From Solver Feedback to Faithful Plans: Multi-Role Reinforcement Learning for Symbolic Planning

Chenghao Zhang[1] Yikai Mao[1] Shanqi Liu[2] Haoyu Gao[3] SaiSai Hu[4] Dan Roth[1]

Abstract

Reliable planning requires converting natural-language instructions into executable symbolic specifications, yet large language models remain brittle without costly PDDL annotations and may exploit solver success in semantically unfaithful ways. We study how to learn faithful natural-language-to-PDDL formalization using only solver feedback, without human-written demonstrations. We propose a solver-grounded multi-role reinforcement learning framework where a single language model acts as an Actor, Judge, and Editor for generation, verification, and repair. The Actor proposes PDDL specifications, the Judge provides a solver-calibrated quality signal, and the Editor performs bounded diagnostic-conditioned refinement. On PlanBench, our method improves average success from 35.5% for LLM+P to 70.8%, achieves 66.3% faithful success, and reduces semantic drift to 6.4%. These results show that organizing solver feedback into generation, verification, and repair roles enables more scalable and faithful annotation-free symbolic planning.

1 Introduction

Structured planning is a central capability for intelligent agents that must transform high-level instructions into executable action models under constraints. It is critical for robotics, workflow automation, logistics, and interactive decision-making, where success depends not only on producing plausible language but also on satisfying explicit state-transition semantics. Large language models (LLMs) have shown impressive general-purpose reasoning and instruction-following abilities (Brown et al., 2020; OpenAI, 2024; Touvron et al., 2023; Lin et al., 2026), yet reliable planning remains difficult because valid plans require logical consistency, long-horizon state tracking, and faithful grounding in task constraints.

Recent benchmarks have made this limitation increasingly clear. PlanBench shows that LLMs often fail on classical planning domains, especially when predicate names are obfuscated and surface lexical cues are removed (Valmeekam et al., 2023). This supports the broader view that LLMs often behave as approximate retrievers rather than systematic planners over state transitions (Kambhampati et al., 2024). Even reasoning-enhanced models and chain-of-thought prompting remain brittle under planning-specific perturbations, suggesting that fluent intermediate reasoning is not equivalent to executable planning (Stechly et al., 2024; Valmeekam et al., 2024). NATURAL PLAN further shows that realistic natural-language planning becomes sharply harder as constraint complexity increases, and self-correction does not reliably recover valid solutions (Zheng et al., 2024).

A promising response is to use LLMs not as direct planners, but as formalizers that translate natural-language tasks into symbolic representations such as Planning Domain Definition Language (PDDL), after which an external solver performs the actual planning. LLM+P demonstrates the value of this neuro-symbolic decomposition by delegating search to a classical planner (Liu et al., 2023), and subsequent work explores PDDL goal translation, generalized planning programs, and environment-aided PDDL construction (Xie et al., 2023; Silver et al., 2023; Mahdavi et al., 2024). However, these approaches still face a fundamental supervision bottleneck: high-quality natural-language-to-PDDL formalization typically requires curated demonstrations, fixed domain assumptions, or supervised initialization. Instruction-tuning methods such as PDDL-Instruct improve symbolic planning performance, but their reliance on annotated examples limits scalability to new domains and language distributions (Verma et al., 2025).

The deeper challenge is that solver feedback alone is not automatically a faithful learning signal. A PDDL solver can verify whether a generated specification is syntactically valid and executable, but solver success is defined with respect to the generated formal specification rather than the original natural-language

[1] University of Pennsylvania.
[2] Unaffiliated.
[3] Georgia Institute of Technology.
[4] Pace University.

