A Category-Theoretic Approach to Neural-Symbolic Task Planning with Bidirectional Search

Shuhui Qu
Stanford University
shuhuiq@stanford.edu

Jie Wang
Stanford University
jiewang@stanford.edu

Kincho H. Law
Stanford University
law@stanford.edu

Abstract

We introduce a *Neural-Symbolic Task Planning* framework integrating Large Language Model (LLM) decomposition with category-theoretic verification for resource-aware, temporally consistent planning. Our approach represents states as objects and valid operations as morphisms in a categorical framework, ensuring constraint satisfaction through mathematical pullbacks. We employ bidirectional search that simultaneously expands from initial and goal states, guided by a learned planning distance function that efficiently prunes infeasible paths. Empirical evaluations across three planning domains demonstrate that our method improves completion rates by up to 6.6% and action accuracy by 9.1%, while eliminating resource violations compared to the existing baselines. These results highlight the synergy between LLM-based operator generation and category-theoretic verification for reliable planning in domains requiring both resource-awareness and temporal consistency.

1 Introduction

Effective task planning remains a critical challenge in artificial intelligence, particularly in domains where resource constraints, temporal consistency, and trustworthiness are paramount (Ghallab et al., 2004; Zhang et al., 2023; Jiang et al., 2024). Large Language Models (LLMs) (Achiam et al., 2023; Grattafiori et al., 2024; Touvron et al., 2023) offer powerful generative capabilities for natural language planning, but frequently overlook domain constraints (Wang et al., 2024; Valmeekam et al., 2024), yielding plans that violate resource limitations or temporal dependencies (Valmeekam et al., 2023). In contrast, classical symbolic planners (Pallagani et al., 2022; Illanes et al., 2020; Ghallab et al., 2004) ensure formal correctness but suffer from limited flexibility and require extensive domain engineering.

Recent research has attempted to bridge this conceptual gap through methods such as Chain-of-Thought (Wei et al., 2022), Monte Carlo Tree Search (MCTS)-based planning (Zhao et al., 2023), and reinforcement learning methods (Chen et al., 2025; Dalal et al., 2024). However, these approaches encode constraints as heuristic signals or sparse rewards (Havrilla et al., 2024; Huang et al., 2022) without providing structural guarantees. Other reasoning-oriented approaches such as Tree-of-Thoughts (ToT) (Yao et al., 2023a), ReWOO (Xu et al., 2023), and ToS (Katz et al., 2024) improve reasoning depth and search efficiency, but still lack mechanisms for ensuring compositional validity of generated plans. As benchmark evaluations of LLM planning expand (Stein et al., 2023; Wu et al., 2025), the need for principled approaches that unify neural flexibility with formal constraint enforcement becomes urgent.

We address these challenges by introducing *Neural-Symbolic Task Planning* (Figure 1). The framework comprises three key innovations:

1. **LLM-Driven Operator Decomposition:** A formalized technique for transforming natural language tasks into structured categorical specifications through iterative refinement, creating a bridge between unstructured language and mathematical formalism.

2. **Category-Theoretic Verification:** A novel framework that leverages category theory to represent planning domains, modeling states as objects and operations as morphisms in a categorical framework. By employing mathematical pullbacks, we provide compositional validity guarantees that ensure resource, temporal, and logical constraint satisfaction throughout the planning process.

3. **Bidirectional Search:** A theoretically-grounded algorithm that simultaneously ex-

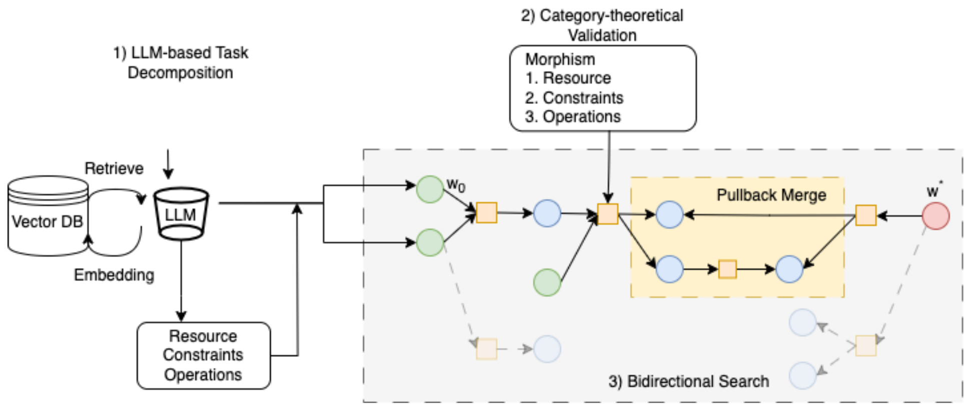


Figure 1: Neural-Symbolic Task Planning framework with three key stages: (1) LLM decomposition of natural language tasks into structured specifications, (2) category-theoretic verification to ensure constraint satisfaction, (3) bidirectional search to efficiently connect initial and goal states.

plores from initial and goal states guided by a categorical distance function, reducing computational complexity from $O(b^L)$ to $O(b^{L/2})$ while maintaining plan optimality.

Our contribution centers on the integration of category-theoretic verification with neural operator generation and search. This enables our framework to act as a constraint-safety layer that can be applied to LLM-driven planners, including CoT(Wei et al., 2022), ReAct(Yao et al., 2023b), ToT(Yao et al., 2023a), ensuring that generated plans remain resource-aware, temporally consistent, and logically valid.

We evaluate our framework across three diverse planning domains: cooking recipes (RecipeNLG) (Bień et al., 2020), procedural texts (ProcessBench) (Zheng et al., 2024), and standardized procedures (Proc2PDDL) (Zhang et al., 2024b). Our method consistently achieves 15–25% higher completion rates than other baselines, while substantially reducing resource/time violations by up to 77%. These results demonstrate that combining LLM-based operator generation with category-theoretic verification creates a powerful synergy for reliable, flexible planning in constraint-intensive domains.

2 Related Work

Classical Planning. Symbolic planners (Ghallab et al., 2004; Jiang et al., 2019; Höller et al., 2020) guarantee correctness but require extensive domain engineering and struggle with partially specified domains (Smirnov et al., 2024; Zhang et al., 2023). Hybrid approaches such as Fast-Downward (Helmert, 2006) and LAMA (Richter and Westphal, 2010) add heuristics, but they lack mechanisms for handling quantitative resource and temporal constraints.

LLM-Based Planning. Recent approaches leverage LLMs (Achiam et al., 2023; Touvron et al., 2023) to generate plans directly from text (Dagan et al., 2023; Song et al., 2023; Zeng et al., 2023), avoiding domain engineering. However, these models often act as black boxes that violate logical, temporal, or resource constraints (Valmeekam et al., 2022; Gestrin et al., 2024). To improve robustness, several works have introduced search-augmented techniques: Monte Carlo Tree Search (MCTS) (Zhao et al., 2023; Zhang et al., 2024a), ReAct (Yao et al., 2023b), Reflexion (Shinn et al., 2023), LLMFP(Hao et al., 2024), integrate dynamic programming(Dagan et al., 2023), or feedback-driven strategies (Shah et al., 2023; Suri et al., 2024). These methods demonstrate the potential of combining search with neural heuristics and LLM judge(Gu et al., 2024) but still lacking structural correctness guarantees(Kambhampati et al., 2024).

Reasoning-Oriented LLM Frameworks Parallel to direct plan generation, reasoning-oriented frameworks such as Tree-of-Thoughts(Yao et al., 2023a), ReWOO (Xu et al., 2023), and ToS (Katz et al., 2024) enhance reasoning depth and search efficiency by structuring LLM outputs into tree- or workflow-like processes. While effective for improving exploration, these methods also do not guarantee principled categorical verification when integrating multiple constraints across domains.

Neural-Symbolic Methods. Neural-symbolic approaches (DeLong et al., 2024; Mao et al., 2019) aim to combine neural flexibility with symbolic precision in domains such as visual reasoning (Hudson and Manning, 2019) and program synthesis (Ellis et al., 2021). Category theory provides powerful mathematical frameworks for compositional reasoning (Rydeheard and Burstall, 1988; Pierce, 1991; Jacob, 1990; Walters and Walters, 1991; Baez and Pollard, 2017), though prior applications have largely focused on symbolic systems without deep integration of neural operator generation.

Our framework uniquely combines the generative capabilities of LLMs with category-theoretic verification to structurally enforce resource, temporal, and logical constraints. By embedding pullback-based validation into a bidirectional search framework, we bridge the gap between the flexibility of LLM planners and the formal guarantees of symbolic reasoning.

3 Problem Statement

We formalize task planning as a category-theoretic framework where states are objects and operations are morphisms. Each state $w \in W = (r, s, l, t)$ encapsulates resources r, symbolic progress s, logical constraints l, and temporal allocations t. Morphisms $f : w_1 \to w_2$ represent valid state transitions that preserve resource bounds, state validity, constraint satisfaction, and temporal consistency.

Definition 3.1 (Planning Problem)**.** Given an initial state $w_0 = (r_0, s_0, l_0, t_0)$ and goal specification $w^* = (r^*, s^*, l^*, t^*)$, find a sequence of morphisms in planning category $\mathcal{T}$:

$$w_0 \xrightarrow{f_1} w_1 \xrightarrow{f_2} \cdots \xrightarrow{f_{n-1}} w_{n-1} \xrightarrow{f_n} w_n$$

such that each intermediate state w_i remains valid under categorical constraints, and w_n satisfies the criteria in w^*.

A more formal problem statement can be found in Appendix A.

4 Theoretical Analysis

In this section, we analyze the formal properties of the category-theoretic verification framework. We establish three key guarantees: local reachability, global completeness, and probabilistic completeness. Together, these theorems ensure that our approach preserves the rigor of symbolic planning while leveraging the generative flexibility of LLMs. Crucially, they highlight our main contribution: by embedding category-theoretic constructs (in particular, pullback-based verification) into an LLM-driven planner, we can provide structural guarantees that are missing from existing heuristic or black-box approaches.

