Verification-Guided Abstraction Generation with Large Language Models for Generalized Planning

Zhenhe Cui[1, 3], Huaxiang Xia[1], Hangjun Shen[1], Kailun Luo[2,*], Yong He[1], Wei Liang[1]
[1]Hunan University of Science and Technology, Xiangtan, China
[2] Dongguan University of Technology, Dongguan, China
[3] Sanya Institute of Hunan University of Science and Technology, Sanya, China

Abstract

Generalized planning (GP), where a single policy solves multiple instances from a common planning domain, remains a challenging problem in AI. Abstraction plays an important role in GP, and Qualitative Numerical Planning (QNP) provides a compact abstract model for GP. However, useful QNP abstractions are difficult to construct and require substantial manual effort and expertise. We study whether QNP abstractions can be generated from PDDL domains and training instances by coupling LLM generation with symbolic reasoning. We propose a generate–debug–repair framework in which an LLM proposes abstract features and constructs a candidate QNP abstraction over the initial states, actions, and goals of the training instances. To improve reliability, we introduce verification-guided repair based on approximate soundness. Given a candidate abstraction, we solve it, validate the induced abstract instances, and apply one of two checking branches: SRCB tests an m-simulation relation between induced abstract and concrete instances to target approximate soundness, while PRCB tests whether the induced abstract plan refines into valid concrete solutions. Detected failures are converted into structured feedback for iterative repair. Experiments on seven GP benchmark domains with five LLMs show that verification-guided repair improves LLM-generated QNP abstractions over both single-pass generation and LLM self-repair, and helps some models produce abstractions that transfer across instances. Our results suggest that LLMs can serve as generators of symbolic abstractions for GP when coupled with formal error detection and structured repair feedback.

1 Introduction

Generalized planning (GP) aims to compute a single policy that solves a family of instances from the same planning domain [1–6]. Its central challenge is to reason beyond individual instances: a generalized policy must capture structural regularities that make many instances solvable, rather than incidental details of any one instance. Abstraction therefore plays a central role in GP. To solve low-level (LL) instance families, one must identify a high-level (HL) representation that preserves the structure relevant to solving all of them. Given such an abstraction, a planner can solve a compact HL problem and map the solution back to the original problem. Thus, a key problem in GP is not only how to solve the abstract problem, but how to obtain abstractions with useful correctness properties.

Qualitative Numerical Planning (QNP) [3], an extension of classical planning with non-negative numerical variables that can be qualitatively increased or decreased, provides a compact abstraction model for GP. Prior work has shown that QNP abstractions are effective for GP [4, 5, 7–9]. For example, Bonet and Geffner [4] showed that GP problems can be solved by abstracting them into QNPs, and that solutions to sound abstractions can be transferred back to the original problem. Cui

*Corresponding author. Email: luokl@dgut.edu.cn

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

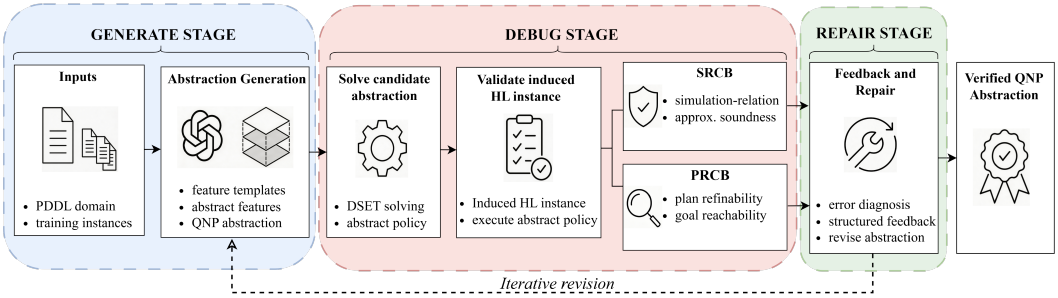

Figure 1: Overview of verification-guided QNP abstraction generation with LLMs for GP.

et al. [6] proposed a QNP abstraction framework for GP based on the situation calculus and Golog [10]. Using refinement mappings to relate QNP problems to GP problems, they provided a formal account of sound QNP abstractions and their connection to general solutions. These results establish QNP as a principled abstraction language for GP, but also reveal a central automation bottleneck. Constructing useful QNP abstractions remains a substantial expert-engineering challenge: existing approaches typically require domain experts to carefully design the abstraction-generation machinery, including the feature space [4, 11], search bias [11, 12], construction procedure [12, 13], refinement assumptions [14], and correctness criteria [9, 15]. Such machinery can be effective, but is difficult to reuse across domains and remains a major obstacle to scaling GP abstraction methods.

Large language models (LLMs) offer a new opportunity to reduce this expert-engineering burden. Given their ability to interpret symbolic specifications and generate structured outputs, LLMs are plausible candidates for proposing QNP abstractions from PDDL domain descriptions and a small set of training instances. Recent work has shown that LLMs can support planning in multiple ways [16–21], including acting as generalized planners [18], and has begun to explore LLM-based abstraction in planning [22]. However, abstraction generation is more demanding than producing a plan for a single instance. A useful abstraction must capture structure shared across instances, and small symbolic errors can make the resulting HL solution non-refinable or unsound. This raises two questions: can pretrained LLMs generate QNP abstractions from domain descriptions and instances, and can formal methods diagnose and repair the errors in such abstractions?

We address these questions with a neuro-symbolic generate–debug–repair framework for QNP abstraction generation. Our goal is not to eliminate prior structure altogether: the framework uses predefined feature templates to delimit a reusable abstraction space. Rather, we shift part of the expert burden from designing domain-specific abstraction-generation algorithms to letting an LLM search within a template-induced abstraction space, while symbolic verification diagnoses and repairs candidate abstractions. As shown in Figure 1, an LLM first proposes abstract features from the predefined templates and constructs a candidate QNP abstraction by abstracting the initial states, action descriptions, and goals of the training instances. The abstraction is then passed to a symbolic verification module. Rather than using verification only as a post hoc evaluator [8], we use it as an interactive source of structured feedback. When verification fails, the detected error is converted into structured feedback and returned to the LLM to guide abstraction repair. The model then revises the abstraction, and the process repeats under a bounded prompting budget.

The debugging procedure is based on approximate abstraction [15]. Given a candidate QNP abstraction and a refinement mapping, the framework checks whether the abstraction supports reliable refinement from HL solutions to LL instances. We instantiate this idea with two complementary branches. The first, *simulation-relation-checking-based debugging* (SRCB), targets a stronger abstraction-level criterion by checking whether an m-simulation relation holds between the induced HL and LL instances. The second, *policy-refinability-checking-based debugging* (PRCB), directly tests solution-level refinability by checking whether the abstract solution induced by the candidate abstraction can be refined into a valid LL plan.

We evaluate the framework on seven GP benchmark domains using five representative LLMs from distinct model families. The results show that verification-guided repair improves the coverage of LLM-generated QNP abstractions over both single-pass generation and LLM self-repair, with the largest gains appearing for stronger base models. Under SRCB, GPT improves from 0.14 with

self-repair to 0.60 with verification-guided repair, while Claude improves from 0.00 to 0.58. Under PRCB, the coverage of GPT increases from 0.21 to 0.60, and that of Claude from 0.31 to 0.59.

In summary, this paper makes the following novel contributions:

- We formulate QNP abstraction generation for GP as a neuro-symbolic generation problem that combines LLM-based abstraction generation with symbolic validation and repair.
- We introduce a verification-guided generate–debug–repair framework that turns formal checking failures into structured feedback for iterative abstraction repair.
- We develop two complementary debugging criteria, SRCB and PRCB, capturing stronger abstraction-level soundness and more direct solution-level refinability, respectively.
- We provide an empirical study across seven GP benchmark domains and five LLMs, showing that verification-guided repair improves abstraction coverage over both single-pass generation and LLM self-repair, and reveals systematic model- and domain-dependent failure modes.