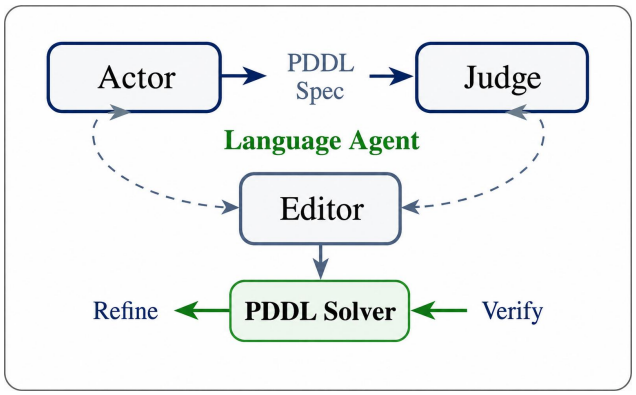

Figure 1: One Brain Three Roles framework. A single LLM plays Actor, Judge, and Editor roles with shared parameters, grounded by deterministic PDDL solver feedback.

intent. Thus, optimizing only for solver success can reward underspecified or semantically shifted formalizations that are easy to solve but no longer faithful to the task. This creates a high-level conflict between *annotation-free learning* and *semantic faithfulness*: the former demands replacing human labels with environment feedback, while the latter requires preventing the model from exploiting imperfections or ambiguities in that feedback. Similar concerns arise in broader reinforcement learning settings, where optimizing a proxy reward can induce reward gaming when the reward does not fully capture the intended objective (Skalse et al., 2022).

This paper asks: *Can an LLM learn reliable symbolic planning from solver feedback alone, without any human-annotated PDDL demonstrations?* We address this question by treating the symbolic solver as the only grounded oracle while organizing learning around generation, verification, and repair. At a high level, the model learns not only to produce a candidate formal specification, but also to evaluate solver-grounded correctness and refine failed specifications using executable diagnostics, thereby converting sparse solver outcomes into a more informative training process.

We propose **Solver-Grounded Multi-Role Reinforcement Learning**, a zero-annotation framework for natural-language-to-PDDL planning. A single LLM is conditioned to play three coordinated roles: an Actor that proposes formal specifications, a Judge that learns a solver-calibrated quality signal, and an Editor that performs bounded repair from solver diagnostics. This design preserves the scalability of solver-only supervision while reducing the risk that optimization collapses into purely solver-facing shortcuts.

Our contributions are as follows:

- We identify a key obstacle in annotation-free symbolic planning: raw solver success is a necessary but insufficient learning signal because it can diverge from semantic faithfulness to the original natural-language task. This reframes NL-to-PDDL learning as a problem of solver-grounded optimization under potential specification drift.
- We introduce a multi-role reinforcement learning framework in which generation, verification, and repair are learned jointly from deterministic solver feedback, without human-annotated PDDL demonstrations. The shared-role design enables verifier co-evolution and diagnostic-conditioned refinement while keeping most model parameters coupled across roles.
- Across PlanBench domains, our method achieves strong in-domain planning success, including substantial gains on obfuscated and structurally challenging settings. It also transfers zero-shot to ProntoQA and NATURAL PLAN, indicating that solver-grounded multi-role learning improves not only benchmark-specific PDDL synthesis but also broader planning and constraint-satisfaction behavior.

2 Related Work

LLMs as planners. A first line asks whether scaling and prompting alone are sufficient for planning. Chain-of-thought and search-based deliberation can improve intermediate reasoning (Wei et al., 2022; Yao et al., 2023; Hao et al., 2023), but planning benchmarks show that fluent rationales do not guarantee executable state-transition reasoning. PlanBench and follow-up analyses reveal large gaps between textual plausibility and valid planning behavior, especially under predicate obfuscation, longer horizons, and reasoning-model variants (Valmeekam et al., 2023; Kambhampati et al., 2024; Stechly et al., 2024;