4.1 State Space Properties

Let a planning distance function be $D : W \times W \to \mathbb{R}$ that estimates the minimum cost to transform one state into another. It enables theoretical guarantees through three properties:

1. **Component Integration**: D incorporates all four state components (resources, symbolic state, logical constraints, temporal intervals)
2. **Categorical Consistency**: It respects the category structure, with $D(w_1, w_2) < \infty$ only when morphisms can connect the states
3. **Continuous Measure**: It provides a differentiable measure of "plan difficulty" between states, guiding search toward promising paths

4.2 Theoretical Guarantees

Our first theorem establishes local reachability in the planning space:

Theorem 4.1 (ϵ-Reachability)**.** *For any two states $w_1, w_2 \in W$ with $D(w_1, w_2) < \epsilon$, there exists a sequence of valid morphisms $f_1, ..., f_k$ such that $f_k \circ ... \circ f_1(w_1) = w_2$, where $k \leq \lceil 1/\epsilon \rceil$.*

This theorem guarantees local connectivity of the categorical state space: nearby states can always be connected via a bounded number of morphisms. This ensures that our planner can efficiently explore neighborhoods of valid states without "falling out" of the constraint-respecting space. Proof can be found in Appendix B.

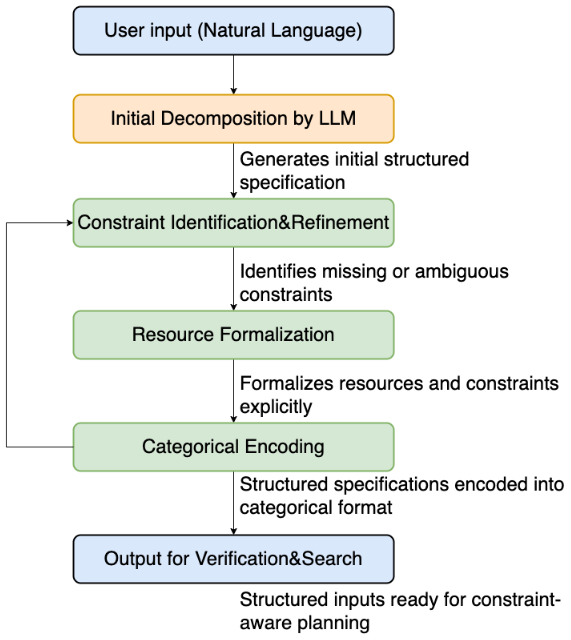
Building on local connectivity, we establish global completeness:

Theorem 4.2 (Completeness)**.** *If a valid plan exists between initial state w_0 and goal state w^*, the bidirectional search algorithm will find it.*

Completeness is the cornerstone of symbolic planning. By proving completeness despite the stochasticity of LLM-generated operators, we show that our neural-symbolic framework provides formal coverage guarantees—the planner will not overlook feasible solutions simply because of neural variability.

Theorem 4.3 (Probabilistic Completeness)**.** *Under bounded resources and finite constraints, the probability of finding a valid plan in n steps is:*

$$P(\textit{find plan in } n \textit{ steps}) \geq 1 - e^{-\lambda n} \quad (1)$$



User input (Natural Language)
Initial Decomposition by LLM
Generates initial structured specification
Constraint Identification&Refinement
Identifies missing or ambiguous constraints
Resource Formalization
Formalizes resources and constraints explicitly
Categorical Encoding
Structured specifications encoded into categorical format
Output for Verification&Search
Structured inputs ready for constraint-aware planning

Figure 2: Iterative LLM-based planning formulation process with feedback loops that enable progressive refinement from natural language to categorical representations.

where $\lambda > 0$ *depends on the reliability of LLM-generated morphisms.*

This result ensures robustness under uncertainty: even though LLM-generated morphisms may be noisy or inconsistent, our framework converges exponentially toward valid plans as the number of steps n increases. This property provides a strong theoretical foundation for the reliability under stochastic language-based operators.

The theoretical foundation is central to our contribution: category-theoretic verification not only ensures structural correctness of plans but also enables principled integration of neural generative models into symbolic reasoning.

5 Methodology

We now turn to our Neural-Symbolic Task Planning framework, which combines LLM-based operator generation, pullback-based verification, and bidirectional search to generate valid plans (Figure 1).

5.1 LLM-Based Task Decomposition

We transform high-level user queries into formal specifications through a systematic four-stage process using a pretrained Large Language Model (e.g., GPT-4, Llama) (Figure 2):

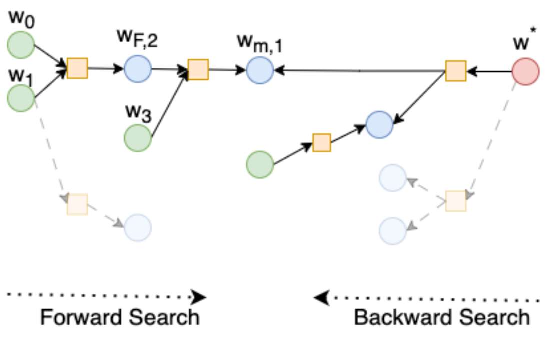


Figure 3: Bidirectional search reduces the effective search depth by simultaneously expanding from both the initial state w_0 and goal state w^*. When a pullback exists between states w_2^F and w^* (at meeting point $w_{m,1}$), a valid plan can be constructed with fewer expansions.

- **Initial Decomposition:** Extract candidate resources, operators, and constraints from natural language.
- **Constraint Refinement:** Identify ambiguities, clarify task specifications, and resolve implicit dependencies through targeted queries.
- **Resource Formalization:** Transform resource into typed, quantified specifications.
- **Categorical Encoding:** Encode specifications as categorical objects, morphisms, and constraints.

This iterative process uses feedback loops to progressively refine representations until they reach the precision required for category-theoretic planning, significantly reducing the manual engineering typically needed for symbolic approaches. To ensure reproducibility across domains, we provide in Appendix D a prompt template and guidelines that generalizes across domains.

5.2 Bidirectional Search

Task planning can be formulated using a variety of search and optimization strategies (e.g., A*, MCTS). We focus on bidirectional search, one of the most efficient formulations, as it reduces search depth from $O(b^L)$ to $O(b^{L/2})$ while retaining completeness guarantees, as illustrated in Figure 3. Our algorithm draws inspiration from Retro* and DESP (Xie et al., 2022; Yu et al., 2024) but is generalized to operate with category-theoretic validation. For a valid morphism sequence $\mathcal{P} = \{f_1, f_2, f_3, ...\}$, the total cost of the sequence is $\sum_1^n c(f)$, where $c(f)$ is the cost of applying morphism f.

5.2.1 Planning Distance

We now define our planning distance function D that estimates the minimum cost to transform one state into another as:

$$D(w_1, w_2) = \alpha_s d_s(s_1, s_2) + \alpha_r ||r_1 - r_2|| + \alpha_l d_l(l_1, l_2) + \alpha_t d_t(t_1, t_2) \quad (2)$$

where $\alpha_r, \alpha_s, \alpha_l, \alpha_t$ are weighting factors, and d_s, d_t, d_l are appropriate metrics for symbolic states, temporal components, and logical constraints, respectively. More details can be found in Appendix C[1]. This function serves as a domain-general heuristic that guides both forward search (from initial state) and backward search (from goal state), enabling efficient identification of promising meeting points. Importantly, the distance formulation is not specific to DESP or Retro* but can be embedded into a wide range of search frameworks (including A* and MCTS), making our approach adaptable across different planning backbones.

5.2.2 Search Graphs

We follow the same configuration as DESP and maintain two search graphs:

1. $\mathcal{G}^F$ (forward) initiates from w_0 and expands in a "bottom-up" manner by applying *forward morphisms* $f : w \to w'$.
2. $\mathcal{G}^B$ (backward) starts from w^* and expands "top-down" by applying *backward morphisms* that effectively invert feasible transitions.

The search uses an AND-OR graph structure (Xie et al., 2022) with objects in category $w \in W$ as OR-nodes and valid morphisms as AND-nodes(all children must be solved).

Our implementation supports two search strategies using a target condition function $\gamma : W \to W$:

- **Front-to-End (F2E)**: Target opposing end states: $\gamma(w) = w^*$ for $w \in \mathcal{G}^F$ and $\gamma(w) = w_0$ for $w \in \mathcal{G}^B$
- **Front-to-Front (F2F)**: Target closest states in opposing graph: $\gamma(w) = \arg\min_{w' \in \mathcal{G}^B} D(w, w')$ for $w \in \mathcal{G}^F$, $\gamma(w) = \arg\min_{w' \in \mathcal{G}^F} D(w', w)$ for $w \in \mathcal{G}^B$

[1]The planning distance $D(w_1, w_2)$ serves as a heuristic: it guides expansions but does not affect correctness, which is guaranteed by pullback verification. D can be learned from train–test splits (minimizing distance for valid transitions, maximizing for invalid ones) or replaced with simple metrics (e.g., cosine or L_2). Thus, training D improves efficiency but remains optional.

5.2.3 Search Procedure

The search procedure (Figure 4) selects and expands frontier states from both graphs:

Following Retro*, We let V_w be the minimum cost to achieve state w from w_0; $V_t(w|\mathcal{G})$ be the estimated cost of achieving w^* using state w given search graph $\mathcal{G}$; $rn(w|\mathcal{G})$ be the minimum cost to reach state w in search graph $\mathcal{G}$; D_w be the distance $D(\gamma(w), w)$ between a state and its target; $sn(w|\mathcal{G})$ be the step number represented as $D_w - V_w$ for related frontier nodes; $D_t(w|\mathcal{G})$ be the multiset of $D_w - V_w$ values along the minimum cost route through state w.

Frontier State Selection. Let $\mathcal{F}^F$ and $\mathcal{F}^B$ denote the frontier sets of the unsolved states in the forward and backward graphs, respectively.

For backward selection in the backward graph, we select a frontier state that minimizes the expected total cost of planning from the initial state w_0 to the goal state w^* through that state: $w_{\text{select},B} \leftarrow \arg\min_{w \in \mathcal{F}^B} \Big[V_t(w|\mathcal{G}^B) + \min(D_t(w|\mathcal{G}^B))\Big]$

The forward selection in the forward graph is identical to Retro*:

$$w_{\text{select},F} \leftarrow \arg\min_{w \in \mathcal{F}^F} V_t(w|\mathcal{G}^F)$$

State Expansion Policies. For backward expansion, we follow AND-OR-based algorithms in calling a single-step morphism, applying the top n predicted morphisms to the selected frontier node and adding the resulting morphisms and their states as nodes to the graph.