To the best of our knowledge, this is the first framework for LLM-based QNP abstraction generation in GP that couples symbolic abstraction generation with formal verification-guided repair.

2 Preliminaries

Generalized Planning. A GP domain is a tuple $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$, where $\mathcal{P}$ is a set of predicates and $\mathcal{A}$ is a set of action schemata. Each action schema $a(\vec{x}) \in \mathcal{A}$ has the form $a(\vec{x}) = \langle \texttt{pre}(a(\vec{x})), \texttt{eff}(a(\vec{x})) \rangle$, where $\texttt{pre}(a(\vec{x}))$ specifies the conditions that must be satisfied before $a(\vec{x})$ is executed, while $\texttt{eff}(a(\vec{x}))$ defines the effects produced after its execution, both $\texttt{pre}(a(\vec{x}))$ and $\texttt{eff}(a(\vec{x}))$ are sets of first-order literals defined on $\mathcal{P}$[2].

A GP problem is a tuple $\mathcal{Q} = \langle \mathcal{D}, S_0, S_G \rangle$, where $\mathcal{D}$ is a GP domain, and S_0 and S_G are first-order formulas specifying the initial and goal conditions, respectively. A GP instance of $\mathcal{Q}$ is a tuple $Q = \langle \mathcal{D}, O, s_0, s_g \rangle$, where $\mathcal{D}$ is a GP domain, O is a finite set of objects, s_0 and s_g are states satisfying S_0 and S_G, respectively. We identify a state with the set of ground atoms that are true in it. Given an action schema $a(\vec{x})$ and a tuple of objects $\vec{o}$, the corresponding ground action $a(\vec{o})$ is obtained by replacing each variable in $\vec{x}$ with the corresponding object in $\vec{o}$. We say that $a(\vec{o})$ is applicable in a state s if s satisfies the precondition of $a(\vec{o})$. We use the notation $\mu(s, a(\vec{o}))$ to denote the successor state obtained by applying $a(\vec{o})$ in s. A plan for an instance Q is an action sequence $\alpha = a_1, \ldots, a_n$ such that each a_i is applicable in the state reached after executing $a_1, \ldots, a_{i-1}$, and the final state satisfies the goal of Q.

Example 1 We use the Gripper domain [17] to illustrate the definitions, see Figure 2. In this domain, a robot with two grippers must transport balls between rooms until each ball reaches its designated goal room. Some predicates are *carry*, *at-robby*, *at*, and *goal-at*; some actions are *move*, *pick*, and *drop*. A GP problem $\mathcal{Q}_g^l$ in this domain has initial condition $S_0^l = \exists b, r, g.\, \textit{at}(b, r) \wedge \textit{free}(g) \wedge \textit{at-robby}(r)$ and goal condition $S_G^l = \forall b, r.\, \textit{goal-at}(b, r) \rightarrow \textit{at}(b, r)$.

Qualitative Numerical Planning. QNP extends classical planning with boolean and non-negative numerical variables that can be qualitatively increased or decreased [3]. Let $\mathcal{B}$ be a set of propositional variables and $\mathcal{N}$ a set of non-negative numerical variables. Let $\mathcal{L}$ denote the set of all consistent sets of literals of the form p or $\neg p$ for $p \in \mathcal{B}$, and $n = 0$ or $n > 0$ for $n \in \mathcal{N}$. Let $\mathcal{L}_\mathcal{B}$ denote the restriction of $\mathcal{L}$ to literals over $\mathcal{B}$ only. A QNP domain is a tuple $\mathcal{D}_h = \langle \mathcal{B}, \mathcal{N}, \mathcal{A}_h \rangle$, where $\mathcal{A}_h$ is a set of actions. Each action $a \in \mathcal{A}_h$ is associated with a precondition $\texttt{pre}(a) \in \mathcal{L}$, a propositional effect $\mathcal{B}\text{-}\texttt{eff}(a) \in \mathcal{L}_\mathcal{B}$, and a numerical effect $\mathcal{N}\text{-}\texttt{eff}(a)$ consisting of atoms of the form $inc(n)$ or $dec(n)$ for $n \in \mathcal{N}$. Any action containing $dec(n)$ must also require $n > 0$ as a precondition.

A QNP problem is a tuple $\mathcal{Q} = \langle \mathcal{D}, S_0, S_G \rangle$, where $\mathcal{D}$ is a QNP domain, $S_0 \in \mathcal{L}$ denotes the initial condition, and $S_G \in \mathcal{L}$ denotes the goal condition. A *qualitative state* (qstate) $\bar{s}$ of $\mathcal{Q}$ is an element of $\mathcal{L}$ that contains a literal of each variable. An instance of $\mathcal{Q}$ is a *numerical planning* problem $Q = \langle \mathcal{D}, s_0, s_g \rangle$ which replaces S_0 and S_G with states s_0 and s_g satisfying S_0 and S_G, respectively. A *(quantitative) state* of Q is an assignment of non-negative values to all $n \in \mathcal{N}$ and of truth values to

[2]We use the first-order logic with counting [23] as the underlying logic.

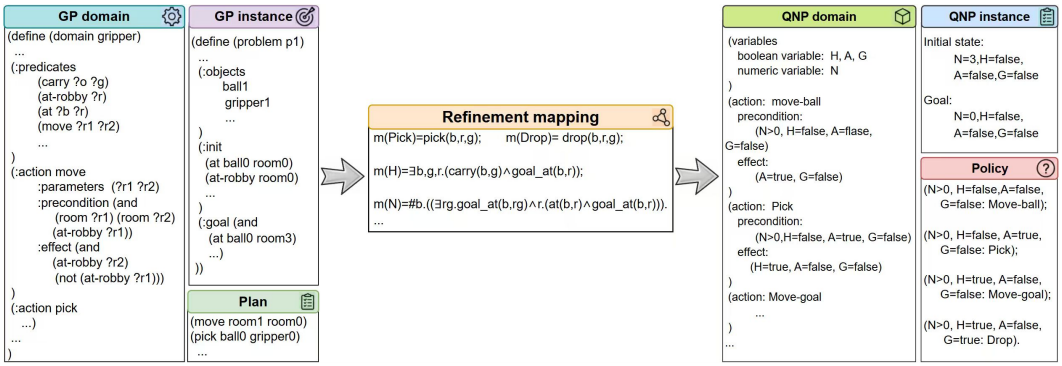

Figure 2: Illustration of abstraction generation for the Gripper domain, showing the HL and LL domains, refinement mapping, LL instance and plan, HL instance, and policy.

$p \in \mathcal{B}$. For a state s and a qstate $\bar{s}$, s satisfies $\bar{s}$ if the assignment of values for variables in s satisfies the literals in $\bar{s}$. For a state s and an applicable action $a \in \mathcal{A}_h$, we write $\eta(s, a)$ for the successor state that results from executing a in state s. A policy π for a QNP problem Q is a partial mapping from qstates into actions. Given a policy π, a π-trajectory is a sequence of states $s_0, s_1, \ldots$ (finite or infinite) such that for all $i \geq 0$, s_{i+1} can result from performing $\pi(\bar{s}_i)$ in s_i, where $\bar{s}_i$ is the qstate satisfied by s_i. A policy π solves an instance of $\mathcal{Q}_h$ if every π-trajectory starting from the initial state reaches a goal state. It solves $\mathcal{Q}_h$ if it solves every instance of $\mathcal{Q}_h$.

Example 2 In Figure 2, the QNP abstraction $\mathcal{Q}_g^h$ uses one numerical variable N for the number of balls not yet transported, and three boolean variables H, A, G describing whether a gripper carries a ball, the robot is at a source room, and the robot is at a target room. Some abstract actions are *Move-Ball*, *Pick*, *Move-Goal*, and *Drop*. The initial state is $S_0 = \{N > 0, \neg H, \neg A, \neg G\}$, and the goal is $S_G = \{N = 0, \neg H, \neg A, \neg G\}$.

