systematically (e.g. flow from A to B -> "A_to_B").
2. CONSTRAINTS: turn EVERY requirement in the text into a linear constraint.
   - a capacity/limit -> "<=";  a demand/requirement -> ">=";  a balance/equality -> "==".
   - Each constraint has a linear "lhs" ({var: coeff}), an "op", an integer "rhs",
     a short "name", and a "comment" quoting the phrase it encodes.
3. APPLY THE WHAT-IF QUERY COMPLETELY. This is the step most often done wrong.
   A change like "+29% demand at X" affects EVERY constraint tied to X -- if X has
   several demand types (e.g. light AND dark), scale ALL of them, not just one.
   Round scaled integer bounds to keep the requirement satisfied (a ">=" demand
   rounds UP: 20*1.29=25.8 -> 26; 30*1.29=38.7 -> 39). Missing or half-applied
   edits produce a WRONG model that looks cheaper than the true answer.
   - FORCED-INCLUSION queries ("Why not select / use / open X?", "Why don't we
     ship on edge X?") do NOT mean remove X -- they mean the OPPOSITE: FORCE X
     into the solution and re-optimize to expose its cost. Add a ">=" (or "==")
     LOWER bound so the quantity is used: an edge -> flow >= 1; a plant/site ->
     open == 1. Do NOT set it to 0 and do NOT name the model "no_X"/"exclude_X".
     Forcing a normally-unused option ON makes the optimum go UP, not down; a
     model that looks CHEAPER than the base means you inverted the query.
     Only phrasings that explicitly forbid/close/drop X map to "== 0".
4. OBJECTIVE: the quantity to minimize/maximize, as {var: coeff} + constant.

WORKED MICRO-EXAMPLE A (2 cafes, light+dark demand, "+50% demand at cafeB"):
  cafeB light 10 -> ceil(10*1.5)=15 ; cafeB dark 4 -> ceil(4*1.5)=6  (BOTH scaled)
  {"lhs": {"toB_light": 1}, "op": ">=", "rhs": 15, "name": "cafeB_light_demand",
    "comment": "cafeB light demand 10 increased 50% -> 15"},
  {"lhs": {"toB_dark": 1}, "op": ">=", "rhs": 6, "name": "cafeB_dark_demand",
    "comment": "cafeB dark demand 4 increased 50% -> 6"}

WORKED MICRO-EXAMPLE B (forced inclusion, "Why not select the edge from plant 3
to warehouse 3?"): FORCE that edge on -- ship at least 1 unit, NOT 0.
  {"lhs": {"ship_plant3_warehouse3": 1}, "op": ">=", "rhs": 1,
    "name": "force_edge_p3_w3",
    "comment": "'why not select edge plant3->warehouse3' forces the route ON (>=1)"}
  (WRONG would be bounding ship_plant3_warehouse3 to hi=0 or naming the model
   "no_plant3_to_warehouse3" -- that excludes the edge, the opposite of the query.)

Context:
{context}
Query (what-if to apply -- apply it to ALL affected constraints):
{query}

Return ONLY a JSON model object with keys in this EXACT order: name, sense
("min"|"max"), variables [{"name","lo","hi"}], objective
{"coeffs":{var:coeff},"constant"}, constraints
[{"lhs":{var:coeff},"op":"<="|">="|"==","rhs","name","comment"}].
Output objective BEFORE constraints -- the objective is shorter and must not be
truncated. Every constraint MUST carry a "comment" tying it to the problem text.
This is the JSON model.
```

E.5 OPTIGUIDE BLUEPRINT-DECOMPOSITION PROMPT

After translation, the planner calls the blueprint agent to decompose the model into an ordered sequence of decision subgoals. Each subgoal is a group of variable-assignment decisions that progressively constrain the model; the solver is invoked automatically after the last subgoal and is never listed as a step. The prompt template is reproduced below.

```
Decompose this optimization problem into an ordered list
of DECISION subgoals -- each subgoal is a group of variable-assignment decisions
that narrow the model toward a feasible solution.

IMPORTANT: the problem AND the what-if query have ALREADY been translated into the
formal model -- the variables, constraints, objective, and the query's effect (e.g.
an increased demand) are all encoded already. Your subgoals therefore only ASSIGN
the decision variables; they must NEVER re-apply, update, or restate the what-if or
any data value, because that work is already done.
```

```
The planner AUTOMATICALLY tries to solve/optimize the model once the LAST subgoal
is done -- solving is NOT a step you plan. Every subgoal, INCLUDING THE LAST, must
be a group of decisions that adds or tightens constraints on the variables.