For state w in $\mathcal{G}^F$ (forward direction), we perform the forward expansion procedure:

- For state w, we generate successor states w' via morphisms $f : w \to w'$ and initialize $sn(w'|\mathcal{G}^F) \leftarrow V_{w'} = D(w', \gamma(w'))$

For state w in $\mathcal{G}^B$ (backward direction):

- For state w, we generate predecessor states w' via morphisms $f : w' \to w$ and initialize the values as:
  - $rn(w'|\mathcal{G}^B) \leftarrow V_{w'}$
  - $sn(w'|\mathcal{G}^B) \leftarrow D(\gamma(w'), w') - V_{w'}$

Value Propagation. After value initialization, for $\mathcal{G}^F$, we update values using the propagation from the Retro* algorithm.

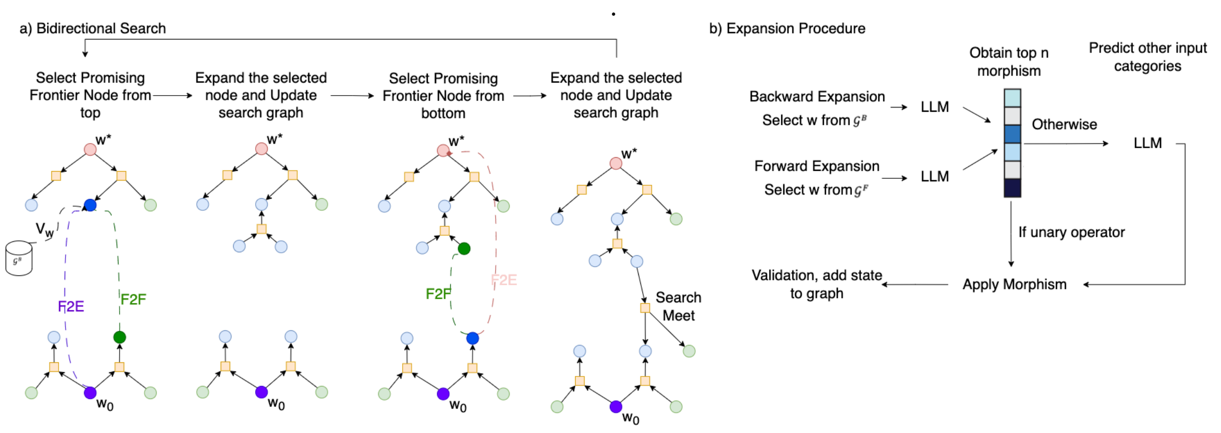

a) Bidirectional Search
Select Promising Frontier Node from top
Expand the selected node and Update search graph
Select Promising Frontier Node from bottom
Expand the selected node and Update search graph
w*
V_w
F2E
F2F
w0
Search Meet
b) Expansion Procedure
Obtain top n morphism
Predict other input categories
Backward Expansion
Select w from G^B
LLM
Otherwise
LLM
Forward Expansion
Select w from G^F
LLM
If unary operator
Validation, add state to graph
Apply Morphism

Figure 4: (a) Bidirectional Search algorithm. Evaluation of top nodes is based on both cost V_w and distance D. (b) Overview of the one-step expansion procedures.

For $\mathcal{G}^B$, we update the graphs through uppropagation and downpropagation. Similar to AND-OR algorithms, we first propagate updates to relevant values up the graph, and then down propagate to related nodes.

Uppropagation (for morphism nodes f and state nodes w):

$$sn(f|\mathcal{G}^B) \leftarrow \sum_{w \in ch(f)} sn(w|\mathcal{G}^B)$$

$$sn(w|\mathcal{G}^B) \leftarrow \begin{cases} [D_w - V_w] \text{ , if } w \in \mathcal{F}^B \\ sn\Big(\arg\min_{f \in ch(w)} rn(f)|\mathcal{G}^B \Big) \end{cases}$$

Downpropagation:

$$\begin{aligned} D_t(f|\mathcal{G}^B) \leftarrow{} & sn(pr(f)|\mathcal{G}^B) \\ & - sn\Big(\arg \min_{f' \in ch(pr(f))} rn(f'|\mathcal{G}^B)|\mathcal{G}_B \Big) \\ & + sn(f|\mathcal{G}^B) \end{aligned}$$

$$D_t(w|\mathcal{G}^B) \leftarrow D_t\Big(\arg \min_{f \in pr(w)} rn(f|\mathcal{G}^B)|\mathcal{G}^B \Big)$$

where the ch and pr functions denote the children and parent nodes; sn tracks the differences for nodes, enabling efficient propagation of cost estimates throughout the search graph. These update rules ensure that cost information flows correctly between states (objects in our category) and the morphisms connecting them.

5.2.4 Forward expansion policy with single-step morphism

LLM-based Morphism Generation. In this work, we use LLMs to generate valid morphisms through two key functions:

$$\phi_f : W \times W \to f = \text{LLM}(w_1, w_2)$$
$$\phi_w : W \times W \times f \to W = \text{LLM}(w_1, w_2, f)$$

The function ϕ_f generates candidate morphisms between states, while ϕ_w determines the resulting state after applying a morphism. These functions are implemented as structured prompts to the LLM that request specific outputs conforming to our categorical framework.

Merging via Pullbacks. Periodically, we attempt to connect the search graphs by finding states $w^F \in \mathcal{G}^F$ and $w^B \in \mathcal{G}^B$ with $D(w^F, w^B) < \epsilon$ that can be connected through category-theoretic pullback checks, where ϵ is a small value for threshold. When we find candidate states, we verify their compatibility using pullback checks and compose their respective plan fragments to obtain a complete sequence from w_0 to w^*.

5.3 Pullback Checks for Plan Validity

Pullbacks ensure plan compositions respect all constraints by computing potential pullback states and verifying their validity. When a valid pullback exists, we compose partial plans while guaranteeing constraint satisfaction. The verification process for states w_1 and w_2 with morphisms to a common state w_c works as follows:

1. Compute potential pullback state $w_p = (r_p, s_p, l_p, t_p)$ where:
    - r_p satisfies resource constraints for both states
    - $l_p = l_1 \wedge l_2$ (logical AND of constraints)
    - $t_p = t_1 \cap t_2$ (intersection of temporal intervals)
    - s_p is a valid symbolic state with transitions to both s_1 and s_2

2. Verify that w_p is a valid state (satisfies all capacity constraints)

3. Confirm that morphisms $p_1 : w_p \to w_1$ and $p_2 : w_p \to w_2$ exist

5.4 Algorithm Summary

Algorithm 1 in Appendix E outlines our bidirectional search procedure. The algorithm initializes search graphs from initial and goal states, then iteratively selects and expands states from both frontiers. After each expansion, it attempts to connect the search graphs via pullback checks. When a valid connection is found, it composes the partial plans to form a complete solution.

We establish the computational efficiency of our bidirectional search approach:

Theorem 5.1 (Time Complexity). *Given maximum path length L, branching factor b, and n states, the bi-directional search algorithm has time complexity* $O(b^{L/2})$.

This represents a quadratic improvement in the exponent compared to unidirectional search ($O(b^L)$), making our approach more efficient for practical applications.

6 Experiments

We evaluate our approach on three datasets with diverse planning characteristics: PLANBENCH (goal-oriented planning), RECIPENLG (resource and temporal constraints), and PROC2PDDL (formal planning with precondition/effect validation).

6.1 Datasets and Planning Scenarios

PlanBench. PlanBench[2] (Valmeekam et al., 2023) consists of 600 Blocksworld problems in PDDL format. Tasks involve transforming block configurations into goal states under logical constraints and cost minimization. We use a 50–50 train–test split.

RecipeNLG. RecipeNLG (Bień et al., 2020) contains cooking recipes with ingredient lists and step-by-step directions. We augment recipes with explicit resource limits (e.g., "$\leq$ 1/2 cup sugar" for health-conscious modifications) and temporal intervals (e.g., "bake 20–25 minutes") using GPT-4, testing quantitative resource and timing. We use an 80–20 train–test split.

[2]https://github.com/karthikv792/LLMs-Planning/

Proc2PDDL. Proc2PDDL[3] (Zhang et al., 2024b) provides 95 procedural texts with expert-annotated PDDL domain files across 27 domains. We evaluate precondition/effect prediction and executable plan generation using a 50–50 split per domain.

6.2 Baselines and Comparative Methods

We compare against direct prompting, reasoning-augmented prompting, and search-augmented planners, all using GPT-4o unless otherwise noted:

GPT-4o (Direct Prompting). Prompted with raw task descriptions and request step-by-step plans, without additional reasoning instructions.

CoT-GPT4o (Chain-of-Thought). Prompted with chain-of-thought. Explicit reasoning over resources, temporal requirements, and dependencies before producing a plan.

Thoughts-of-Search (Katz et al., 2024) Structures LLM exploration as a guided search tree for improved reasoning depth.

ReAct(Yao et al., 2023b) Interleaves reasoning traces with environment interactions to refine planning decisions.

LLM+P(Liu et al., 2023) Augments LLMs with symbolic planners for constraint-aware reasoning.

LLM-MCTS(Zhao et al., 2023) Monte Carlo Tree Search with 50 rollouts per problem, guided by LLM confidence scores.

Our approach combines LLM-based operator generation with category-theoretic verification and bidirectional search (details in Appendix C).

6.3 Evaluation Metrics

For PlenBench, we report: (1) **Completion rate**: Percentage of problems solved correctly; (2) **Cost optimality**: Percentage of solutions with minimal cost; For RecipeNLG: (3) **BLEU Score**; (4) **Constraint violations**: Percentage of solutions violating resource or (5) temporal constraints; For Proc2PDDL (6) **Action-wise accuracy**: Percentage of correctly predicted preconditions/effects; and (7) **Problem-file solve rate**: percentage of files executable in a PDDL solver.

6.4 Results

Table 1 summarizes performance across all datasets. Our approach consistently outperforms all baselines, achieving state-of-the-art results across PlanBench, RecipeNLG, and Proc2PDDL.

[3]https://github.com/zharry29/proc2pddl

Table 1: Performance comparison across all datasets. Best results in **bold**, second best <u>underlined</u>.