An abstraction framework from QNP to GP. Bonet and Geffner [4] proposed a QNP abstraction framework for GP, and we adapt the framework to our setting. We assume that there are two planning domains, one is a QNP domain $\mathcal{D}_h = \langle \mathcal{B}, \mathcal{N}, \mathcal{A}_h \rangle$ and the other is a GP domain $\mathcal{D}_l = \langle \mathcal{P}, \mathcal{A}_l \rangle$. We consider $\mathcal{D}_h$ as a HL domain and $\mathcal{D}_l$ a LL domain. To define the QNP abstractions for GP problems, we first introduce the notion of refinement mapping m to relate expressions between two languages of $\mathcal{D}_h$ and $\mathcal{D}_l$. We say that a function m is a refinement mapping from $\mathcal{D}_h = \langle \mathcal{B}, \mathcal{N}, \mathcal{A}_h \rangle$ to $\mathcal{D}_l = \langle \mathcal{P}, \mathcal{A}_l \rangle$ iff: (1) for each HL propositional variable p, $m(p) = \phi_p$, where ϕ_p is a LL closed FOL formula; (2) for each HL numerical variable n, $m(n) = \tau_n$, where τ_n is a LL counting term; (3) and for each HL action a_h, $m(a_h) = a_l(\vec{x})$, where $a_l(\vec{x}) \in \mathcal{A}_l$. In concrete instances, $m(a_h)$ denotes the LL action schema associated with a_h; its applicable groundings are treated as refinements of a_h. An example refinement mapping m from $\mathcal{Q}_g^h$ to $\mathcal{Q}_g^l$ is shown in Figure 2.

Given a QNP problem $\mathcal{Q}_h$ and a GP problem $\mathcal{Q}_l$, we now establish the correlation between their states under m. Assume that the HL and LL domains share the same variable assignment v. Let s_h be a state of a QNP instance Q_h and s_l a state of a GP instance Q_l. We say that s_h is *m-isomorphic* to s_l, written $s_h \sim_m s_l$, if: (1) for every $p \in \mathcal{B}$, we have $s_h \models p$ iff $s_l \models m(p)$; (2) for every $n \in \mathcal{N}$, we have $s_h \models n = y$ iff $m(n)$ is evaluated to y in s_l. Based on this notion, we then correlate the transitions of Q_h and Q_l under m. Let Δ_Q denote the state set of instance Q. A relation $R \subseteq \Delta_Q^h \times \Delta_Q^l$ is called an *m-simulation* relation between Q_h and Q_l, iff $\langle s_h, s_l \rangle \in R$ implies that: (1) $s_h \sim_m s_l$; (2) for any HL action a, if there is a state s_h' in Q_h, such that $s_h' \in \eta(s_h, a_h)$, then there is a state s_l' and a grounded refinement action a_l of a_h in Q_l, such that $s_l' = \mu(s_l, a_l)$ and $\langle s_h', s_l' \rangle \in R$. We say that Q_h is *m-similar* to Q_l, written $Q_h \precsim_m Q_l$, if there exists an m-simulation relation containing the pair $\langle s_0^h, s_0^l \rangle$.

We say that a QNP problem $\mathcal{Q}_h$ is a *sound abstraction* of a GP problem $\mathcal{Q}_l$ relative to a refinement mapping m if, for every instance Q_l of $\mathcal{Q}_l$, there exists an instance Q_h of $\mathcal{Q}_h$ such that $Q_h \precsim_m Q_l$. When attention is restricted to a finite set $\mathcal{I}$ of GP instances, we correspondingly say that $\mathcal{Q}_h$ is an *approximate sound abstraction* of $\mathcal{Q}_l$ relative to $\mathcal{I}$ if, for every $Q_l \in \mathcal{I}$, there exists an instance Q_h of $\mathcal{Q}_h$ satisfying $Q_h \precsim_m Q_l$. This notion is particularly useful in data-driven settings. Importantly,

Bonet et al. [11] proved that if $\mathcal{Q}_h$ is an approximate sound abstraction of $\mathcal{Q}_l$ relative to a finite instance set $\mathcal{I}$, and π is a solution of $\mathcal{Q}_h$, then the refined policy $m(\pi)$ solves every instance in $\mathcal{I}$.

3 Method

Given a GP domain and a training instance set, our goal is to produce a QNP abstraction. Our framework follows a generate–debug–repair loop: an LLM first generates a candidate QNP abstraction, symbolic checking is then used to diagnose abstraction errors, and any detected failures are translated into structured feedback for iterative repair.

3.1 Abstraction Generation

The abstraction generation stage takes as input a GP domain description and a small set of training instances, and outputs a candidate QNP abstraction. We first ask the LLM to generate abstract features from the GP domain. The second step takes the generated features together with the training instances, and asks the LLM to construct a QNP abstraction. Feature generation is restricted to two predefined templates. Boolean features have the form $p \doteq \exists \vec{x}.\phi(\vec{x})$, where $\phi(\vec{x})$ is a first-order formula over $\mathcal{D}_l$. Numerical features have the form $n \doteq \#\vec{x}.\delta(\vec{x})$, where $\delta(\vec{x})$ is a first-order formula over $\mathcal{D}_l$, and $\#\vec{x}.\delta(\vec{x})$ is a counting term denoting the number of tuples $\vec{x}$ satisfying δ. The resulting boolean and numerical feature sets define the abstract variable sets $\mathcal{B}$ and $\mathcal{N}$, respectively. These features, together with their corresponding formulas or terms, are output as part of the refinement mapping. Given these features and the training instances, the LLM is then asked to construct an abstract initial state S_0, an abstract goal condition S_G, and an abstract action set $\mathcal{A}_h$ from the training instances. The output of this stage is therefore a candidate QNP abstraction. We defer the prompt templates to the appendix.

3.2 Verification-Guided Repair

Given a candidate QNP abstraction $\mathcal{Q}_h$ and a refinement mapping m, we debug the abstraction against the training instances using one of two methods: simulation-relation-checking-based debugging (SRCB) or policy-refinability-checking-based debugging (PRCB). The key insight is that state-action information from concrete instances provides diagnostic evidence for identifying errors in candidate abstractions. The goal is to repair the abstraction so that the resulting abstract solutions can guide search in the concrete instances. The workflows of SRCB and PRCB are described below.

Abstraction Solvability Check (ASC). We first use the solver DSET [24] to solve the QNP abstraction $\mathcal{Q}_h$. If DSET reports that $\mathcal{Q}_h$ has no solution, or fails due to a timeout or an unexpected dead-end, we treat this as evidence that the abstraction itself is flawed and return the solver output as feedback. Otherwise, DSET returns a policy π, and debugging proceeds to the next step.

Definition 1 (Induced high-level instance) Let $\mathcal{Q}_h = \langle \mathcal{D}_h, S_0, S_G \rangle$ be a QNP abstraction, $Q_l = \langle \mathcal{D}_l, O_l, s_0^l, s_g^l \rangle$ an LL instance, and m a refinement mapping. The *HL instance induced by Q_l under* m is defined as $Q_h = \langle \mathcal{D}_h, s_0^h, s_g^h \rangle$, where s_0^h and s_g^h are the quantitative states abstracted from s_0^l and s_g^l via m, or equivalently, the states are m-isomorphic to s_0^l and s_g^l, respectively.

HL Instance Validity Check (HLIVC). This step checks whether each training instance is properly abstracted and whether the policy can be executed on the induced HL instance. For each LL training instance Q_l, we construct the induced HL instance Q_h using the refinement mapping m. We then check whether Q_h is a valid instance of $\mathcal{Q}_h$, namely whether the initial state and goal of Q_h satisfy the initial state and goal of $\mathcal{Q}_h$, respectively. If this fails, then the abstraction does not correctly cover the training instance, and the mismatch is returned as feedback. If the induced HL instance is valid, we execute the policy π on Q_h. If execution aborts before reaching the HL goal, or becomes stuck in an unexpected dead-end, we again return the resulting witness as feedback. Otherwise, we obtain an abstract plan σ_h induced by π, which is then used in the third step.