Context:
{context}
Query:
{query}

Rules for the subgoals:
- Each must be an ACTIONABLE decision group that ADDS constraints to the model
  (e.g. "set supplier->roastery shipping quantities", "assign roastery->cafe
  light/dark deliveries meeting cafe demand").
- FORBIDDEN as subgoals (they add no variable-assignment constraints, so the search
  stalls on them): anything phrased as "solve", "optimize", "minimize cost",
  "verify", "check/ensure flow conservation", "compare to baseline", "analyze",
  "review", "report", OR anything that "applies/updates/sets the demand", "adjusts
  the what-if", or "increases/scales a value". Demand, flow-conservation, and the
  what-if are ALREADY in the model -- do NOT create a subgoal to apply or verify
  them. Solving/optimizing is done automatically at the end, never as a subgoal.
- The FINAL subgoal must still commit real decisions (e.g. the last routing
  assignments), NOT a wrap-up/verify/solve step.
- Subdivide the task in the most EFFECTIVE way: partition ALL the decision
  variables into coherent groups (e.g. by stage, facility, or resource) so that
  completing the subgoals in order assigns every variable. Use as many or as few
  subgoals as the problem needs -- enough that each variable is covered by some
  subgoal, ordered so earlier groups constrain later ones. Prefer coherent,
  decision-dense groups over both a single giant step and many trivial ones.

Return ONLY a JSON array of short string labels (the ordered groups of decisions).
```

E.6 OPTIGUIDE LOCAL-AGENT PROPOSAL PROMPT

At each MCTS node expansion the local agent is asked to propose up to k candidate constraints for the active subgoal. The prompt supplies the full symbolic state (variables with bounds, existing constraints, current cost) and the blueprint progress so the agent can make a completeness check before proposing. The template is reproduced below.

```
Propose up to {k} candidate decisions for the current subgoal.
Problem: {problem}
What-if query to solve: {query}

--- CURRENT SYMBOLIC STATE (the SMT model you are editing) ---
Variables (use ONLY these exact names in any constraint; bounds shown):
{variables}
Current feasible total cost ({sense}): {current_cost}
Model constraints so far (each with the requirement it encodes):
{constraints}
Objective ({sense}):
{objective}

--- AGENT STATE (blueprint progress) ---
Completed subgoals: {done}
Active subgoal: {subgoal}
Remaining subgoals: {remaining}
Constraints you have committed so far: {committed}

CRITICAL -- completeness. The model above is the WHOLE truth the solver sees. If a
real requirement of the problem is NOT encoded as a constraint, the solver will
exploit its absence and return a WRONG (too-cheap / infeasible-in-reality) answer,
which makes your proposed action FALSE. Before proposing, check that every
requirement implied by the active subgoal is captured; if one is missing, propose
the constraint that adds it. Never omit a constraint to make a plan look better.

CRITICAL -- every action MUST include a formal constraint. A natural-language
description alone is not accepted and will be discarded. The constraint's `lhs`
keys MUST be variable names from the list above verbatim -- do not invent or
reformat names. Use "==" to pin a variable to an exact value, ">=" or "<=" to
bound it. Prefer "==" for routing decisions so the plan is fully determined.
Return ONLY JSON. Either
{"actions":[{"nl":"<decision>","score":<0..1>,
"constraint":{"lhs":{var:coeff},"op":"<="|">="|"==","rhs":<number>,"name":"..."}}]}
or, if the subgoal is impossible,
{"infeasible":true,"reason":"...","suggestion":"<optional correction hint>"}.
```

E.7 BASELINE PROMPTS

ReAct (OptiGuide).

```
You are solving a supply-chain optimization what-if question.
This is an Integer Linear Program (ILP): every decision variable must take an
INTEGER value, and all constraints (capacities, demands, flow conservation) must
hold exactly. Do not use fractional quantities; the minimized total cost is the
objective at an integer-feasible assignment.
Reason step by step (Thought:), then:
1. Give a line 'Answer: <number>' with the minimized total cost as a plain integer.
2. Immediately after, fill in the JSON template below with the EXACT integer values
   you used -- no nulls left, no extra keys.