Method	PlanBench		RecipeNLG			Proc2PDDL	
	Comp%	Cost Opt%	BLEU	Res Viol%	Temp Viol%	Action Acc%	PF Solve%
GPT-4o	34.3	33.0	<u>0.903</u>	27.7	32.4	15.9	33.7
CoT-GPT4o	47.0	41.5	0.902	21.5	24.3	9.3	21.1
ToS	41.5	36.3	0.898	26.6	30.5	10.4	24.7
ReAct	63.0	56.8	**0.915**	19.4	22.9	34.6	43.7
LLM+P	<u>90</u>	<u>83.3</u>	0.888	<u>3.4</u>	<u>5.7</u>	<u>72.0</u>	<u>79.2</u>
LLM-MCTS	69.0	63.1	0.881	18.8	19.7	21.4	45.3
Ours	**96.6**	**93.5**	0.901	**0**	**1.4**	**81.1**	**87.4**

PlanBench Our method achieves the highest completion rate (96.6%) and cost optimality (93.5%), improving by 6.6% and 10.9% over the strongest LLM+P baseline; +27.6% and +30.4% over the LLM-MCTS. This demonstrates that category-theoretic verification effectively enforces logical dependencies (e.g., supporting block structures), preventing invalid moves that other LLM-based planners frequently make.

RecipeNLG All methods achieve comparable BLEU scores (0.881–0.915), suggesting similar textual quality. However, our method achieves near-perfect constraint satisfaction with 0% resource violations and only 1.4% temporal violations, far surpassing both LLM-MCTS (18.8%, 19.7%) and LLM+P (3.4%, 5.7%). This improvement is most pronounced in recipes with complex resource tracking requirements, such as recipes using partial ingredients across multiple steps. For example, when handling recipes requiring resource splitting (e.g., using half of an ingredient in one step given the global resource constraint), our pullback-based verification preserved consistency that baselines failed to capture.

Proc2PDDL This dataset is the most challenging, requiring formal reasoning over preconditions and effects. Our method achieves the highest action accuracy (81.1%) and solver success rate (87.4%), outperforming LLM+P by +9.1% and +8.2% respectively. The improvement is particularly significant for multi-step procedures with long-range dependencies, where pullback verification successfully preserves logical consistency throughout the planning process, which will be shown in our ablation study.

6.5 Ablation Studies

Reasoning vs. non-reasoning Table 2 shows the influence of LLM backbone type and scale. Reasoning vs. non-reasoning. Reasoning-augmented models (o4-mini, Claude-3.5, Qwen3-14B, DeepSeek-R1) achieve higher raw performance than non-reasoning models (GPT-4o, LLaMA). Our categorical verification, however, boosts both categories: for reasoning models, it enforces stricter constraint validity (e.g., o4-mini improves to 98.8% completion, 93.7% cost optimality); for non-reasoning models, it compensates for weaker reasoning depth, lifting LLaMA-3-13B to 91.0/83.3, rivaling much larger models.

Table 2: Performance comparison across difference LLM backbones.

Base LLM	PlanBench	
	Comp%	Cost Opt%
GPT-4o	96.6	93.5
o4-mini	98.8	93.7
Claude-3.5	94.3	91.0
LLaMA-3-70B	92.4	85.1
LLaMA-3-13B	91.0	83.3
LLaMA-3-8B	72.7	59.4
DeepSee-R1-Distill-Qwen-14B	94.9	88.2
Qwen3-14B	93.6	87.1

Scaling effect Larger backbones generally yield better results (LLaMA-3-70B at 92.4% vs. LLaMA-3-8B at 72.7%), but our framework narrows the scale gap: Qwen3-14B (93.6%) and DeepSeek-R1 (94.9%) approach or surpass the performance of GPT-4o and LLaMA-3-70B despite being smaller. This shows that verification amplifies the planning ability of mid-scale reasoning models, making them competitive with much larger non-reasoning backbones.

Distance functions Table 3 highlights the role of the planning distance D. Bidirectional search with a learned D achieves the best performance across all datasets, reducing constraint violations on RecipeNLG and boosting action accuracy on Proc2PDDL. However, even a raw metric D (cosine or L_2) performs well, showing that training D improves efficiency but is not essential for correct-

Table 3: Impact of difference distance function. all using LLaMA-3-13B unless otherwise noted.

Method	PlanBench		RecipeNLG		Proc2PDDL	
	Comp%	Cost Opt%	Res Viol%	Temp Viol%	Action Acc%	PF Solve%
MCTS + raw D	40.7	34.7	1.9	18.1	10.4	20.1
MCTS + learned D	61.2	57.3	15.6	16.3	16.2	31.7
Bidirectional + raw D	78.3	75.0	14.5	7.3	51.4	64.6
Bidirectional + learned D	**91.0**	**83.3**	**4.2**	**3.8**	**57.9**	**71.6**

Table 4: Impact of verification on PlanBench.

Variant	Comp (%)	Cost Opt (%)
With verification	**96.6**	**93.5**
Without verification	59.3	47.4
Absolute Difference	37.3	46.1

Table 5: Search strategy comparison on PlanBench for different Plan Length. (P.L.)

Search Strategy	Simple (<5 P.L.)	Complex (>5 P.L.)
Bidirectional	**98.1%**	**84.5%**
LLM-MCTS	88.3%	42.8%
GPT-4	65.2%	18.7%

ness verification guarantees validity regardless of distance quality.

Impact of verification. Table 4 shows that removing categorical verification reduces completion rates by 37.3% and cost optimality by 46.1% on PlanBench. The verification component ensures physical constraints in block stacking are maintained, preventing invalid moves such as removing blocks that support other blocks. Without verification, the planner generates invalid plans.

Search strategy comparison. Table 5 demonstrates the advantage of bidirectional search over alternatives, particularly as problem complexity increases. For complex problems with plan lengths exceeding 5 steps, bidirectional search achieves 84.5% completion, substantially outperforming LLM-MCTS (42.8%) and LLM-only approaches (18.7%). This performance gap widens exponentially with plan length. At 8-step plans, the completion rate difference between bidirectional search and LLM-MCTS increases to 38.9 percentage points. The deterioration in performance for non-bidirectional approaches occurs primarily at decision points requiring long-horizon planning. This confirms our theoretical complexity reduction from $O(b^L)$ to $O(b^{L/2})$ translates to practical performance gains on complex planning tasks.

These results demonstrate that both category-theoretic verification and bidirectional search contribute significantly to performance. Verification ensures plan validity while bidirectional search enables efficient exploration.

7 Conclusion

We introduced a Neural-Symbolic Task Planning framework integrating LLM-based decomposition with category-theoretic verification for resource-aware planning. By modeling states as categorical objects and operations as morphisms, our approach ensures constraint satisfaction through pullbacks while using bidirectional search for computational efficiency. Experiments across three domains demonstrate significant improvements over existing methods for completion rate and violation reduction. Our results establish category-theoretic verification as a promising approach for neural-symbolic planning in resource-constrained tasks.

7.1 Limitations

Our approach faces challenges with complex temporal dependencies, computational overhead for complex tasks with large state spaces despite the $O(b^{L/2})$ complexity reduction, and degraded performance when domain knowledge is missing from the LLM's pre-training. Nevertheless, our experiments confirm that neural-symbolic integration substantially improves constraint satisfaction while maintaining natural language flexibility.

Acknowledgments

This research is partially supported by Stanford's Center for Sustainable Development and Global Competitiveness (SDGC) and the Yonghua Foundation. The authors would like to thank Dr. Spencer Breiner and Dr. Ram Sriram of the US National Institute of Standards and Technology and Dr. Eswaran Subrahmanian of Carnegie Mellon University for their helpful comments and suggestions.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

John C Baez and Blake S Pollard. 2017. A compositional framework for reaction networks. *Reviews in Mathematical Physics*, 29(09):1750028.

Michał Bień, Michał Gilski, Martyna Maciejewska, Wojciech Taisner, Dawid Wisniewski, and Agnieszka Lawrynowicz. 2020. Recipenlg: A cooking recipes dataset for semi-structured text generation. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 22–28.

Kevin Chen, Marco Cusumano-Towner, Brody Huval, Aleksei Petrenko, Jackson Hamburger, Vladlen Koltun, and Philipp Krähenbühl. 2025. Reinforcement learning for long-horizon interactive llm agents. *arXiv preprint arXiv:2502.01600*.

Gautier Dagan, Frank Keller, and Alex Lascarides. 2023. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*.

Murtaza Dalal, Tarun Chiruvolu, Devendra Chaplot, and Ruslan Salakhutdinov. 2024. Plan-seq-learn: Language model guided rl for solving long horizon robotics tasks. *arXiv preprint arXiv:2405.01534*.

Lauren Nicole DeLong, Ramon Fernández Mir, and Jacques D Fleuriot. 2024. Neurosymbolic ai for reasoning over knowledge graphs: A survey. *IEEE Transactions on Neural Networks and Learning Systems*.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. 2021. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pages 835–850.

Elliot Gestrin, Marco Kuhlmann, and Jendrik Seipp. 2024. Nl2plan: Robust llm-driven planning from minimal text descriptions. *arXiv preprint arXiv:2405.04215*.

Malik Ghallab, Dana S. Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*. Elsevier.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2024. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*.

Yilun Hao, Yang Zhang, and Chuchu Fan. 2024. Planning anything with rigor: General-purpose zero-shot planning with llm-based formalized programming. *arXiv preprint arXiv:2410.12112*.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. 2024. Teaching large language models to reason with reinforcement learning. *arXiv preprint arXiv:2403.04642*.

Malte Helmert. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

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

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Drew Hudson and Christopher D Manning. 2019. Learning by abstraction: The neural state machine. *Advances in neural information processing systems*, 32.

León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. 2020. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the international conference on automated planning and scheduling*, volume 30, pages 540–550.

Jeremy Jacob. 1990. Categorising non-interference. In *[1990] Proceedings. The Computer Security Foundations Workshop III*, pages 44–50. IEEE.

Xue Jiang, Yihong Dong, Lecheng Wang, Zheng Fang, Qiwei Shang, Ge Li, Zhi Jin, and Wenpin Jiao. 2024. Self-planning code generation with large language models. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–30.