Simulation Relation Existence Check (SREC). This step checks whether there exists an m-simulation relation between the state space of the induced HL instance and that of the corresponding LL instance. Given an induced HL instance Q_h and the corresponding LL instance Q_l, we synchronously construct state transition diagrams of Q_h and Q_l in a depth-first manner starting from

Algorithm 1: BUILDSIMULATIONRELATION for SREC

Input: LL instance Q_l, HL instance Q_h, initial states s_0^l, s_0^h, refinement mapping m
Output: SUCCESS or Error witness
if $s_0^h \not\sim_m s_0^l$ **then return** Witness(INITMISMATCH, s_0^h, s_0^l)
$V \leftarrow \emptyset$ `// visited buffer for state pairs` $\langle s^h, s^l \rangle$
Function SREC(s^h, s^l)**:**
  if $\langle s^h, s^l \rangle \in V$ **then return** SUCCESS
  $V \leftarrow V \cup \{\langle s^h, s^l \rangle\}$
  foreach $a^h \in \mathsf{app}(s^h)$ **do**
    $s_+^h \leftarrow \eta(s^h, a^h)$
    $C \leftarrow \{a^l \in \mathsf{app}(s^l) \mid a^l$ refines a^h under $m\}$
    simulated $\leftarrow$ False
    $W_{last} \leftarrow$ null `// buffer for the deepest error witness`
    foreach $a^l \in C$ **do**
      $s_+^l \leftarrow \mu(s^l, a^l)$
      if $s_+^h \sim_m s_+^l$ **then**
        $res \leftarrow$ SREC(s_+^h, s_+^l) `// recursive check with backtracking`
        if $res =$ SUCCESS **then**
          simulated $\leftarrow$ True
          break
        else
          $W_{last} \leftarrow res$ `// branch failed, record error and try next` a^l
    if not simulated **then**
      if $W_{last} \neq$ null **then return** W_{last}
      return Witness(TRANSITIONMISMATCH, $s^l, \mathsf{app}(s^l), s^h, a^h, s_+^h$)
  return SUCCESS
return SREC(s_0^h, s_0^l)

their initial states s_0^h and s_0^l. The construction terminates when no newly discovered HL state can be further expanded. During this process, we check whether there exists an m-simulation relation R between them. Since $s_0^h \sim_m s_0^l$, we have $\langle s_0^h, s_0^l \rangle \in R$. Now assume that $\langle s_i^h, s_i^l \rangle \in R$. For every applicable HL action a_i^h in s_i^h and every successor $s_{i+1}^h = \eta(s_i^h, a_i^h)$, SREC checks whether there exists an applicable LL action a_i^l in s_i^l that refines a_i^h under m and, for the successor $s_{i+1}^l = \mu(s_i^l, a_i^l)$, satisfies $s_{i+1}^h \sim_m s_{i+1}^l$. If this condition fails, the algorithm returns a witness consisting of either a mismatched action pair or a mismatched action-state pair.

Proposition 1 Let $\mathcal{I}_{tr}$ be a set of training instances of the GP problem $\mathcal{Q}_l$, and let $\mathcal{Q}_h$ be a candidate QNP abstraction. For each $Q_l \in \mathcal{I}_{tr}$, let Q_h^l denote the HL instance induced by Q_l under the refinement mapping m. If SRCB succeeds on every induced pair (Q_h^l, Q_l), then $Q_h^l \preceq_m Q_l$ for every $Q_l \in \mathcal{I}_{tr}$. Hence, $\mathcal{Q}_h$ is an approximate sound abstraction of $\mathcal{Q}_l$ relative to $\mathcal{I}_{tr}$.

HL Plan Refinability Check (HLPRC). This step checks whether a plan for the induced HL instance can be refined as an executable action sequence for the corresponding LL instance. Given an LL instance Q_l, its induced HL instance Q_h, an HL plan σ_h for Q_h, and the refinement mapping m, we refine σ_h to construct an LL state-transition tree, called a *refined tree*, and use it to identify abstraction errors. The basic idea is to expand the LL initial state s_0^l in a depth-first manner under the guidance of σ_h. Since s_0^h abstracts s_0^l, we have $s_0^h \sim_m s_0^l$. Assume that s_i^h and s_i^l are reachable states satisfying $s_i^h \sim_m s_i^l$. Let a_i^h be the next action prescribed by σ_h in s_i^h. If no applicable LL action a_i^l in s_i^l refines a_i^h, then the algorithm returns the mismatched witness. Otherwise, for each candidate refinement a_i^l, let s_{i+1}^h be the successor of s_i^h along σ_h, and s_{i+1}^l the successor of s_i^l under a_i^l. If $s_{i+1}^h \not\sim_m s_{i+1}^l$, the algorithm records a mismatched witness and continues the search. If

Algorithm 2: BUILDREFINEDTREE for HLPRC

Input: LL instance Q_l, HL instance Q_h, HL plan $\sigma_h = \langle a_0^h, \ldots, a_{k-1}^h \rangle$, refinement mapping m
Output: SUCCESS, error witness, or refined tree T
Initialize T with root r labeled by $\ell(r) = s_0^l$; $W \leftarrow \emptyset$
if $s_0^h \not\sim_m s_0^l$ **then return** Witness(INITMISMATCH, s_0^h, s_0^l)
Function SEARCH(v, s^h, i)**:**
  $s^l \leftarrow \ell(v)$ `// LL state stored at node` v
  if $s^h \models s_g^h$ *and* $s^l \models s_g^l$ **then return** SUCCESS
  if $i = k$ **then return** FAIL
  $a^h \leftarrow \sigma_h[i]$; $s_+^h \leftarrow \eta(s^h, a^h)$
  $C \leftarrow \{a^l \in \mathsf{app}(s^l) \mid a^l \text{ refines } a^h \text{ under } m\}$
  if $C = \emptyset$ **then**
    $W \leftarrow W \cup \{\mathsf{Witness}(\text{ACTIONMISMATCH}, (a^h, \mathsf{app}(s^l)))\}$ `// action error`
    return FAIL
  foreach $a^l \in C$ **do**
    $s_+^l \leftarrow \mu(s^l, a^l)$
    if $s_+^h \not\sim_m s_+^l$ **then**
      $W \leftarrow W \cup \{\mathsf{Witness}(\text{STATEMISMATCH}, (a^h, s_+^h), (a^l, s_+^l))\}$ `// state error`
      continue
    add child v_+ to v with $\ell(v_+) = s_+^l$ and edge label a^l `// expand refined tree`
    if SEARCH($v_+, s_+^h, i+1$) = SUCCESS **then return** SUCCESS
  return FAIL
if SEARCH($r, s_0^h, 0$) = SUCCESS **then return** SUCCESS
if $W \neq \emptyset$ **then return** SelectWitness(W)
return T `// complete refined tree without synchronized success`

$s_{i+1}^h \sim_m s_{i+1}^l$, then the search continues from (s_{i+1}^h, s_{i+1}^l). In particular, if s_{i+1}^h and s_{i+1}^l satisfy the goals s_g^h and s_g^l of Q_h and Q_l, respectively, then the algorithm returns *Success*. Hence, this checking step returns one of three outcomes: *Success*, a structured error witness, or a refined tree T.

LL Goal Reachability Check (LLGRC). This step checks whether the refined LL action sequence can reach the LL goal. Given the refined tree T, we diagnose potential abstraction errors using LL goal-reachability information. Excluding the nodes at depth k, we perform a $(k\text{-}i)$-step goal-reachability check[3] in a bottom-up manner on the nodes at depth i of T. For any node s_i^l at depth i of T, if s_i^l is $(k\text{-}i)$-step goal-reachable and all its successor nodes s_{i+1}^l are $(k\text{-}i\text{-}1)$-step goal-unreachable, then the corresponding LL transition $(s_i^h, a_i^h, s_{i+1}^h)$ is identified as a likely source of abstraction error.