--- PROBLEM (natural language) ---
{problem}

--- WHAT-IF QUESTION ---
{question}

--- ASSIGNMENT TEMPLATE (fill after Answer line) ---
{template}
```

Tree of Thoughts (OptiGuide). ToT uses three prompts. The *generate* prompt proposes k distinct next reasoning steps (any step may terminate by including an `Answer:` line). The *evaluate* prompt scores a partial reasoning trace 0–10. The *finalize* prompt commits the full plan from the best reasoning trace. The search runs BFS with $n_{\text{sel}} = 3$ retained thoughts per level, up to $n_{\text{steps}} = 5$ levels.

Generate prompt:

```
You are solving an optimization what-if problem by exploring a tree of reasoning steps.

PROBLEM:
{problem}

WHAT-IF QUESTION:
{query}

REASONING SO FAR:
{thoughts}

Propose {k} DISTINCT next reasoning steps that advance toward the OPTIMAL feasible plan.
Each step is a short paragraph exploring one concrete strategy. A step MAY complete the
solution: if you are ready to commit, make the step state the full plan (all decisions)
and END it with a line exactly of the form
  Answer: <total cost as a number>
or, if you conclude no feasible plan exists,
  Answer: infeasible

Return ONLY JSON:
{"thoughts": ["<step 1>", "<step 2>", ...]}
```

Evaluate prompt:

```
Rate how promising this partial reasoning is for reaching the OPTIMAL feasible plan.

PROBLEM:
{problem}

WHAT-IF QUESTION:
{query}

REASONING TO EVALUATE:
{thoughts}

Score it as an integer 0-10 where 10 = clearly on track to the optimal feasible plan and
0 = wrong, infeasible, or a dead end. Return ONLY JSON: {"value": <0-10>, "why": "<brief>"}
```

Finalize prompt:

```
Based on your reasoning below, commit to the FINAL complete plan.