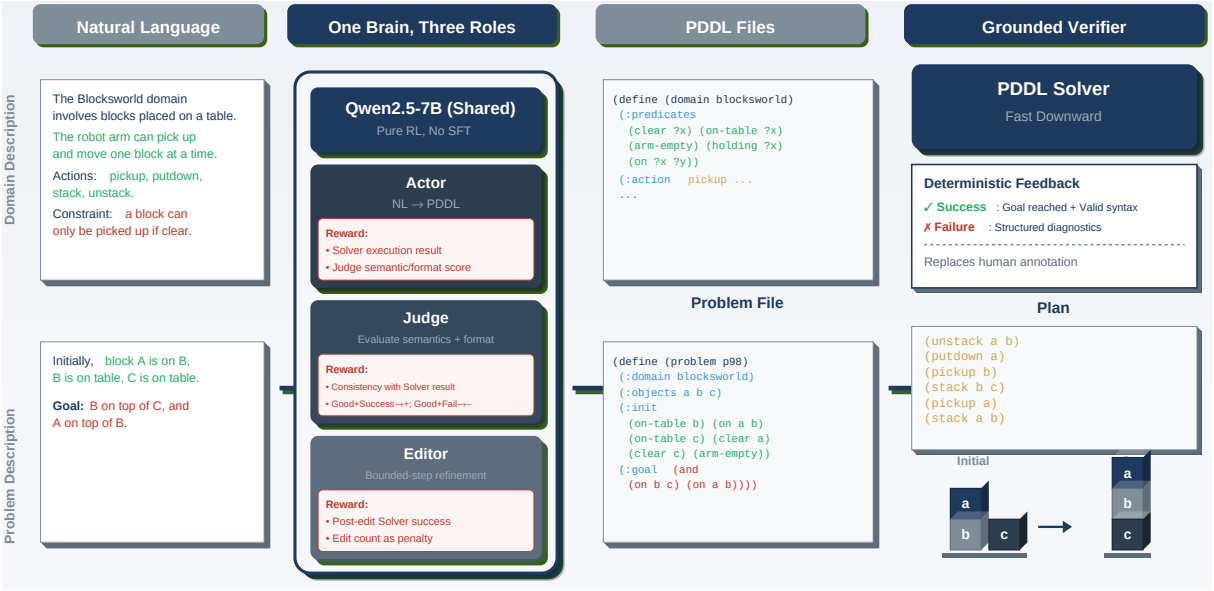

Figure 2: **Solver-Grounded Multi-Role RL for NL-to-PDDL Planning.** A single shared LLM backbone is conditioned into Actor, Judge, and Editor roles. The Actor generates an initial PDDL specification. A deterministic PDDL Solver verifies executability, producing a success label and diagnostics on failure. The Judge predicts a calibrated solver score, and the Editor uses diagnostics to perform bounded local edits, feeding back to the solver.

Valmeekam et al., 2024). NATURAL PLAN further shows that realistic language planning degrades sharply as constraints become more complex (Zheng et al., 2024). These works motivate execution-grounded evaluation, whereas we use solver feedback as the training signal, optimizing planning competence through repeated interaction with a symbolic verifier.

LLM-as-formalizer and neuro-symbolic planning. A second line shifts from direct plan generation to inducing formal representations that planners can solve. LLM+P demonstrates this paradigm by translating natural language into PDDL, invoking a classical planner, and verbalizing the resulting plan (Liu et al., 2023); subsequent work studies goal translation, generalized planning programs, domain generation, and text-to-PDDL evaluation (Xie et al., 2023; Silver et al., 2023; Oswald et al., 2024; Zuo et al., 2025; Zhang et al., 2024). Recent studies show that LLM-as-formalizer can outperform direct planning, but still suffers from semantic omissions, naturalness shift, and scaling failures in large formal structures (Huang and Zhang, 2025; Jiang et al., 2026). Supervised methods such as PDDL-Instruct improve performance with annotated instruction tuning (Verma et al., 2025), while environment-interaction methods refine PDDL through feedback (Mahdavi et al., 2024). In contrast, we formulate NL-to-PDDL induction as annotation-free reinforcement learning, where an Actor generates specifications, a Judge learns solver-calibrated scoring, and an Editor performs diagnostic-conditioned repair.