Proposition 2 For every training instance $Q_l \in \mathcal{I}_{tr}$, let Q_h^l denote the HL instance induced by Q_l under the refinement mapping m. If σ_h is a plan for Q_h^l and PRCB succeeds on $(Q_h^l, \sigma_h, m, Q_l)$, then σ_h can be refined into a valid plan for Q_l.

Example structured feedback: Action mismatch in HLPRC	
Verification context	Check whether the HL plan σ_h for Q_h is refinable in Q_l under m.
Failed condition	No applicable LL action refines the HL action a_i^h.
Diagnostic evidence	Current HL action a_i^h; applicable LL actions $\mathsf{app}(s_i^l) = \{a_1^l, \ldots, a_n^l\}$.
Repair instruction	Revise the action a_i^h of the QNP abstraction $\mathcal{Q}_h$ so that it correctly abstracts one applicable LL action from $\{a_1^l, \ldots, a_n^l\}$.

Template-based structured feedback. When a verification check fails, it returns an error witness that identifies both the failed condition and the symbolic information involved, such as an invalid

[3]This check is performed by Fast Downward.

induced HL instance, an action mismatch, a successor-state mismatch, or a failed goal-reachability condition. We render the witness as a repair prompt with a fixed template containing the verification context, diagnostic evidence, and the requested revision. Thus, the LLM receives feedback grounded in formal verification results rather than unconstrained self-reflection.

Debugging Strategy. The debugging loop applies template-based feedback after each failed check. If the abstraction passes the current check, we proceed to the next step; otherwise, the returned witness is rendered into a structured repair prompt and sent back to the LLM. The prompt identifies the failed stage, the symbolic mismatch, and the part of the abstraction to revise. We impose a time limit T on each debugging step; if the LLM's reasoning time exceeds this limit, we treat the response as invalid, reissue the prompt, and continue repair. We also impose a prompting budget N to bound debugging cost. If this budget is exceeded, the process terminates and returns the current abstraction. In the experiments, this loop corresponds to the verification guided repair setting.

SRCB vs. PRCB. SRCB and PRCB target complementary correctness criteria. SRCB checks a stronger abstraction-level property: passing SRCB on an instance set certifies that the candidate abstraction satisfies the corresponding approximate soundness criterion on that set. PRCB, by contrast, does not certify approximate soundness, but directly tests whether an induced HL solution can be refined into a valid LL solution. Consequently, SRCB provides a stronger correctness certificate, while PRCB offers a weaker but more direct solution-level validation.

4 Experiments

Experimental setup. We evaluate our approach on seven GP benchmark domains from the literature [18]: **Delivery**, **Forest**, **Gripper**, **Miconic**, **Ferry**, **Spanner**, and **Heavy**. These domains cover transportation, navigation, ordering constraints, and resource reuse. For each domain, we generate 4 training instances and 10 evaluation instances, each with 10–30 objects. For each instance, we use Z3 [25] to obtain models of the GP problem's initial and goal formulas, used as the instance initial state and goal. The domain description and all instances are provided to the LLM in PDDL form. We compare three generation and repair settings. In single-pass generation (SP), the QNP abstraction is evaluated without repair. In self-repair (SR), the LLM revises the abstraction under the same repair budget as VR, but receives no solver output or verification witness. In verification-guided repair (VR), the LLM revises the abstraction using structured feedback produced by SRCB or PRCB. In each run, SP, SR, and VR start from the same initial abstraction, so SR controls for iterative prompting while VR tests verification-guided feedback. Existing symbolic abstraction methods often assume expert-designed features or domain-specific abstraction machinery and can achieve near-perfect coverage under those assumptions; our goal is instead to reduce such engineering within a reusable template-induced abstraction space. We use the same model version and decoding configuration for all three settings of each LLM family: GPT-5.4 (Extended Thinking), Claude-Opus-4.6 (Extended), Gemini-3.1 (Pro), DeepSeek-V3.2 (Reasoner), and Qwen-3.5 (Max Thinking). For SR and VR, we allow at most $N = 10$ repair iterations, with a time limit of $T = 30$ minutes per interaction. Outputs that cannot be parsed as valid QNP abstractions, exceed the time limit, or contain missing required fields are treated as invalid and reissued until the repair budget is exhausted.

Evaluation metric. Our main evaluation metric is *coverage rate*, defined as the fraction of evaluation instances successfully handled by a generated abstraction. Since we consider two verification criteria, coverage is measured differently in the two settings. Under SRCB, an instance is covered if the abstraction passes the approximate-soundness check. Under PRCB, an instance is covered if a policy computed from the abstraction can be refined into a valid plan for the corresponding concrete instance. Although computed under different criteria, both notions measure the fraction of test instances on which the generated abstraction is usable under the corresponding verification criterion. We report average coverage over 4 runs, with 10 evaluation instances per run. This also explains why the SP and SR results differ between the SRCB and PRCB blocks in Table 1: the same generated abstractions are evaluated under different criteria. Thus, higher coverage indicates broader applicability under the corresponding verification criterion.

Main results. Table 1 reports coverage under VR, SR, and SP. Overall, verification-guided repair improves abstraction quality across most model families and domains, with the largest gains for

Criterion	Domain	GPT			Claude			Gemini			DeepSeek			Qwen		
		VR	SR	SP	VR	SR	SP	VR	SR	SP	VR	SR	SP	VR	SR	SP
SRCB	Gripper	**1.00**	0.00	0.05	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Miconic	**0.20**	0.00	0.00	**0.03**	0.00	0.00	0.00	0.00	0.00	**0.05**	0.00	0.00	0.00	0.00	0.00
	Ferry	**1.00**	0.50	0.25	**1.00**	0.00	0.00	**0.50**	0.05	0.05	**0.20**	0.00	0.00	0.00	0.00	0.00
	Delivery	**1.00**	0.00	0.00	**1.00**	0.00	0.00	**0.50**	0.00	0.00	**0.23**	0.00	0.00	0.00	0.00	0.00
	Heavy	**1.00**	0.50	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	**0.25**	0.00	0.00	**0.25**	0.00	0.00
	Forest	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Spanner	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Average	**0.60**	0.14	0.04	**0.57**	0.00	0.00	**0.14**	0.01	0.01	**0.10**	0.00	0.00	**0.04**	0.00	0.00
PRCB	Gripper	**1.00**	0.00	0.00	**1.00**	0.00	0.00	**0.33**	0.00	0.00	**0.05**	0.00	0.00	0.00	0.00	0.00
	Miconic	**0.20**	0.00	0.00	**0.15**	0.00	0.00	**0.08**	0.00	0.00	**0.05**	0.00	0.00	0.00	0.00	0.00
	Ferry	**1.00**	0.50	0.25	**1.00**	0.15	0.15	**1.00**	0.10	0.10	**0.55**	0.00	0.00	0.00	0.00	0.00
	Delivery	**1.00**	0.00	0.00	**1.00**	1.00	1.00	**0.25**	0.00	0.00	**0.50**	0.00	0.00	0.00	0.00	0.00
	Heavy	**1.00**	1.00	0.75	**1.00**	1.00	1.00	**1.00**	1.00	1.00	**0.50**	0.25	0.25	**0.50**	0.00	0.00
	Forest	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Spanner	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Average	**0.61**	0.21	0.14	**0.59**	0.31	0.31	**0.38**	0.16	0.16	**0.24**	0.04	0.04	**0.07**	0.00	0.00

Table 1: Coverage rate (in [0,1]) of QNP abstractions generated by LLMs on the evaluation set. Results are averaged over 4 runs and 10 evaluation instances per run.

stronger models such as GPT and Claude. The comparison with SR is particularly important: SR uses the same repair budget as VR but receives no verification witness, yet remains substantially weaker for the strongest models. This shows that structured verification feedback provides useful repair information beyond additional LLM revision rounds. Gemini and DeepSeek also benefit, especially under PRCB, while Qwen rarely produces repairable abstractions. At the domain level, repair is most effective on Gripper, Ferry, Delivery, and Heavy, whereas Forest and Spanner remain challenging. These results suggest that verification-guided repair works best when the initial abstraction captures some correct HL structure but contains local errors that can be diagnosed and fixed. Figure 3 compares the verification time of SRCB and PRCB on instances that pass or fail the corresponding check. SRCB is generally more expensive on passing instances, especially in Gripper and Miconic, while PRCB often spends more time before detecting failure. We additionally report run-to-run standard deviations over four independent runs in Table 4. The gains of VR are stable for stronger models: GPT obtains 0.60 ± 0.06 under SRCB and 0.61 ± 0.07 under PRCB, while Claude obtains 0.57 ± 0.01 and 0.59 ± 0.04, respectively.