PROBLEM:
{problem}
```

```
WHAT-IF QUESTION:
{query}

YOUR REASONING:
{thoughts}

State the complete assignment (every decision) and the total cost. End with a line
exactly of the form:
  Answer: <total cost as a number>
or, if no feasible plan exists:
  Answer: infeasible
```

LLM-Modulo (ChinaTravel). A direct generation agent (ReAct-style mini-loop) runs up to `max_iters` tool calls to generate a plan. The external hard-logic checker serves as the critic; if the plan fails, the agent's trace and the checker's feedback are concatenated and the agent retries once. No internal symbolic model is maintained.

F Implementation Details

HMCTS-SMT is implemented in Python 3.11. The LLM interface wraps an OpenAI-compatible REST endpoint, all calls are synchronous and single-threaded, in Table 7, we present the implementation settings.

Table 7: Implementation and inference settings.

Parameter	Value
Haiku model identifier	`anthropic-claude-4.5-haiku`
Sonnet model identifier	`anthropic-claude-4.5-sonnet`
Model access date	September 2025
Temperature	0.2
Maximum output tokens	16,384
Candidate width k	3 (OptiGuide) / 2 (ChinaTravel)
c_{puct}	1.4 (OptiGuide) / 0.15 (ChinaTravel)
Maximum MCTS iterations	1,000 (OptiGuide) / 300 (ChinaTravel)
Maximum restarts	3
Solver and version	Z3 4.16.0 (Python bindings)
Solver timeout	10,000 ms per call
UNKNOWN policy	Treat as UNSAT (reject candidate)
Reward (OptiGuide)	non-terminal: $r = 0.5 \cdot \text{cov}$
Reward (ChinaTravel)	non-terminal: $r = \text{cov}^2 + 0.1 \cdot \text{slack}$; terminal: same with $\text{cov} = 1$
Random seeds	0 (single run per query)

OptiGuide. A single `Z3Oracle` instance is shared across all 1,000 MCTS iterations per query. The oracle uses Z3's named-assertion unsat-core extraction. Infeasible instances (where the what-if modification renders the LP infeasible) are detected when $\varphi.\textsc{Check}(\Phi_{\text{root}})$ returns UNSAT before search begins; the system reports "correct infeasible" and skips the search.

ChinaTravel. A fresh `Z3Oracle` is created per query. The `TravelState` object holds a reference to the query's `Problem` (base constraints plus accumulated branch updates). `state.commit(action)` clones the problem and appends $\Delta(a)$; `state.gate_reason(oracle)` calls `oracle.check` and returns the SAT flag and unsat core. Monetary variables are declared as Z3 `Real` to handle fractional prices; time and count variables remain `Int`.

Baselines. ReAct sends a single LLM call per query. ToT runs a BFS beam with width 5 and depth 5. Both baselines share the same model identifiers and temperature as HMCTS-SMT.

G Benchmark Details

G.1 OptiGuide Adaptation

OptiGuide (Li et al., 2023) was originally designed for solver-code generation: given a Gurobi model and a what-if question, an LLM generates Python code that modifies and re-solves the model. We adapt it to *direct plan construction*.

Problem extraction. Each benchmark instance (Coffee: $n = 273$; Facility: $n = 165$) is translated into a native Python `Problem` object with integer variables and linear constraints (no Gurobi dependency).

Reference objectives. For feasible instances, the native Z3 optimizer computes the ground-truth optimal value. For infeasible instances (Coffee: 7; Facility: 0), the ground truth is "infeasible."

Instance selection. All instances from the OptiGuide benchmark are used, no filtering is applied other than removing duplicate questions.

Task transformation. Instead of generating solver code, the model must output a complete variable assignment (a JSON object matching the assignment template) and a claimed cost (`Answer: <n>`). This eliminates the code-execution step.

Evaluation. An independent pure-Python verifier checks: (a) all variables are integers within their domains; (b) all constraints hold; (c) the claimed cost equals the Z3-computed optimum within tolerance 10^{-4}. A plan is *valid* if (a) and (b) hold; it is *optimal* if additionally (c) holds. A plan is *correct infeasible* if the verifier confirms infeasibility and the model declares it. The objective comparison uses absolute tolerance 10^{-4} (the objective is always an integer).