Yu-qian Jiang, Shi-qi Zhang, Piyush Khandelwal, and Peter Stone. 2019. Task planning in robotics: an empirical comparison of pddl-and asp-based systems. *Frontiers of Information Technology & Electronic Engineering*, 20:363–373.

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

Michael Katz, Harsha Kokel, Kavitha Srinivas, and Shirin Sohrabi Araghi. 2024. Thought of search: Planning with language models through the lens of efficiency. *Advances in Neural Information Processing Systems*, 37:138491–138568.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*.

Vishal Pallagani, Bharath Muppasani, Keerthiram Murugesan, Francesca Rossi, Lior Horesh, Biplav Srivastava, Francesco Fabiano, and Andrea Loreggia. 2022. Plansformer: Generating symbolic plans using transformers. *arXiv preprint arXiv:2212.08681*.

Benjamin C Pierce. 1991. *Basic category theory for computer scientists*. MIT press.

Silvia Richter and Matthias Westphal. 2010. The lama planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.

David E Rydeheard and Rod M Burstall. 1988. *Computational category theory*, volume 152. Prentice Hall Englewood Cliffs.

Dhruv Shah, Michael Robert Equi, Błażej Osiński, Fei Xia, Brian Ichter, and Sergey Levine. 2023. Navigation with large language models: Semantic guesswork as a heuristic for planning. In *Conference on Robot Learning*, pages 2683–2699. PMLR.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

Pavel Smirnov, Frank Joublin, Antonello Ceravola, and Michael Gienger. 2024. Generating consistent pddl domains with large language models. *arXiv preprint arXiv:2404.07751*.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2998–3009.

Katharina Stein, Daniel Fišer, Jörg Hoffmann, and Alexander Koller. 2023. Autoplanbench: Automatically generating benchmarks for llm planners from pddl. *arXiv preprint arXiv:2311.09830*.

Gaurav Suri, Lily R Slater, Ali Ziaee, and Morgan Nguyen. 2024. Do large language models show decision heuristics similar to humans? a case study using gpt-3.5. *Journal of Experimental Psychology: General*, 153(4):1066.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

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

Robert Frank Carslaw Walters and Richard F Walters. 1991. *Categories and computer science*. Cambridge University Press.

Kevin Wang, Junbo Li, Neel P Bhatt, Yihan Xi, Qiang Liu, Ufuk Topcu, and Zhangyang Wang. 2024. On the planning abilities of openai's o1 models: Feasibility, optimality, and generalizability. *arXiv preprint arXiv:2409.19924*.

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

Zirui Wu, Xiao Liu, Jiayi Li, Lingpeng Kong, and Yansong Feng. 2025. Haste makes waste: Evaluating planning abilities of llms for efficient and feasible multitasking with time constraints between actions. *arXiv preprint arXiv:2503.02238*.

Shufang Xie, Rui Yan, Peng Han, Yingce Xia, Lijun Wu, Chenjuan Guo, Bin Yang, and Tao Qin. 2022. Retrograph: Retrosynthetic planning with graph search. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2120–2129.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. Rewoo: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

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

Kevin Yu, Jihye Roh, Ziang Li, Wenhao Gao, Runzhong Wang, and Connor Coley. 2024. Double-ended synthesis planning with goal-constrained bidirectional search. *Advances in Neural Information Processing Systems*, 37:112919–112949.

Zhen Zeng, William Watson, Nicole Cho, Saba Rahimi, Shayleen Reynolds, Tucker Balch, and Manuela Veloso. 2023. Flowmind: automatic workflow generation with llms. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pages 73–81.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024a. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772.

Shun Zhang, Zhenfang Chen, Yikang Shen, Mingyu Ding, Joshua B Tenenbaum, and Chuang Gan. 2023. Planning with large language models for code generation. *arXiv preprint arXiv:2303.05510*.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. 2024b. Proc2pddl: Open-domain planning representations from texts. *arXiv preprint arXiv:2403.00092*.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. *Advances in Neural Information Processing Systems*, 36:31967–31987.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2024. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*.

A Formal Problem Statement

Here, we show the formal problem statement.

A.1 Category-Theoretic Planning Framework

We formalize planning as a category-theoretic problem where states are objects and operations are morphisms. Each state captures resource usage, active constraints, symbolic progress, and temporal allocations. The morphisms represent valid transitions/operations that preserve these properties through constraint verification.

Definition A.1 (Planning Domain)**.** A planning domain consists of:

- A set of resource types I, where each type $i \in I$ has an associated ordered monoid $(R_i, +i, \leq_i, 0_i)$ and capacity bound ri, max
- A set of symbolic states S with a directed graph $G_S = (S, E_S)$ of valid transitions
- A set of logical constraints $\mathcal{L}$ expressed as predicates over resources, states, and temporal properties
- A temporal framework $\mathcal{T}$ for representing time intervals and precedence relations

Definition A.2 (Planning Category)**.** Let $\mathcal{T}$ be a category whose objects are hybrid task states $w = (r, s, l, t)$ where:

- $r \in R = \prod_{i \in I} R_i$ represents resource configuration, with each component $r[i] \leq r_{i,max}$ respecting capacity bounds
- $s \in S$ is a discrete symbolic state from the state transition graph G_S
- $l \in L = 0, 1^k$ is a boolean vector encoding k logical constraints, where $l[j] = 1$ indicates constraint j is satisfied
- $t \in T \subseteq \mathbb{R}^+ \times \mathbb{R}^+ \times \mathcal{P}(I)$ represents temporal intervals $[t_{start}, t_{end}]$ and scheduling constraints over a set of interval relations I

Definition A.3 (Morphism)**.** A morphism $f : w_1 \to w_2$ in $\mathcal{T}$ transforms state $w_1 = (r_1, s_1, l_1, t_1)$ to state $w_2 = (r_2, s_2, l_2, t_2)$ while preserving categorical constraints. The transformation is characterized by component functions (f_r, f_s, f_l, f_t) that may depend on all aspects of the input state, ensuring:

- **Resource validity**: $r_2 = f_r(r_1, s_1, l_1, t_1)$ where $r_2[i] \leq r_{i,max}$ for all resource types i. Resource transformations respect the properties of the underlying ordered monoids.
- **State transitions**: $s_2 = f_s(r_1, s_1, l_1, t_1)$ such that $(s_1, s_2) \in E_S$ is an edge in the state transition graph, and all preconditions for the transition are satisfied.
- **Constraint satisfaction**: $l_2 = f_l(r_1, s_1, l_1, t_1)$ where:
  - Invariant constraints remain satisfied: if $l_1[j]$ is an invariant and $l_1[j] = 1$ then $l_2[j] = 1$
  - Postcondition constraints may be established: l_2 may have additional satisfied constraints
  - Precondition constraints are checked before applying the morphism
- **Temporal consistency**: $t_2 = f_t(r_1, s_1, l_1, t_1)$ preserves precedence relations and ensures non-overlapping intervals for mutually exclusive operations.

Each morphism has an associated probability $p(f) \in [0, 1]$ reflecting its empirical success rate. Morphism composition $g \circ f$ is valid if and only if all component functions compose and preserve the above constraints.

As shown in the methodolgy, in our neural-symbolic framework, morphism are generated by an LLM conditioned on the current state. The detail will be explained in the next section.

The power of our framework comes from compositional verification using categorical pullbacks:

Definition A.4 (Pullback)**.** Given morphisms $f : A \to C$ and $g : B \to C$ in category $\mathcal{T}$, a pullback consists of:

- An object P (the pullback object)
- Morphisms $p_1 : P \to A$ and $p_2 : P \to B$

such that $f \circ p_1 = g \circ p_2$ (i.e., both paths from P to C yield the same result), forming a commutative square. Furthermore, for any other object Q with morphisms $q_1 : Q \to A$ and $q_2 : Q \to B$ satisfying $f \circ q_1 = g \circ q_2$, there exists a unique morphism $u : Q \to P$ such that $p_1 \circ u = q_1$ and $p_2 \circ u = q_2$ (i.e., the morphism u preserves all path relationships).

Lemma A.5 (Pullback Structure)**.** *Given morphisms $f : A \to C$ and $g : B \to C$ in category $\mathcal{T}$, if a pullback exists, it is an object P with morphisms $p_1 : P \to A$ and $p_2 : P \to B$ such that:*

- *$P = (r_P, s_P, l_P, t_P)$ where:*
  - *r_P satisfies $p_{1r}(r_P) = r_A$ and $p_{2r}(r_P) = r_B$*
  - *s_P is a symbolic state with valid transitions to both s_A and s_B*
  - *l_P^{inv} preserves all invariant constraints from both l_A^{inv} and l_B^{inv}*
  - *t_P is a valid refinement of both t_A and t_B*
- *The diagram commutes: $f \circ p_1 = g \circ p_2$*
- *For any object Q with morphisms $q_1 : Q \to A$ and $q_2 : Q \to B$ satisfying $f \circ q_1 = g \circ q_2$, there exists a unique morphism $u : Q \to P$ such that $p_1 \circ u = q_1$ and $p_2 \circ u = q_2$*

Theorem A.6 (Plan Compatibility Characterization)**.** *Given morphisms $f : A \to C$ and $g : B \to C$ in category $\mathcal{T}$:*

1. *If a pullback of f and g exists, then the plans represented by f and g are compatible, meaning:*
   - *Resource usage from both plans can be combined without exceeding capacity limits*
   - *All invariant logical constraints from both plans can be simultaneously satisfied*
   - *Time intervals from both plans can be merged without violating precedence constraints*
2. *Conversely, if no pullback exists, then the plans are incompatible with respect to at least one of these constraint types.*

Constructively, given states w_A and w_B with morphisms to common state w_C, the pullback state $w_P = (r_P, s_P, l_P, t_P)$ can be computed as:

- Resources: For each resource type i, $r_P[i]$ is a minimum valid configuration that maps to both $r_A[i]$ and $r_B[i]$ through the respective morphisms
- Logical state: $l_P[j] = l_A[j] \wedge l_B[j]$ for invariant constraints (logical AND)
- Temporal windows: $t_P = t_A \cap t_B$ (interval intersection) when non-empty
- Symbolic state: A valid state s_P with transitions to both s_A and s_B in the state graph G_S

Definition A.7 (Planning Problem)**.** Given:

- Initial state $w_0 = (r_0, s_0, l_0, t_0)$ with available resources and constraints
- Goal specification $w^* = (r^*, s^*, l^*, t^*)$ defining desired properties

Find a sequence of morphisms in $\mathcal{T}$ for a *planning category*:

$$w_0 \xrightarrow{f_1} w_1 \xrightarrow{f_2} \cdots \xrightarrow{f_{n-1}} w_{n-1} \xrightarrow{f_n} w_n$$

such that each intermediate state w_i remains valid under the categorical constraints, and w_n satisfies or exceeds the criteria in w^*.