Comparison with Symbolic Abstraction Methods. Dong et al. [9] proposed an automatic abstraction-generation method for generalized planning domains with baggable types. Their method is sound and complete, meaning that if a domain satisfies their definition of a generalized planning domain with baggable types, the abstraction generated by their method achieves a coverage rate of 1. Among the seven domains considered in our work, their method has a coverage rate of 1 on Gripper, Ferry, and Spanner, while achieving a coverage rate of 0 on the remaining domains. The method of Bonet et al. [15] achieves full coverage on Gripper, Delivery, and Heavy, and a coverage rate of 0.2 on Ferry. Its executions on Miconic, Forest, and Spanner reached the 30-minute time limit, resulting in a coverage rate of 0 in our reproduction. Compared with the two symbolic methods, our approach covers all instances in Ferry and Delivery, achieves nonzero coverage on Miconic, and performs comparably on Gripper and Heavy. However, none of the evaluated methods achieves positive coverage on Forest, and our approach does not cover Spanner.

Domain \ Approach	Dong et al. [9]	Bonet et al. [15]	Ours
Gripper	1	1	1
Miconic	0	0	0.2
Ferry	1	0.2	1
Delivery	0	1	1
Heavy	0	1	1
Forest	0	0	0
Spanner	1	0	0

Table 2: Comparison with symbolic abstraction methods

Error analysis. Table 3 reports the average number of detected errors per run during verification-guided repair. Unlike SR, VR receives verification witnesses during repair, so its failures can be

Criterion	Error Source	GPT	Claude	Gemini	DeepSeek	Qwen
SRCB	ASC	1.82	0.32	0.68	6.61	8.43
	HLIVC	1.25	0.00	0.46	0.96	0.50
	SREC	2.29	3.07	2.64	2.29	0.54
	Timeout	0.00	2.00	0.00	0.00	0.00
	No Answer	0.00	0.00	5.11	0.00	0.00
PRCB	ASC	2.25	0.64	2.64	6.11	7.93
	HLIVC	1.29	0.14	0.46	1.04	0.75
	HLPRC	1.61	2.75	2.61	1.82	0.61
	LLGRC	0.00	0.00	0.00	0.00	0.00
	Timeout	0.00	1.79	0.00	0.00	0.00
	No Answer	0.00	0.00	2.16	0.00	0.00

Table 3: Average number of detected error sources per run during verification-guided repair.

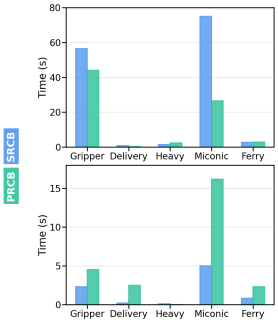

Figure 3: Verification time.

traced to specific verification stages such as ASC, HLIVC, SREC, and HLPRC. Across models and under both verification criteria, ASC is a frequent source of failure, suggesting that abstraction solvability remains a major bottleneck. The dominant criterion-specific errors differ between the two branches: under SRCB, the main additional failures arise from SREC, whereas under PRCB they arise from HLPRC. This indicates that the main remaining difficulty lies either in establishing approximate soundness or in supporting refinable HL policies, depending on the verification criterion used. By contrast, we do not observe LLGRC errors on the tested domains. This does not mean that LLGRC is unnecessary in general. Rather, it indicates that this type of failure is not frequently triggered by the benchmark suite used here. The error table also reveals clear model-dependent differences. GPT and Claude typically produce fewer structural errors than DeepSeek and Qwen, whereas DeepSeek and Qwen accumulate substantially more ASC errors. Gemini exhibits an additional failure mode in the form of `No Answer` outputs, which partly explains its lower coverage under SRCB.

Challenging domains and failure cases. Two domains remain particularly challenging: Spanner and Forest. In Spanner, models often generate an abstract action such as $walk\text{-}to\text{-}spanner$ to represent movement from a location without a spanner to one containing a spanner. However, the refinement of such an action should be a *sequence* of LL actions, e.g., $walk(r0, r1), walk(r1, r2)$, whereas our current formalization assumes that each abstract action is mapped to a single LL action schema. Models also frequently mishandle the one-way predicate $link$ by treating it as symmetric, which leads to unusable abstractions. In Forest, useful abstractions appear to require numerical features such as distances between coordinates in the grid. Such features cannot be expressed in the logical language used in our current implementation. The persistent failure on these domains therefore reflects limitations not only of current LLMs, but also of the abstraction language and refinement assumptions built into our framework. Taken together, these failure cases are informative rather than merely negative. They identify concrete boundaries of the current approach and suggest that future progress will require both stronger abstraction languages and richer refinement models.

5 Discussion and Future Work

This paper investigates whether pretrained LLMs can generate QNP abstractions for GP, and whether formal verification can make such abstractions more reliable. Our results show that raw LLM-generated abstractions are often flawed, and that LLM self-repair alone is not sufficient to fix them reliably. Verification-guided repair provides stronger improvements, suggesting that localized formal feedback is important for abstraction repair. The framework also reveals a clear distinction between the stronger approximate-soundness criterion enforced by SRCB and the weaker plan-level refinability criterion checked by PRCB. The current framework has two main limitations. First, its abstraction language is constrained by predefined feature templates, which may fail to express richer numerical, geometric, or temporally extended abstractions, as seen in Forest and Spanner. Second, repair depends on the precision of solver feedback; when DSET fails to solve a candidate abstraction, the feedback may be too coarse to identify the exact symbolic cause. These results show that combining LLM-based abstraction proposal with formal verification yields a practical route to generating reusable planning abstractions. Future work should explore more expressive abstraction models and richer refinement mappings.

Acknowledgments

We thank anonymous reviewers for their valuable comments. This research was supported by National Natural Science Foundation of China (Nos. 62406108, 62206055, and U2468205), Hainan Provincial Natural Science Foundation of China (No. 625QN362), and Scientific Research Fund of Hunan Provincial Education Department (No. 23B0471).

References

[1] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):615–647, 2011.

[2] Javier Segovia Aguas, Jiménez Sergio, and Anders Jonsson. Generalized planning with procedural domain control knowledge. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS*, pages 285–293, 2016.

[3] Siddharth Srivastava, Shlomo Zilberstein, Neil Immerman, and Hector Geffner. Qualitative numeric planning. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, pages 1010–1016, 2011.

[4] Blai Bonet and Hector Geffner. Features, projections, and representation change for generalized planning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI*, pages 4667–4673, 2018.

[5] León Illanes and Sheila A. McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI*, pages 7610–7618, 2019.

[6] Zhenhe Cui, Yongmei Liu, and Kailun Luo. A uniform abstraction framework for generalized planning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1837–1844, 2021.

[7] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*, pages 1048–1055, 2017.

[8] Zhenhe Cui, Weidu Kuang, and Yongmei Liu. Automatic verification for soundness of bounded QNP abstractions for generalized planning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI*, pages 3149–3157, 2023. doi: 10.24963/IJCAI.2023/351. URL `https://doi.org/10.24963/ijcai.2023/351`.

[9] Hao Dong, Zheyuan Shi, Hemeng Zeng, and Yongmei Liu. An automatic sound and complete abstraction method for generalized planning with baggable types. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 14875–14884, 2025.

[10] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, 2001.