G.2 ChinaTravel Evaluation

ChinaTravel uses the official evaluation pipeline from Shao et al. (2026) with the following dimensions.

Structural validity (Schema pass). The output JSON matches the required itinerary schema: correct top-level keys, valid activity-type fields, resolvable POI names, and complete time fields.

Commonsense compliance (micro-averaged). Each itinerary is checked against domain commonsense rules (e.g., return trip departs after the last local activity, accommodation booked for every overnight stay, opening hours respected). The micro score is the fraction of individual rule checks that pass, the macro score (*Commonsense pass*) requires 100% micro satisfaction.

Hard-logic compliance (micro-averaged). Query-specific constraints are extracted programmatically from each query's annotation: budget limit, party size, ticket/room counts, transport mode, cuisine type, attraction type, and named-POI requirements. The macro score requires all constraints to pass.

All pass. A plan receives "all pass" only if schema, commonsense, and hard-logic all pass simultaneously.

Evaluator version and preprocessing. ChinaTravel benchmark from Shao et al. (2026). English runs use the English-column DB; the evaluator's `normalize_concept_value` maps English DB values to Chinese canonical equivalents before comparing against hard-logic predicates. No query filtering is applied.

H Complete Experimental Results

H.1 Full OptiGuide Results

Table 8 reports the complete Coffee results. HMCTS-SMT substantially improves validity and optimality over all baselines, incresing valid plans with Sonnet from (61.3%) validity rate in ReAct baseline to (94.1%) validity rate with HMCTS-SMT.

Table 8: Complete Coffee results ($n = 273$, including seven infeasible instances). Gap is measured over valid plans.

Method	Model	Optimal	Subopt.	Valid	Invalid	Infeas.	Gap	Subopt. gap	Calls	Latency
HMCTS-SMT	Sonnet	225 (82.4%)	32 (11.7%)	257 (94.1%)	10 (3.7%)	6/7	0.42%	3.38%	21.1	194 s
ReAct	Sonnet	53 (19.4%)	114 (41.8%)	167 (61.2%)	106 (38.8%)	0/7	2.57%	3.77%	1.0	19 s
ToT	Sonnet	6 (2.2%)	22 (8.1%)	28 (10.3%)	238 (87.2%)	7/7	5.34%	6.80%	21.1	199 s
HMCTS-SMT	Haiku	176 (64.5%)	47 (17.2%)	223 (81.7%)	43 (15.8%)	7/7	1.53%	6.48%	47.2	189 s
ReAct	Haiku	42 (15.4%)	48 (17.6%)	90 (33.0%)	183 (67.0%)	0/7	3.05%	5.71%	1.0	11 s
ToT	Haiku	16 (5.9%)	17 (6.3%)	33 (12.2%)	232 (85.6%)	6/7	3.31%	6.42%	62.9	236 s

In Table 9 HMCTS-SMT improves the Facility results over 165 queries. HMCTS-SMT with Sonnet produces 143 valid plans (86.7%) and 111 optimal solutions (67.3%). ReAct achieves valid-plan rates of 27.3% with Haiku and 44.2% with Sonnet, with only two optimal solutions under each configuration.

Table 9: Complete Facility results ($n = 165$; no infeasible instances). Gap is the median relative objective gap over valid plans. HMCTS-SMT uses the UNSAT-feedback retry variant. All runs have $n = 165$.

Method	Model	n	Optimal	Subopt.	Valid	Invalid	Infeas.	Gap	Subopt. gap	Calls	Latency
HMCTS-SMT	Sonnet	165	111 (67.3%)	32 (19.4%)	143 (86.7%)	22 (13.3%)	0 (0.0%)	0.33%	1.49%	12.7	115.4 s
ReAct	Sonnet	165	2 (1.2%)	71 (43.0%)	73 (44.2%)	92 (55.8%)	0 (0.0%)	5.56%	5.71%	1.0	20.9 s
ToT	Sonnet	165	4 (2.4%)	44 (26.7%)	48 (29.1%)	117 (70.9%)	0 (0.0%)	5.32%	5.80%	30.1	243.8 s
HMCTS-SMT	Haiku	165	95 (57.6%)	4 (2.4%)	99 (60.0%)	66 (40.0%)	0 (0.0%)	0.44%	5.60%	18.7	77.0 s
ReAct	Haiku	165	2 (1.2%)	43 (26.1%)	45 (27.3%)	120 (72.7%)	0 (0.0%)	7.41%	7.76%	1.0	10.6 s
ToT	Haiku	165	3 (1.8%)	32 (19.4%)	35 (21.2%)	130 (78.8%)	0 (0.0%)	6.87%	7.52%	60.2	223.1 s

H.2 Full ChinaTravel Results

Table 10 reports results on all 154 Human-split queries. HMCTS-SMT with Sonnet returns 122 valid complete plans (79.2%), compared with 43 (27.9%) for LLM-Modulo. ReAct returns a complete plan for nearly every query, but none of its plans satisfy all evaluation dimensions. HMCTS-SMT obtains the strongest all-pass results, reaching 20.1% with Sonnet and 16.2% with Haiku. Both ReAct configurations achieve an all-pass rate of 0%, while LLM-Modulo reaches 1.3%. The main improvement comes from commonsense satisfaction: HMCTS-SMT reaches 49.4% with Sonnet, compared with 3.9% for LLM-Modulo and 0% for ReAct.