While LLMs can generate candidate morphisms, they may produce invalid or inconsistent operations. Our framework addresses this by integrating LLM-based generation with category-theoretic verification. For single operations (unary morphisms), we directly verify constraint satisfaction. For combining plan fragments (binary morphisms), we use pullbacks to ensure compositional validity.

B Proof

B.1 Plan Composition

Proof. Let $f : A \to C$ and $g : B \to C$ be morphisms in our planning category $\mathcal{T}$, where states are represented as $w = (r, s, l, t)$. Let the pullback object be P with projections $p_1 : P \to A$ and $p_2 : P \to B$ such that $f \circ p_1 = g \circ p_2$. We prove each guarantee in turn:

1. Resource Compatibility: By definition, the resource component of our states is represented as a vector $r \in R \subseteq \mathbb{R}^n$ subject to capacity constraints. For valid morphisms f and g, we have:

$$f_r(r_A) = r_C \text{ and } g_r(r_B)$$

Let the resource component at the pullback be r_P. By the universal property of pullbacks, r_P must map to both r_A and r_B through p_{1r} and p_{2r} respectively:

$$p_{1r}(r_P) = r_A \text{ and } p_{2r}(r_P) = r_B$$

For these mappings to exist, r_P must satisfy the resource constraints of both plans. Since resource transformations in our category are monotonic (resources can only be consumed, not created), r_P must contain at least the maximum resource requirements of both plans. Formally:

$$r_P[i] \geq \max(r_A[i], r_B[i]) \text{ for each resource dimension}$$

Given that f and g are valid morphisms respecting capacity constraints, we know:

$$r_A[i] \leq r_{max}[i] \text{ and } r_B[i] \leq r_{max}[i]$$

Therefore:

$$r_P[i] \leq r_{max}[i] \text{ for all } i$$

Thus, the combined resource usage at P remains within capacity constraints.

2. Logical Consistency: Let the logical constraint vectors be l_A, l_B, and l_P for states A, B, and P respectively. Valid morphisms in our category must preserve satisfied constraints monotonically, meaning:

$$\text{If } l_A[j] = 1, \text{ then } l_C[j] = 1$$

$$\text{If } l_B[j] = 1, \text{ then } l_C[j] = 1$$

For the pullback object P, the logical constraints must be consistent with both A and B. Since constraint satisfaction is preserved by morphisms, l_P must satisfy:

$$\text{If } l_P[j] = 1, \text{ then } l_A[j] = 1 \text{ and } l_B[j] = 1$$

Conversely, if a constraint is satisfied in both A and B, it must be satisfied in P:

$$\text{If } l_A[j] = 1 \text{ and } l_B[j] = 1, \text{ then } l_P[j] = 1$$

This construction ensures that l_P preserves all constraints satisfied in both A and B, while not introducing any new constraints that would create inconsistencies when mapped to either A or B.

3. Temporal Coherence: For the temporal component, let $t_A = [t_A.start, t_A.end]$, $t_B = [t_B.start, t_B.end]$, and $t_P = [t_P.start, t_P.end]$ represent the time intervals for states A, B, and P respectively. Valid morphisms in our category must preserve temporal ordering and non-overlapping constraints. For the pullback to exist, the time intervals must be compatible, meaning there exists a valid time interval t_P that can be mapped to both t_A and t_B while preserving ordering constraints. The most general such interval is the intersection:

$$t_P.start = \max(t_A.start, t_B.start)$$

$$t_P.end = \min(t_A.end, t_B.end)$$

For this interval to be valid, we must have $t_P.start \le t_P.end$, which is guaranteed when t_A and t_B have a non-empty intersection. When no such intersection exists, the pullback does not exist, correctly indicating that the plans cannot be composed with respect to their temporal constraints. For the precedence relations in $\mathcal{P}(\mathcal{I})$, the pullback preserves all shared precedence constraints between t_A and t_B. Any precedence relation satisfied in both partial plans will be preserved in the pullback. Thus, when a pullback exists, the time intervals from both plans can be merged without temporal conflicts. Therefore, the existence of a pullback P for morphisms $f : A \to C$ and $g : B \to C$ guarantees resource compatibility, logical consistency, and temporal coherence of the composed plan. □

B.2 Reachability

Proof of Theorem 4.1 (ϵ-Reachability). Let $w_1 = (r_1, s_1, l_1, t_1)$ and $w_2 = (r_2, s_2, l_2, t_2)$ be states in W with $D(w_1, w_2) < \epsilon$, where ϵ is sufficiently small. We show the existence of a sequence of valid morphisms $f_1, f_2, \dots, f_k$ such that $f_k \circ \cdots \circ f_1(w_1) = w_2$ where $k \le \lceil 1/\epsilon \rceil$.

We construct a sequence of intermediate states $\{w_1 = \tilde{w}_0, \tilde{w}_1, \dots, \tilde{w}_k = w_2\}$ and corresponding morphisms $f_i : \tilde{w}_{i-1} \to \tilde{w}_i$ such that each transition is valid according to our category definition.

Construction: Let $p : [0,1] \to W$ be a continuous path such that $p(0) = w_1$ and $p(1) = w_2$, where w_1 to w_2 are our state space. Such a path exists since the state components:

- Resources r and temporal components t are continuous
- Symbolic states $\phi(s)$ are continuous and connected by valid transition function.
- Logical constraints l that can be updated monotonically

We partition $[0,1]$ into $\lceil 1/\epsilon \rceil$ equal intervals and define intermediate states:

$$\tilde{w}_i = p\left(\frac{i}{\lceil 1/\epsilon \rceil}\right) \text{ for } i = 0, 1, \dots, \lceil 1/\epsilon \rceil$$

Validity of Transitions: For each pair of consecutive states $\tilde{w}_{i-1}$ and $\tilde{w}_i$, we have:

$$D(\tilde{w}_{i-1}, \tilde{w}_i) \le \frac{D(w_1, w_2)}{\lceil 1/\epsilon \rceil} < \frac{\epsilon}{\lceil 1/\epsilon \rceil} \le \epsilon'$$

We now verify that there exists a valid morphism $f_i : \tilde{w}_{i-1} \to \tilde{w}_i$ for each pair:

1. **Resource Component:** For resources, let $\tilde{r}_{i-1}$ and $\tilde{r}_i$ be the resource vectors of $\tilde{w}_{i-1}$ and $\tilde{w}_i$. $||\tilde{r}_{i-1} - \tilde{r}_i|| \le \frac{||r_1 - r_2||}{\lceil 1/\epsilon \rceil} < \frac{\epsilon/\alpha_r}{\lceil 1/\epsilon \rceil}$ is sufficiently small, given a sufficiently small ϵ. We can thus define a valid resource transformation $f_{ir}(\tilde{r}_{i-1}) = \tilde{r}_i$ that respects capacity bounds.
2. **Symbolic State:** For symbolic states, $\tilde{s}_{i-1}$ and $\tilde{s}_i$, the distance $||\phi_s(\tilde{s}_{i-1}) - \phi_s(\tilde{s}_i)|| < \frac{\epsilon/\alpha_s}{\lceil 1/\epsilon \rceil}$. Given a sufficiently small ϵ, either $\tilde{s}_{i-1} = \tilde{s}_i$ or there exists a direct valid transition between them.

3. **Logical Constraints:** For logical constraints $\tilde{l}_{i-1}$ and $\tilde{l}_i$, we have $||\phi_l(\tilde{l}_{i-1}) - \phi_l(\tilde{l}_i)|| < \frac{\epsilon/\alpha_l}{\lceil 1/\epsilon \rceil}$. Given the monotonicity requirement (constraints can only be added, not removed), we ensure that each intermediate state only adds constraints that are satisfied in w_2. In other word, for sufficiently small ϵ, at most one constraint changes per step.

4. **Temporal Component:** For temporal components $\tilde{t}_{i-1}$ and $\tilde{t}_i$, we have $||\phi_t(\tilde{t}_{i-1}) - \phi_t(\tilde{t}_i)|| < \frac{\epsilon/\alpha_t}{\lceil 1/\epsilon \rceil}$. Since temporal changes must preserve precedence relations and scheduling constraints, we define the transformation to gradually adjust time intervals while maintaining these properties.

Composition of Morphisms: We define each morphism $f_i : \tilde{w}_{i-1} \to \tilde{w}_i$ as the tuple:

$$f_i = (f_{ir}, f_{is}, f_{il}, f_{it})$$

Each component function is constructed to ensure the validity conditions of our category. By the category axioms, each f_i is a valid morphism in $\mathcal{T}$.

Plan Length: The total number of morphisms in our constructed sequence is $k = \lceil 1/\epsilon \rceil$, and the composition $f_k \circ \cdots \circ f_1$ transforms w_1 into w_2 as required.

Therefore, for any two states $w_1, w_2 \in W$ with $D(w_1, w_2) < \epsilon$, there exists a sequence of at most $\lceil 1/\epsilon \rceil$ valid morphisms connecting them. □

B.3 Completeness

Proof of Theorem 4.2 (Completeness). We need to prove that if a valid plan exists between initial state w_0 and goal state w^*, then our bidirectional search algorithm will find it.

Step 1: Plan Existence and State Space Coverage. Let $P^* = \{f_1, f_2, \ldots, f_n\}$ be a valid plan from w_0 to w^*, where each f_i is a morphism in our category $\mathcal{T}$. This plan induces a sequence of states $w_0, w_1, w_2, \ldots, w_n = w^*$ where $w_i = f_i(w_{i-1})$.

Given our distance metric D, we can choose $\epsilon > 0$ such that any state in our search space is within ϵ-distance of at least one state in the optimal plan P^*. This is possible because:

1. The resource space R is bounded by capacity constraints
2. The symbolic state space S is finite
3. The logical constraint space L is finite (with 2^k possible configurations)
4. The temporal space T has bounded time windows

Therefore, we can construct a finite covering of the state space with ϵ-balls centered on states in the optimal plan.