[11] Blai Bonet, Guillem Francès, and Hector Geffner. Learning features and abstract actions for computing generalized plans. In *Proceedings of the AAAI Conference on Artificial Intelligence, (AAAI)*, pages 2703–2710, 2019.

[12] Siddharth Srivastava, Neil Immerman, Shlomo Zilberstein, and Tianjiao Zhang. Directed search for generalized plans using classical planners. In *Proceedings of the International Conference on Automated Planning and Scheduling, (ICAPS)*, pages 226–233, 2011.

[13] Damir Lotinac, Javier Segovia Aguas, Sergio Jiménez Celorrio, and Anders Jonsson. Automatic generation of high-level state features for generalized planning. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3199–3205. IJCAI/AAAI Press, 2016. URL `http://www.ijcai.org/Abstract/16/453`.

[14] Liangda Fang, Xiaoman Wang, Zhang Chen, Kailun Luo, Zhenhe Cui, and Quanlong Guan. A syntactic approach to computing complete and sound abstraction in the situation calculus. In *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pages 14911–14921, 2025.

[15] Blai Bonet, Raquel Fuentetaja, Yolanda E-Martín, and Daniel Borrajo. Guarantees for sound abstractions for generalized planning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1566–1573, 2019.

[16] Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models - a critical investigation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 75993–76005. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/efb2072a358cefb75886a315a6fcf880-Paper-Conference.pdf.

[17] Sadegh Mahdavi, Raquel Aoki, Keyi Tang, and Yanshuai Cao. Leveraging environment interaction for automated pddl translation and planning with large language models. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 38960–39008. Curran Associates, Inc., 2024. doi: 10.52202/079017-1230. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/44af065477781e7f8a8589b14a62c489-Paper-Conference.pdf.

[18] Tom Silver, Soham Dan, Kavitha Srinivas, Joshua B. Tenenbaum, Leslie Pack Kaelbling, and Michael Katz. Generalized planning in PDDL domains with pretrained large language models. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan, editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2024, February 20-27, 2024, Vancouver, Canada*, pages 20256–20264. AAAI Press, 2024. doi: 10.1609/AAAI.V38I18.30006. URL https://doi.org/10.1609/aaai.v38i18.30006.

[19] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=4O0v4s3IzY.

[20] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato, editors, *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, Proceedings of Machine Learning Research, pages 9118–9147. PMLR, 2022. URL https://proceedings.mlr.press/v162/huang22a.html.

[21] Weiyu Liu, Geng Chen, Joy Hsu, Jiayuan Mao, and Jiajun Wu. Learning planning abstractions from language. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=3UWuFoksGb.

[22] Bita Banihashemi, Megh Patel, and Yves Lespérance. Using large language models for abstraction of planning domains-extended version. *arXiv preprint arXiv:2510.20258*, 2025.

[23] Dietrich Kuske and Nicole Schweikardt. First-order logic with counting. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12, 2017.

[24] Hemeng Zeng, Yikun Liang, and Yongmei Liu. A native qualitative numeric planning solver based on AND/OR graph search. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI*, pages 4693–4700, 2022.

[25] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of the Fourteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 337–340, 2008.

A Supplementary Experimental Results

This appendix provides additional run-to-run stability analysis.

Criterion	Domain	GPT	Claude	Gemini	DeepSeek	Qwen
SRCB	Gripper	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	Miconic	0.20 ± 0.40	0.03 ± 0.05	0.00 ± 0.00	0.05 ± 0.06	0.00 ± 0.00
	Ferry	1.00 ± 0.00	1.00 ± 0.00	0.50 ± 0.58	0.20 ± 0.24	0.00 ± 0.00
	Delivery	1.00 ± 0.00	1.00 ± 0.00	0.50 ± 0.58	0.22 ± 0.29	0.00 ± 0.00
	Heavy	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	0.25 ± 0.50	0.25 ± 0.50
	Forest	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	Spanner	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	Average	$\mathbf{0.60 \pm 0.06}$	$\mathbf{0.57 \pm 0.01}$	$\mathbf{0.14 \pm 0.00}$	$\mathbf{0.10 \pm 0.06}$	$\mathbf{0.04 \pm 0.07}$
PRCB	Gripper	1.00 ± 0.00	1.00 ± 0.00	0.33 ± 0.47	0.05 ± 0.10	0.00 ± 0.00
	Miconic	0.20 ± 0.40	0.15 ± 0.30	0.07 ± 0.15	0.05 ± 0.06	0.00 ± 0.00
	Ferry	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.55 ± 0.53	0.00 ± 0.00
	Delivery	1.00 ± 0.00	1.00 ± 0.00	0.25 ± 0.50	0.50 ± 0.58	0.00 ± 0.00
	Heavy	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.50 ± 0.58	0.50 ± 0.58
	Forest	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	Spanner	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	Average	$\mathbf{0.61 \pm 0.07}$	$\mathbf{0.59 \pm 0.04}$	$\mathbf{0.38 \pm 0.16}$	$\mathbf{0.24 \pm 0.17}$	$\mathbf{0.07 \pm 0.08}$

Table 4: Run-to-run stability of verification-guided repair. Each entry reports mean $\pm$ sample standard deviation over four independent runs for the VR setting.

B Prompt Templates

This appendix presents the prompt templates used for feature generation, abstraction generation, self-repair, and verification-guided repair.

B.1 Feature Generation Prompt

We use the following prompt to generate candidate boolean and numerical features.

`Input:` GP domain $\mathcal{D}_l$

`Task:`

- `Generate boolean and numerical features for constructing a QNP abstraction of` $\mathcal{D}_l$.

`Feature templates:`

- `Boolean feature:` $p \doteq \exists \vec{x}.\phi(\vec{x})$.
- `Numerical feature:` $n \doteq \#\vec{x}.\delta(\vec{x})$.

`Here,` $\phi(\vec{x})$ `and` $\delta(\vec{x})$ `are first-order formulas over predicates in` $\mathcal{D}_l$, `and` $\#\vec{x}.\delta(\vec{x})$ `denotes the number of tuples satisfying` δ.

`Output:`

- `Return the boolean feature set` $\mathcal{B}$ `and numerical feature set` $\mathcal{N}$, `together with their defining formulas or counting terms.`

B.2 QNP Abstraction Generation Prompt

We use the following prompt to generate a candidate QNP abstraction from the generated features and training instances.

Input: GP domain $\mathcal{D}_l$, training set $\{Q_l^1, \ldots, Q_l^n\}$, boolean feature set $\mathcal{B}$, numerical feature set $\mathcal{N}$

Task:

- Generate a QNP abstraction for the training instances using the given features.

Steps:

- Compute the abstract initial condition S_0 from the initial states of $\{Q_l^1, \ldots, Q_l^n\}$.
- Compute the abstract goal condition S_G from the goals of $\{Q_l^1, \ldots, Q_l^n\}$.
- Construct the abstract action set $\mathcal{A}_h$ based on the action set $\mathcal{A}_l$ of domain $\mathcal{D}_l$.
- Specify the refinement mapping m for the abstract features and actions.

Output:

- Return the QNP abstraction $\mathcal{Q} = \langle \mathcal{D}_h, S_0, S_G \rangle$, where $\mathcal{D}_h = \langle \mathcal{B}, \mathcal{N}, \mathcal{A}_h \rangle$, and the refinement mapping m.

B.3 Self-Repair Prompt

The self-repair baseline uses the same initial abstraction, training instances, and repair budget as VR, but does not receive any solver output, verification witness, or structured diagnostic feedback. At each repair iteration, the LLM receives its current QNP abstraction, the GP domain, and the training instances, and is asked to revise the abstraction by self-assessment.

Input: GP domain $\mathcal{D}_l$, training set $\{Q_l^1, \ldots, Q_l^n\}$, current QNP abstraction $\mathcal{Q}$

Task:

- Review the QNP abstraction $\mathcal{Q}$ and check whether it correctly captures the structure of the training instances.

Check:

- Check the abstract features, initial state, goal condition, abstract actions, and refinement mapping.