Table 10: Complete results on ChinaTravel Human ($n = 154$).

Method	Model	Valid	Schema	Commonsense	Hard logic	All pass
ReAct	Haiku	154 (100.0%)	111 (72.1%)	0 (0.0%)	27 (17.5%)	0 (0.0%)
ReAct	Sonnet	153 (99.4%)	109 (70.8%)	0 (0.0%)	38 (24.7%)	0 (0.0%)
LLM-Modulo	Haiku	13 (8.4%)	36 (23.4%)	3 (1.9%)	31 (20.1%)	2 (1.3%)
LLM-Modulo	Sonnet	43 (27.9%)	31 (20.1%)	6 (3.9%)	47 (30.5%)	2 (1.3%)
HMCTS-SMT	Haiku	100 (64.9%)	96 (62.3%)	40 (26.0%)	40 (26.0%)	25 (16.2%)
HMCTS-SMT	Sonnet	122 (79.2%)	122 (79.2%)	76 (49.4%)	47 (30.5%)	31 (20.1%)

Table 11 reports results on the 150 Medium-split queries. HMCTS-SMT with Sonnet produces 129 valid plans (86.0%) and six all-pass plans (4.0%). LLM-Modulo with Sonnet produces 46

valid plans (30.7%) and one all-pass plan (0.7%), while both ReAct configurations obtain no all-pass plans.

Table 11: Complete results on ChinaTravel Medium ($n = 150$). Parenthesized values are micro-averaged constraint scores.

Method	Model	Valid	Schema	Commonsense	Hard logic	All pass
ReAct	Haiku	100 (66.7%)	54.0%	0.0% (16.5)	8.7% (79.4)	0 (0.0%)
ReAct	Sonnet	150 (100.0%)	84.0%	0.0% (13.7)	21.3% (74.3)	0 (0.0%)
LLM-Modulo	Haiku	7 (4.7%)	16.7%	1.3% (64.5)	1.3% (36.3)	0 (0.0%)
LLM-Modulo	Sonnet	46 (30.7%)	21.3%	6.7% (93.0)	13.3% (80.8)	1 (0.7%)
HMCTS-SMT	Haiku	103 (68.7%)	62.0%	27.3% (89.5)	4.7% (74.6)	4 (2.7%)
HMCTS-SMT	Sonnet	129 (86.0%)	86.0%	64.7% (97.6)	6.0% (75.3)	6 (4.0%)

I Runtime and Search Statistics

Table 12 provides per-method runtime and search statistics for HMCTS-SMT, the values are mean/median unless otherwise indicated. For the Sonnet–ChinaTravel Human setting, the mean is 146.8 MCTS iterations, 304.0 solver calls, and 582 seconds per query. ReAct completes in a single LLM call ($\sim$2–5 s). ToT runs 5 levels $\times$ 5 thoughts $\times$ SONNET $\approx$ 25 LLM calls ($\sim$30–90 s). LLM-Modulo runs up to 5 tool-use turns per attempt $\times$ 2 attempts ($\sim$40–120 s). A matched-budget comparison would give ReAct and ToT the same total number of LLM calls as HMCTS-SMT (300 iterations $\times$ $\sim$1.5 calls/iter $\approx$ 450 calls). Under this budget, ReAct simply repeats its single-shot generation up to 450 times and returns the best result.

Table 12: HMCTS-SMT statistics on ChinaTravel Human.

Metric	Sonnet	Haiku
MCTS iterations	146.8 / 104	186.3 / 144
Solver calls	304.0 / 225	306.1 / 242
SAT accepted	97.1 / 56	91.7 / 60
UNSAT rejected	12.5 / 6	12.6 / 8
Restarts, mean	0.6	0.7
Latency (s)	582 / 320	826 / 214
Input tokens, mean	617K	112K
Output tokens, mean	26K	43K

With SONNET, 78 queries complete without revision, 56 require one restart, 16 require two, and four require three. The corresponding HAIKU counts are 72, 66, seven, and nine.

J Failure Analysis

Table 13 reports the most frequent hard-logic failures for HMCTS-SMT on the ChinaTravel Human split. Most failures involve semantic entity attributes rather than arithmetic feasibility. Attraction-type requirements account for 46 failures, followed by restaurant-type requirements with 21 failures and named-attraction requirements with 17 failures.

Table 13: Frequent hard-logic failures for HMCTS-SMT on ChinaTravel Human.

Constraint category	Failures	Rate
Attraction type	46	38%
Restaurant type	21	17%
Named attraction	17	14%
Intercity transportation	13	11%
Inner-city transportation	7	6%
Room count	3	2%
Room type	3	2%
Accommodation type	2	2%
Arrival time / number of people	2	1%

These failures primarily concern semantic entity attributes rather than arithmetic inconsistency. A plan may therefore remain SAT under the internal model while violating an omitted, mistranslated, or unsupported external requirement.

Case J.1 — Missing entity attribute (ChinaTravel Medium, SAT but external fail). *Request*: "3-day trip, Beijing to Chengdu, 2 people, budget 10,000 CNY. Sichuan cuisine throughout."

Constraint: membership — cuisine = "Sichuan cuisine" on all meal slots.

Selected action (slot 2, lunch): The local agent selects "Mao's Restaurant," present in the DB.

Symbolic update $\Delta(a)$: $start_2 = 720$, $end_2 = 780$, $cost_2 = 160$. The membership gate confirmed at plan-construction time that at least one Sichuan restaurant exists; the constraint is marked SAT.