Feedback-driven self-improvement and verifiable rewards. A third line explores feedback-driven LLM self-improvement without dense human supervision. Reflexion, Self-Refine, ISR-LLM, and AdaPlanner use critique–revise or environment-feedback loops (Shinn et al., 2023; Madaan et al., 2023; Zhou et al., 2023; Sun et al., 2023), while LLM-as-a-judge methods reduce annotation cost but risk evaluator bias and reward hacking without external grounding (Gu et al., 2024; Li et al., 2025). Recent RLVR and self-play systems show that verifiable feedback can scale reasoning training (Shao et al., 2024; Guo et al., 2025; Zhao et al., 2025; Chen et al., 2025), but they mainly target curated tasks or code/math verifiers and do not address the gap between solver success and semantic faithfulness. Our work brings verifiable-reward learning to symbolic planning by grounding Actor, Judge, and Editor roles in a deterministic PDDL solver, converting sparse executability feedback into calibrated verification and repair signals.

3 Method

3.1 Task and Solver-Grounded Environment

We study natural-language-to-PDDL specification induction without human annotations. Each task is a natural-language planning description $x \sim \mathcal{P}_{\text{data}}$. The model outputs a PDDL specification $y = (y^{\text{dom}}, y^{\text{prob}})$, consisting of a domain file and a problem file. A deterministic PDDL environment $\mathcal{E}$,

Algorithm 1 One solver-grounded multi-role training episode

Require: Task x, solver $\mathcal{E}$, maximum repair horizon $H_{\max}$
Sample initial specification $y_0 \sim \pi_\theta(\cdot \mid x, \mathrm{A})$
Query solver $(b_0, d_0) = \mathcal{E}(y_0)$
Compute Judge score $s_J(x, y_0)$
Set $T \leftarrow 0$, $y_T \leftarrow y_0$, $b_T \leftarrow b_0$, $d_T \leftarrow d_0$
for $t = 0, \ldots, H_{\max} - 1$ **do**
  if $b_T = 1$ **then**
    break
  end if
  Sample repaired specification $y_{t+1} \sim \pi_\theta(\cdot \mid x, y_t, d_t, \mathrm{E})$
  Query solver $(b_{t+1}, d_{t+1}) = \mathcal{E}(y_{t+1})$
  Set $T \leftarrow t + 1$, $y_T \leftarrow y_{t+1}$, $b_T \leftarrow b_{t+1}$, $d_T \leftarrow d_{t+1}$
end for
Update Actor, Judge, and Editor using Eq. (5) and Eq. (6)

implemented with Fast Downward, verifies the generated specification and returns

$$(b, d) = \mathcal{E}(y), \qquad b \in \{0, 1\}, \tag{1}$$

where $b = 1$ indicates that the specification is syntactically valid and solver-executable within the time limit, and d contains structured diagnostics such as parse errors, type mismatches, unsatisfied preconditions, unreachable goals, and failed action traces.

The central challenge is that solver success is not identical to task-level semantic faithfulness. Because the model controls the generated specification, optimizing only b can reward degenerate specifications that weaken goals, remove constraints, or alter predicates to make the instance easier than the original task. Our method keeps the solver as the only external feedback source, but separates the learning problem into generation, calibration, and repair roles so that sparse solver outcomes become more useful for policy optimization.

3.2 Role-Conditioned Multi-Role Policy

We use a single language model with shared backbone parameters and lightweight role-specific parameters. Let $\rho \in \{\mathrm{A}, \mathrm{J}, \mathrm{E}\}$ denote the role identifier for Actor, Judge, and Editor. Each role has a learned embedding e_ρ, and the role-conditioned model defines

$$\pi_\theta(\cdot \mid c, \rho) = \mathrm{LM}_\theta\big(\cdot \mid [e_\rho; c]\big)\,, \tag{2}$$

where c is the role-specific context. The Actor context is x, the Judge context is (x, y), and the Editor context is (x, y_t, d_t), where y_t is the current specification and d_t is the latest solver diagnostic.

The parameters are partitioned as

$$\theta = \theta_{\mathrm{sh}} \cup \theta_{\mathrm{A}} \cup \theta_{\mathrm{J}} \cup \theta_{\mathrm{E}}. \tag{3}$$

The shared backbone θ_{sh} contains approximately 95% of the parameters, while the remaining parameters are lightweight role-specific heads. This design allows each role to specialize while keeping most representation learning coupled across generation, verification, and repair.