Output:

- If you find any errors or missing components, revise the QNP abstraction. Otherwise, return the abstraction unchanged.

B.4 Verification Feedback Templates

The following templates are used to convert verification failures into structured repair prompts for VR.

ASC feedback template. When ASC fails, we instantiate the following repair prompt and return it to the LLM.

You are given a candidate QNP abstraction $\mathcal{Q}$ generated from the training instances. The abstraction has failed the Abstraction Solvability Check (ASC).

Verification context:

- Check whether the candidate QNP abstraction $\mathcal{Q}$ can be solved by the QNP solver.

Failed condition:

- The solver fails to return a solution for $\mathcal{Q}$.

Diagnostic evidence:

- Candidate QNP abstraction: <candidate-qnp-abstraction>
- Solver status: <solver-status>
- Solver message: <solver-message>

Repair instruction:

- Revise the QNP abstraction $\mathcal{Q}$ so that it admits a valid QNP solution. In particular, check whether the abstract initial state, goal condition, action preconditions, propositional effects, numerical effects, and refinement mapping are mutually consistent. Return the revised QNP abstraction and refinement mapping in the required output format.

HLIVC feedback template: policy execution failure. When policy execution on an induced HL instance fails, we instantiate the following repair prompt and return it to the LLM.

You are given a candidate QNP abstraction $\mathcal{Q}$, an LL training instance Q_l, the induced HL instance Q_h, and a policy π computed from $\mathcal{Q}$. The policy execution has failed during the HL Instance Validity Check (HLIVC).

Verification context:

- Check whether the policy π solves the induced HL instance Q_h constructed from Q_l under m.

Failed condition:

- The execution of the policy π aborts before reaching the goal of Q_h.

Diagnostic evidence:

- Candidate QNP abstraction: <candidate-qnp-abstraction>
- LL instance: <ll-instance>
- Induced HL instance: <induced-hl-instance>
- Policy: <policy>
- Aborted execution: <aborted-execution>

Repair instruction:

- Revise the QNP abstraction $\mathcal{Q}$ so that its policy can be executed on the induced HL instance Q_h until the goal is reached. In particular, check whether the abstract initial state, goal condition, action preconditions, propositional effects, numerical effects, and refinement mapping correctly support the policy execution. Return the revised QNP abstraction and refinement mapping in the required output format.

SREC feedback template: action mismatch. When no applicable LL action refines the current HL action during SREC, we instantiate the following repair prompt and return it to the LLM.

You are given a candidate QNP abstraction $\mathcal{Q}$, an induced HL instance Q_h, the corresponding LL training instance Q_l, and a refinement mapping m. The abstraction has failed the Simulation Relation Existence Check (SREC) due to an action mismatch.

Verification context:

- Check whether the state-transition diagram of Q_h can be simulated by that of Q_l under refinement mapping m.

Failed condition:

- No applicable LL action refines the HL action a_h during the simulation check.

Diagnostic evidence:

- Current HL action: <current-hl-action>
- Applicable LL actions: <applicable-ll-actions>

Repair instruction:

- Revise the action a_h of the QNP abstraction $\mathcal{Q}$ so that it correctly abstracts one applicable LL action from <applicable-ll-actions>. Return the revised QNP abstraction and refinement mapping in the required output format.

HLPRC feedback template: action mismatch. When no applicable LL action refines the current HL action during HLPRC, we instantiate the following repair prompt and return it to the LLM.

You are given a candidate QNP abstraction $\mathcal{Q}$, an induced HL instance Q_h, the corresponding LL training instance Q_l, an HL plan σ_h, and a refinement mapping m. The abstraction has failed the HL Plan Refinability Check (HLPRC) due to an action mismatch.

Verification context:

- Check whether the HL plan σ_h for Q_h is refinable in Q_l under m.

Failed condition:

- No applicable LL action refines the HL action a_i^h.

Diagnostic evidence:

- Current HL action: <current-hl-action>
- Applicable LL actions: <applicable-ll-actions>

Repair instruction:

- Revise the action a_i^h of the QNP abstraction $\mathcal{Q}$ so that it correctly abstracts one applicable LL action from <applicable-ll-actions>. Return the revised QNP abstraction and refinement mapping in the required output format.

LLGRC feedback template: goal-reachability failure. When the refined LL execution fails the goal-reachability check, we instantiate the following repair prompt and return it to the LLM.

You are given a candidate QNP abstraction $\mathcal{Q}$, an induced HL instance Q_h, the corresponding LL training instance Q_l, an HL plan σ_h, and its refined LL action sequence σ_l. The refined LL execution has failed the LL Goal Reachability Check (LLGRC).

Verification context:

- Check whether the LL action sequence σ_l refined from the HL plan σ_h reaches the goal of Q_l.

Failed condition:

- The refined LL execution fails the goal-reachability check for Q_l.

Diagnostic evidence:

- HL action-state pair in Q_h: <hl-action-state-pair>
- Corresponding LL action-state pair in Q_l: <ll-action-state-pair>

Repair instruction:

- Revise the corresponding QNP action so that the abstract transition (s_h, a_h) correctly abstracts an LL transition that preserves goal reachability. Return the revised QNP abstraction and refinement mapping in the required output format.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the paper's scope as LLM-based QNP abstraction generation with verification-guided repair for GP. The claims are supported by the SRCB/PRCB framework and the experiments on seven benchmark domains and five LLMs.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper discusses limitations in the challenging-domain analysis and Discussion, including failures on Forest and Spanner, the limited expressiveness of the predefined feature templates, the single-action refinement assumption, and the dependence on solver feedback quality.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: The theoretical statements are given as Proposition 1 and Proposition 2, with their assumptions stated in terms of induced HL instances, refinement mappings, and the success of SRCB or PRCB. These propositions follow directly from the definitions of approximate soundness, m-simulation, and plan refinability introduced in the preliminaries and method sections.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The experimental setup specifies the benchmark domains, instance generation procedure, training/evaluation split, evaluated LLM families, repair budget, timeout, verification criteria, and baselines. The appendix provides the supplementary algorithms, prompt templates, and run-to-run stability results needed to reproduce or verify the main experimental claims.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We will provide an anonymized supplementary package containing the PDDL domains and instances, prompt templates, generated abstractions, verification scripts, and instructions for reproducing the main tables. The code and data will be released in an anonymized form for review and made public upon acceptance.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 specifies the benchmark domains, number of training and evaluation instances, object ranges, LLM variants, repair budget, timeout, and evaluation criteria. The appendix further provides prompt templates and additional stability results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: The main table reports average coverage over four runs, and the appendix reports run-to-run sample standard deviations for verification-guided repair. The text specifies that the standard deviations are computed over independent runs and are used to characterize LLM-induced variability.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The paper reports the per-interaction timeout and verification-time measurements, but does not fully specify the hardware, memory, or total compute required to reproduce all experiments. We will add these details in the supplementary material or final version.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conforms to the NeurIPS Code of Ethics. It uses synthetic planning benchmarks and LLM evaluations, does not involve human subjects or private data, and preserves submission anonymity.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The work is foundational research on generalized planning and symbolic abstraction. Potential positive impacts include reducing manual engineering in planning systems, while potential risks include incorrect abstractions if deployed without verification; the proposed framework mitigates this by using formal checking and structured repair feedback.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release a new high-risk model, scraped dataset, or deployable system with direct misuse potential. It evaluates existing LLMs through their standard interfaces on synthetic planning benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [No]

    Justification: The paper cites the existing benchmark domains, solvers, and LLMs used in the experiments, but does not explicitly list all licenses and terms of use. We will add this information in the supplementary material or final version.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new datasets, models, or software assets as part of the submission. The generated instances and abstractions are used only for the experiments reported in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, user studies, or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve human subjects, personal data, or crowdsourced data collection, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLM usage is central to the proposed method and is described throughout the paper. The method section explains how LLMs generate features and candidate QNP abstractions, and the experiments specify the evaluated LLM families and repair settings.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.