Solver result: SAT.

External evaluator: FAIL — hard-logic checker identifies that "Mao's Restaurant" has cuisine = "Beijing cuisine" in the DB, not Sichuan.

Root cause: The membership gate checks that a matching option *exists* in the DB; it does not pin the committed entity to the required type. This is a delta-fidelity gap: $\Delta(a)$ does not assert that the committed entity belongs to the required cuisine set.

Case J.2 — Temporal compression (ChinaTravel Medium, solver UNSAT). *Request*: "2-day trip, Shanghai to Suzhou, 2 people. Visit a historical site."

Situation: The global agent schedules intercity train (08:00–09:30), breakfast, attraction, lunch, second attraction, dinner, return train (20:00).

Failure: At the return-train slot, the SMT core contains $order_{4,5}$ and min_dur_4, but all available return trains depart before 19:00.

System response: Global agent receives a repair request; revised blueprint removes one attraction, making an 18:30 return train feasible.

Lesson: The global agent should account for return train schedules when generating the blueprint. Injecting an explicit constraint $start_{\text{return}} \leq$ 19:00 (derived from the DB's latest train) into $\mathcal{C}_{\text{base}}$ would prevent this class of failure.

Case J.3 — Budget exhaustion on required slot (ChinaTravel Human). *Request*: "1-day trip, 2 people, budget 2,000 CNY. Hotel with a swimming pool."

Situation: Global agent budgets 400 CNY for the hotel. All swimming-pool hotels cost $\geq 1{,}500$ CNY/night. Two committed meals (¥600 each) and an attraction (¥400) leave only ¥400 for the hotel.

Failure: Hotel slot UNSAT (`budget_total` clause in core).

System response: Budget-repair path fires; global agent drops one free breakfast slot, freeing ¥600 and allowing the swimming-pool hotel to be grounded.

Lesson: Budget allocation at blueprint time should model the minimum required cost of membership-constrained entities. Querying the DB during formalization for the cheapest swimming-pool hotel would allow correct allocation from the start.

K Complete Ablation Results

Table 14 present the results of ablation study, To isolate the contribution of each component, we ablate the three core mechanisms of HMCTS-SMT without the SMT transition gate or the hierarchical decomposition, or the MCTS exploration policy, and compare each variant against the full system.

Table 14: Complete Coffee ablation results. Gap is measured over valid plans.

Model	Variant	Valid	Optimal	Invalid	Infeas.	Gap	Calls	Latency
Sonnet	Full	94.1%	82.4%	3.7%	6/7	0.42%	21.1	194 s
Sonnet	−Gate	85.0%	70.7%	12.4%	7/7	0.95%	30.5	215 s
Sonnet	−Hierarchy	91.2%	26.0%	6.2%	7/7	4.07%	3.5	46 s
Sonnet	Greedy	88.3%	25.3%	9.2%	7/7	6.52%	6.4	44 s
Haiku	Full	81.7%	64.5%	15.8%	7/7	1.53%	47.2	189 s
Haiku	−Gate	72.2%	58.6%	25.3%	7/7	0.76%	62.9	223 s
Haiku	−Hierarchy	80.2%	16.9%	17.2%	7/7	12.08%	5.3	40 s
Haiku	Greedy	75.8%	37.4%	21.6%	7/7	3.98%	6.7	24 s

Removing the SMT gate

Without the SMT gate, the search explores many more infeasible branches: LLM calls explode (Haiku reaches 63 calls versus 21 for the full system). The invalid plans shift from flow/demand logic to "other", suggesting that constraint-free proposals are accepted that violate harder-to-detect structural rules.

Removing the hierarchy

This is the most revealing ablation. Removing the hierarchy collapses optimality: the plan is often valid but far from optimal (65% suboptimal for Sonnet, 63% for Haiku). Without the decomposed search tree, the solver receives partial plans that satisfy the constraints but do not reach the cost minimum. Haiku without hierarchy is essentially useless for optimization (gap p90 $= 39\%$, max $= 73.8\%$).

Replacing MCTS with greedy selection

Without MCTS, selection is greedy and commits to the first acceptable solution. The valid rate remains reasonable, but optimality crashes: only 25–38% of plans are optimal versus 82.5% for the full HMCTS-SMT. The maximum gap reaches 48.6% (Sonnet, greedy), showing that greedy selection sometimes commits to a dramatically suboptimal solution. The zero restarts confirm that greedy search never backtracks.