Step 2: Bidirectional Search Properties. Our bidirectional search algorithm maintains two search graphs:

1. $\mathcal{G}^F$ expanding forward from w_0
2. $\mathcal{G}^B$ expanding backward from w^*

We use a planning distance function D to guide expansions, where $\text{val}^F(w) = V(w) + \min_\gamma D(w, \gamma)$ and $\text{val}^B(w) = V(w) + \min_\gamma D(\gamma, w)$.

At each iteration, the algorithm:

1. Selects the most promising state to expand from each frontier
2. Expands valid operators from these states
3. Attempts to merge partial plans via pullback checks

Step 3: Forward Reachability. We first show that all states in the optimal plan P^* are eventually reached by the forward search.

For each state w_i in the optimal plan,Let $V^*(w_i)$ be the true optimal cost to reach w_i from w_0 and $V(w_i)$ be our algorithm's current estimate of this cost.

We claim that for each w_i, there exists a time when a state w' with $D(w', w_i) < \epsilon$ enters the forward frontier $\mathcal{F}^F$.

Proof by induction:

1. Base case: w_0 is in $\mathcal{F}^F$ initially
2. Inductive step: Assume w'_{i-1} with $D(w'_{i-1}, w_{i-1}) < \epsilon$ is in $\mathcal{F}^F$
3. By Theorem 4.1, there exists a sequence of valid operators from w'_{i-1} to a state w'_i with $D(w'_i, w_i) < \epsilon$
4. Since our algorithm expands all valid operators from frontier states, w'_i will eventually enter $\mathcal{F}^F$

Therefore, the forward search eventually reaches a state near each state in the optimal plan.

Step 4: Backward Reachability. Similarly, for the backward search, all states in the optimal plan are eventually reached by the backward search. For each state w_i in the optimal plan, there exists a time when a state w'' with $D(w'', w_i) < \epsilon$ enters the backward frontier $\mathcal{F}^B$.

Step 5: Meeting of Frontiers. Given Steps 3 and 4, there will eventually be states $w'_i \in \mathcal{F}^F$ and $w''_j \in \mathcal{F}^B$ such that:

1. $D(w'_i, w_i) < \epsilon$
2. $D(w''_j, w_j) < \epsilon$
3. $|i - j| \leq 1$ (the states are adjacent in the optimal plan)

Step 6: Pullback Existence. Given that our states w'_i and w''_j are near adjacent states in the optimal plan, and that the optimal plan respects all constraints, a pullback exists that allows the composition of the forward and backward plans.

The existence of this pullback ensures that our algorithm can merge the partial plans to form a complete plan from w_0 to w^*.

Step 7: Termination. Since our state space is finite under resource bounds and our algorithm systematically explores the space guided by the planning distance D, it will eventually discover the merger point where the pullback exists.

Therefore, if a valid plan exists, our bidirectional search algorithm will find it. □

B.4 Probabilistic Completeness Theorem

Proof of Theorem 4.3 (Probabilistic Completeness). We need to prove that under bounded resources and finite constraints, the probability of finding a valid plan within n steps is at least $1 - e^{-\lambda n}$ for some constant $\lambda > 0$.

This proof addresses the stochastic nature of LLM-generated operators, which introduces uncertainty into the planning process. While our category-theoretic verification ensures that operators are valid when applied, the generation of candidate operators by the LLM involves randomness.

Probabilistic Model: Let us define the following:

- P^* is a valid plan from initial state w_0 to goal state w^*, known to exist by assumption.
- $p_{\min}$ is the minimum probability that the LLM generates a valid operator at any given step of the plan.
- At each step, the LLM may generate multiple candidate operators, but our focus is on whether at least one valid operator toward the solution is among them.

In practice, our algorithm adaptively refine the operator to further boost $p_{\min}$.

Single-Step Success Probability: At each step of the planning process, the LLM generates candidate operators. Let's define:

- E_i: the event that the LLM generates at least one operator at step i that advances the plan toward the goal.
- $p_i = P(E_i)$: the probability of event E_i occurring.

Given our bounded resource assumptions and the categorical structure of our planning domain, the number of possible states is finite. Furthermore, since the LLM's operator generation is based on learned statistical patterns, there exists a minimum probability $p_{\min} > 0$ such that:

$$p_i \geq p_{\min} \quad \text{for all } i \tag{3}$$

This lower bound $p_{\min}$ represents the LLM's worst-case performance in generating useful operators for our planning domain.

Multi-Step Analysis: Finding a valid plan requires successfully generating valid operators for multiple consecutive steps. We model this as a sequence of Bernoulli trials, where each trial corresponds to an attempt to advance the plan by one step.

Let X_n be the random variable representing the number of successful steps completed after n attempts. We're interested in $P(X_n \geq L)$, where L is the length of the optimal plan.

Markov Chain Representation: We can model the planning process as a Markov chain where:

- States correspond to the progress made (number of steps completed toward the goal).
- Transitions occur with probability at least $p_{\min}$ for advancement and at most $(1 - p_{\min})$ for staying in the same state.

This is a birth process with a minimum birth probability of $p_{\min}$. The probability of reaching state L (completing the plan) within n steps can be analyzed using standard results from Markov chain theory.

Deriving the Bound: For a birth process with minimum birth probability $p_{\min}$, the probability of not reaching state L within n steps is bounded by:

$$P(X_n < L) \leq (1 - p_{\min}^L)^{\lfloor n/L \rfloor} \tag{4}$$

This bound reflects that every sequence of L consecutive steps has a probability of at least $p_{\min}^L$ of completing the entire plan.

For large n, we can approximate this as:

$$P(X_n < L) \leq e^{-p_{\min}^L \cdot \lfloor n/L \rfloor} \leq e^{-\lambda n} \tag{5}$$

where $\lambda = p_{\min}^L / L$ is a positive constant.

Therefore, the probability of finding a valid plan within n steps is:

$$P(X_n \geq L) = 1 - P(X_n < L) \geq 1 - e^{-\lambda n} \tag{6}$$

Connection to LLM Confidence: The parameter λ in our bound is directly related to the LLM's operator generation capability:

$$\lambda = \frac{p_{\min}^L}{L} \tag{7}$$

A more capable LLM with higher confidence in generating valid operators would have a larger $p_{\min}$, resulting in a larger λ and faster convergence.

Practical Implications: This bound guarantees exponential convergence: the probability of failure decreases exponentially with the number of steps n. For practical applications, we can calculate how many steps are needed to achieve a desired success probability.

For example, to achieve a success probability of at least $1 - \delta$ for some small $\delta > 0$, we need:

$$1 - e^{-\lambda n} \geq 1 - \delta \tag{8}$$

which gives us:

$$n \geq \frac{\ln(1/\delta)}{\lambda} = \frac{L \cdot \ln(1/\delta)}{p_{\min}^{L}} \tag{9}$$

Therefore, under bounded resources and finite constraints, the probability of finding a valid plan in n steps is at least $1 - e^{-\lambda n}$, providing a formal guarantee of probabilistic completeness for our neural-symbolic planning approach. □

B.5 Time complexity

Proof of Theorem 5.1 (Time Complexity). We analyze the worst-case time complexity of our bidirectional search algorithm for finding a plan of length L with branching factor b in a state space with n states.

Search Space Analysis: In classical forward-only search, the algorithm potentially explores all states reachable within L steps from the initial state w_0. With branching factor b, this yields a search space of size:

$$|S_{\text{forward}}| = \sum_{i=0}^{L} b^i = \frac{b^{L+1} - 1}{b - 1} = O(b^L) \tag{10}$$

Our bidirectional approach simultaneously expands from the initial state w_0 and the goal state w^*. Let's analyze the size of both search frontiers:

1. **Forward Search Frontier:** Starting from w_0, after i expansions, we explore $O(b^i)$ states.
2. **Backward Search Frontier:** Starting from w^*, after j expansions, we explore $O(b^j)$ states.

Meeting Point Analysis: For a plan of length L, the forward and backward frontiers will meet when $i + j \geq L$. The optimal allocation that minimizes the total number of explored states occurs when $i \approx j \approx L/2$.

At this balanced meeting point, the number of states explored by each frontier is:

$$|S_{\text{forward}}| = O(b^{L/2}) \quad \text{and} \quad |S_{\text{backward}}| = O(b^{L/2}) \tag{11}$$

Therefore, the total number of states explored is:

$$\begin{aligned} |S_{\text{total}}| &= |S_{\text{forward}}| + |S_{\text{backward}}| \\ &= O(b^{L/2}) + O(b^{L/2}) = O(b^{L/2}) \end{aligned} \tag{12}$$

Verification Overhead: At each iteration, our algorithm:

1. Selects the most promising state from each frontier using the planning distance function D, which takes $O(\log |F|)$ time with a priority queue, where $|F|$ is the frontier size.
2. Expands the selected state by applying all possible operators, which takes $O(b)$ time.
3. Attempts to find meeting points between the frontiers, which requires checking $O(|F_F| \cdot |F_B|)$ potential state pairs in the worst case, where $|F_F|$ and $|F_B|$ are the sizes of the forward and backward frontiers.
4. Performs pullback verification for promising meeting candidates, which takes $O(d)$ time per candidate, where d is the dimensionality of our state representation.

In the worst case, the frontier sizes grow to $O(b^{L/2})$, making the meeting point search potentially expensive. However, our planning distance function D provides an effective heuristic to limit the number of candidate pairs to consider.

Let k be the number of most promising pairs we consider at each iteration, where k is a constant that depends on the problem domain. The verification overhead per iteration becomes $O(k \cdot d) = O(1)$ for fixed k and d.

Total Complexity: Over the course of the search, we explore $O(b^{L/2})$ states, with each state requiring $O(b)$ time for expansion and $O(1)$ time for verification. Thus, the total time complexity is:

$$T = O(b^{L/2} \cdot b \cdot 1) = O(b^{L/2+1}) = O(b^{L/2}) \tag{13}$$

where the last simplification assumes $b > 1$.