3.3 Solver-Grounded Training Episode

Algorithm 1 summarizes one training episode. The Actor first samples an initial PDDL specification. The solver verifies it and returns a binary success label with diagnostics. The Judge predicts a scalar score for the generated specification. If the initial specification fails, the Editor performs at most $H_{\max}$ repair steps, each conditioned on the task, the current specification, and the latest diagnostic. The episode stops when the solver succeeds or the repair budget is exhausted.

At inference time, the same procedure is used without gradient updates. The system returns the first solver-executable specification found within the repair horizon; if no repair succeeds, it returns the final edited specification for evaluation. Decoding and optimization details are given in Appendix B.2.

3.4 Role-Specific Learning Signals

The Actor learns global specification generation, the Judge learns solver-calibrated verification, and the Editor learns diagnostic-conditioned repair. The Judge score is produced by a scalar head on top of the

shared representation:

$$s_J(x, y) = \sigma\big(g_\theta([e_{\mathrm{J}}; x; y])\big) \in (0, 1), \tag{4}$$

where g_θ is the Judge logit head and σ is the sigmoid function.

For an episode with initial output y_0, initial solver label b_0, final output y_T, final solver label b_T, and repair length T, the role-specific learning signals are

$$\begin{aligned} R_{\mathrm{A}}(x, y_0) &= b_0 + \lambda_{\mathrm{J}}\, s_J(x, y_0), \\ \mathcal{L}_{\mathrm{J}}(x, y_0, b_0) &= \mathrm{BCE}\big(s_J(x, y_0), b_0\big) + \lambda_{\mathrm{band}}\, \ell_{\mathrm{band}}\big(s_J(x, y_0), b_0\big)\,, \\ R_{\mathrm{E}}(x, y_{0:T}) &= b_T - \beta T. \end{aligned} \tag{5}$$

The Actor reward combines the initial solver label with Judge-based shaping. The Judge loss trains s_J to predict solver success and penalizes over-confident false positives. The Editor reward favors successful repairs with fewer editing steps.

Actor and Editor token policies are optimized with PPO, while the Judge is optimized as a calibrated binary predictor. The total loss is

$$\mathcal{L}_{\mathrm{total}} = \mathcal{L}^{\mathrm{A}}_{\mathrm{PPO}} + \lambda_{\mathrm{E}} \mathcal{L}^{\mathrm{E}}_{\mathrm{PPO}} + \lambda_{\mathrm{J}} \mathcal{L}_{\mathrm{J}}. \tag{6}$$

Appendix B.2 specifies the PPO objective, banded Judge penalty, decoding policy, and hyperparameters.

3.5 Judge Calibration and Correctness Connection

The Judge is trained against solver outcomes rather than human semantic labels. Its direct target is therefore solver-verified executability. The connection to task-level correctness depends on how often solver success agrees with a reference correctness notion on the policy-induced distribution. Let $B \in \{0, 1\}$ denote solver success and $C \in \{0, 1\}$ denote task-level correctness for a generated pair (X, Y). If the on-policy disagreement rate satisfies $\Pr(B \neq C) \leq \varepsilon$, and the learned Judge has calibration error δ_J with respect to the Bayes-optimal solver-success predictor, then

$$\mathbb{E}_{(X,Y)}\left[\left|s_J(X, Y) - C(X, Y)\right|\right] \leq \varepsilon + \delta_J. \tag{7}$$

Thus, the Judge provides a reliable shaping signal when solver success and task-level correctness have limited disagreement on generated samples. Appendix C proves Eq. (7), gives the corresponding ranking guarantee, and describes how to estimate the disagreement rate.

3.6 Why Separate Actor, Judge, and Editor Roles

The three roles address different sources of difficulty in annotation-free symbolic planning. The Actor performs a global mapping from natural language to a complete PDDL domain–problem pair. The Judge converts sparse solver outcomes into a calibrated score that can guide learning. The Editor uses solver diagnostics to make bounded local repairs after failure. A single final-outcome policy can leave the initial specification weakly constrained once local repair becomes strong, because final solver success may no longer distinguish good initial formalizations from poor but repairable ones. Appendix D formalizes this failure mode.