Comparison with Unidirectional Search: The standard unidirectional forward search has time complexity $O(b^L)$. Our bidirectional approach achieves $O(b^{L/2})$, which represents a quadratic improvement in the exponent:

$$\frac{b^L}{b^{L/2}} = b^{L/2} \tag{14}$$

This exponential reduction makes problems with large L tractable in practice. For example, with $b = 3$ and $L = 20$, unidirectional search explores up to $3^{20} \approx 3.5 \times 10^9$ states, while our bidirectional approach explores only up to $3^{10} \approx 59,000$ states—a reduction by a factor of approximately $60,000$.

Therefore, the bidirectional search algorithm has time complexity $O(b^{L/2})$. □

C Implementation Details

Our implementation uses Llama3.1-13B as the backbone LLM model. The model is finetuned on a server with AMD EPYC CPU and a single NVIDIA A100 (80GB) GPU.

Dataset Preparation For finetuning the morphism generator ϕ_f, we construct training examples through negative sampling of valid planning pathways. For each state node w_i in the pathway rooted at w^*, we create positive examples using the ground truth morphisms, and negative examples using invalid or suboptimal morphisms. We assign preference scores based on $V_t(w_i|G)$ values obtained through the bidirectional search methodology described in Section 4.2.

For the planning distance function D, we collect training pairs from both forward and backward search spaces. From each valid pathway to w^*, we extract state pairs and their corresponding labels $V_{w^*}(w_i|G_R) - sn(w_i|G_R)$, generating a dataset that captures both top-down and bottom-up planning distances.

For the value estimator V_w given w_0, which we model as MLP, we extract ground truth minimum cost values from completed search trees, using them as supervision signals.

Distance Function Components The symbolic state distance d_s is implemented as MLP$(h_{s_1} - h_{s_2})$, where h_{s_i} is the embedding of symbolic state s_i generated by the LLM. For logical constraints, we use the Jaccard distance $d_l(l_1, l_2) = 1 - \frac{|l_1 \cap l_2|}{|l_1 \cup l_2|}$. Temporal distance d_t is computed as the summation of active time differences: $d_t(t_1, t_2) = \sum_{i \in \text{active}} |t_1(i) - t_2(i)|$. Based on ablation studies, we set component weights to $\alpha_s = 0.85, \alpha_r = 0.05, \alpha_l = 0.05, \alpha_t = 0.05$.

Model Training We train all MLP components using the Adam optimizer with initial learning rate 0.001 and decay factor 0.3. We employ early stopping with patience 3 to prevent overfitting. Through hyperparameter tuning, we selected dropout ratio 0.2 (from [0.1-0.5]), 3 hidden layers (from [2-4]), and hidden dimensions of 1024 for d_s and 256 for V_w. The morphism generator ϕ_f is finetuned using Direct Preference Optimization (DPO) with the TRL library, training for learning rate 1e-5 with batch size 8, gradient accumulation step of 2, decay 0.1. The result is by default.

D Decomposition Prompt

LLM-Driven Structured Task Decomposition

User Query: Train a language model on Dataset X within 12 hours, ensuring memory usage stays under 16GB.

System Prompt: Parse the given task specification into a formal structured representation with the following schema (return as JSON):

1. **Resources**: $\{r_i\}$ – The set of resources with their types, capacities, and initial states (e.g., computational resources, data assets, model artifacts)
2. **Operators**: $\{O_j\}$ – The set of valid operations where:
   - Unary operators: $O_j : S_i \to S_{i+1}$ (e.g., preprocess, validate)
   - Binary operators: $O_j : S_i \times R_k \to S_{i+1}$ (e.g., train_on, evaluate_with)
3. **Constraints**: $\{C_l\}$ – The set of domain and resource constraints, where:
   - Temporal constraints: $\{t_{min}, t_{max}\}$ for each operation
   - Resource bounds: $\{r_{min}, r_{max}\}$ for each resource consumption
   - Precedence constraints: $\{(O_j, O_k) | O_j \prec O_k\}$

This initial decomposition is then progressively refined through subsequent steps.

Constraint Refinement Step

System Prompt: Identify and clarify any ambiguous or missing constraints in the initial specification:

- Initialization prerequisites
- Resource contention:
- Constraint type:

Resource Formalization Step

System Prompt: Formalize each resource with explicit typing, quantification and format:

- Specific units and measures for each resource
- Minimum/maximum values for each constraint
- Formal temporal expressions

The final categorical encoding step transforms these specifications into mathematical objects, morphisms, and constraints suitable for our category-theoretic framework. This iterative process significantly reduces manual engineering effort typically required for symbolic planning approaches, while ensuring the resulting formalization maintains the precision needed for categorical verification.

Meta-Prompt for Domain Adaptation. To adapt the decomposition framework to a new domain, replace the domain-specific primitives in the `Resources`, `Operators`, and `Constraints` fields with entities relevant to that setting. For example, in cooking, resources become ingredients and appliances, operators are actions such as `chop` or `bake`, and constraints encode nutritional or temporal limits; in robotics, resources map to robots and sensors, operators include `move` or `pick`, and constraints enforce energy, safety, or timing bounds. The schema and output format remain unchanged—the only modification is substituting examples and constraints that capture the new domain's requirements.

D.1 Worked Example: Task Decomposition

We illustrate a full decomposition example with the task:

"Bake cookies with limited sugar for diabetes consideration while still tasting good."

Step 1: Initial Decomposition (via LLM). Extract candidate resources, operators, and constraints.
Resources: flour (2 cups), sugar (0.5 cups), erythritol (1/3 cups), oven, mixing bowl.
Operators:

- O_1: mix(ingredients) $\rightarrow$ dough
- O_2: bake(dough, oven) $\rightarrow$ cookies

Constraints:

- Resource: sugar ≤ 0.1 cups
- Temporal: bake duration $\in [15, 20]$ minutes
- Precedence: mix $\prec$ bake

Step 2: Constraint Refinement. The system identifies implicit assumptions:

- Oven must be preheated before bake.
- Sugar substitution with erythritol is allowed but capped at 1/3 cup.
- Mixing requires all dry ingredients to be available simultaneously.

Step 3: Resource Formalization. Resources are typed and quantified explicitly:
{ "flour": {"type": "ingredient", "quantity": "2c"}, "sugar": {"type": "ingredient", "quantity": "0.5c", "max": "0.1c"}, "erythritol": {"type": "ingredient", "quantity": "1/3c", "max": "1/3c"}, "oven": {"type": "appliance", "state": "preheated"}, "bowl": {"type": "container", "capacity": "5c"} }

E Algorithm

Algorithm 1 Bidirectional Search with Planning Distance

```
Require: Initial state $w_0$, Goal state $w^*$, Planning distance $D$, Budget $B$
    $\mathcal{G}^F \leftarrow \{w_0\}, \mathcal{G}^B \leftarrow \{w^*\}$
    $V(w_0) \leftarrow 0, V(w^*) \leftarrow 0$
    $\mathcal{F}^F \leftarrow \{w_0\}, \mathcal{F}^B \leftarrow \{w^*\}$
    $steps \leftarrow 0$
    while $steps < B$ and $(|\mathcal{F}^F| > 0$ and $|\mathcal{F}^B| > 0)$ do
        $w_{select,F} \leftarrow \arg\min_{w \in \mathcal{F}^F} V_t(w|\mathcal{G}^F)$                ▷ Forward selection
        for each valid morphism $f : w_{select,F} \rightarrow w'$ do
            Add $w'$ to $\mathcal{G}^F$ and $\mathcal{F}^F$ if not already present
            $V(w') \leftarrow \min\{V(w'), V(w_{select,F}) + c(f)\}$
            $sn(w'|\mathcal{G}^F) \leftarrow V_{w'} = D(w', \gamma(w'))$
        end for
        Remove $w_{select,F}$ from $\mathcal{F}^F$
        $w_{select,B} \leftarrow \arg\min_{w \in \mathcal{F}^B}[V_t(w|\mathcal{G}^B) + \min(D_t(w|\mathcal{G}^B))]$        ▷ Backward selection
        for each valid morphism $f : w' \rightarrow w_{select,B}$ do
            Add $w'$ to $\mathcal{G}^B$ and $\mathcal{F}^B$ if not already present
            $V(w') \leftarrow \min\{V(w'), V(w_{select,B}) + c(f)\}$
            $rn(w'|\mathcal{G}^B) \leftarrow V_{w'}$
            $sn(w'|\mathcal{G}^B) \leftarrow \{D_{w'} - V_{w'}\} = \{D(\gamma(w'), w') - V_{w'}\}$
        end for
        Remove $w_{select,B}$ from $\mathcal{F}^B$
        Update $sn$ and $D_t$ values via Uppropagation and Downpropagation for $\mathcal{G}^B$
        Attempt pullback checks between states in $\mathcal{G}^F$ and $\mathcal{G}^B$
        for each $w_F \in \mathcal{G}^F$ and $w_B \in \mathcal{G}^B$ with $D(w_F, w_B) < \epsilon$ do
            if there exist morphisms $f_F : w_F \rightarrow w_C$ and $f_B : w_B \rightarrow w_C$ then
                Attempt to construct pullback $w_P$ with projections $p_1 : w_P \rightarrow w_F, p_2 : w_P \rightarrow w_B$
                if valid pullback $w_P$ exists then
                    $plan \leftarrow$ Compose path from $w_0$ to $w_F$ with path from $w_B$ to $w^*$
                    return $plan$
                end if
            end if
        end for
        Prune dominated states from $\mathcal{G}^F$ and $\mathcal{G}^B$
        $steps \leftarrow steps + 1$
    end while
    return no valid plan found
```

F AI Assistant Usage

This research utilized AI assistants including Claude and GPT-4 for several aspects of the paper and dataset preparation. We employed these tools mainly for:

- Dataset enhancement:GPT-4 was used to augment the RecipeNLG dataset with explicit resource constraints (e.g., "2 cups flour maximum") and temporal intervals (e.g., "bake for 20 minutes") to create a more challenging testing environment for constraint satisfaction. This augmentation process was carefully designed and supervised by the authors to ensure consistency and validity of the constraints.
- Implementation support: AI assistants provided code debugging assistance for the implementation of our validation check and bidirectional search algorithm.
- Manuscript preparation: We used AI assistants for literature review to identify relevant papers, proofreading, language refinement, and formatting assistance.
- Proof check: We used AI assistant to refine and check the draft proofs.
- Benchmark: We use AI assistant GPT-4 as one of our benchmark on the dataset