3.7 Parameter Sharing and Reward-Hacking Directions

We now analyze why the shared-backbone design is less permissive than three fully separate role models in a local reward-hacking sense. Let $J_{\mathrm{A}}(\theta)$ denote the Actor objective and let $C(\theta)$ denote expected task-level correctness. Around a reference point θ_0, define the local reward-hacking gradient

$$g_{\mathrm{hack}}(\theta_0) = \nabla_\theta J_{\mathrm{A}}(\theta_0) - a \nabla_\theta C(\theta_0), \qquad a > 0. \tag{8}$$

For a unit-norm perturbation $\Delta\theta$, the first-order Actor gain not explained by correctness is

$$\Delta_{\mathrm{hack}}(\theta_0; \Delta\theta) = g_{\mathrm{hack}}(\theta_0)^\top \Delta\theta.$$

The admissible perturbation set depends on the architecture. With three separate models, the Actor can move in its own P_{A}-dimensional parameter space without directly changing Judge or Editor behavior. With a shared backbone, Judge- and Editor-neutral perturbations are restricted to the small role-private subspace, provided the shared backbone directions are observed by Judge and Editor objectives. Appendix E states the assumptions and proof.

Theorem 3.1 (Reward hacking scaling with parameter count)**.** *Under the local isotropic-gradient model and the shared-backbone constraint in Appendix E, the expected worst-case first-order reward-hacking gain satisfies:*

Table 1: Planning success rate (%) on PlanBench and zero-shot transfer benchmarks. BW denotes BlocksWorld and MBW denotes Mystery BlocksWorld. Avg. is the mean over the four PlanBench domains.

Method	BW	MBW	Logistics	Gripper	Avg.	ProntoQA	Trip	Calendar
LLM^{CoT}	24.0±4.3	0.0±0.0	3.0±1.7	23.0±4.2	12.5±1.6	66.0±4.7	8.0±2.7	34.0±4.7
LLM^{ToT}	3.0±1.7	3.0±1.7	7.0±2.6	13.0±3.4	6.5±1.2	–	–	–
LLM+P	87.0±3.4	37.0±4.8	18.0±3.8	0.0±0.0	35.5±1.8	4.0±2.0	16.0±3.7	12.0±3.2
Ours	**98.0**±1.4	**71.0**±4.5	**58.0**±4.9	**56.0**±5.0	**70.8**±2.1	**90.0**±3.0	**48.0**±5.0	**40.0**±4.9

1. ***Three fully separate models.*** *If the Actor has P_{A} parameters, then*

$$\mathbb{E}\left[\sup_{\Delta\theta\in\mathcal{H}_{\text{sep}}} \Delta_{\text{hack}}(\theta_0;\Delta\theta)\right] \asymp \sigma\sqrt{P_{\text{A}}}. \tag{9}$$

2. ***Shared-backbone architecture.*** *If the Judge- and Editor-neutral perturbations are contained in an Actor head of dimension h, then*

$$\mathbb{E}\left[\sup_{\Delta\theta\in\mathcal{H}_{\text{shr}}} \Delta_{\text{hack}}(\theta_0;\Delta\theta)\right] \lesssim \sigma\sqrt{h}. \tag{10}$$

The theorem shows that separate models allow reward-hacking gains to grow with the Actor parameter count. By contrast, a shared backbone exposes most directions to Judge and Editor objectives, leaving only the small Actor head as the main role-private subspace. Thus, parameter sharing does not eliminate reward hacking, but reduces the local degrees of freedom for improving Actor reward without corresponding verification and repair changes.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate in-domain planning performance on PlanBench (Valmeekam et al., 2023), using four PDDL-derived domains: BlocksWorld, Mystery BlocksWorld, Logistics, and Gripper. Mystery BlocksWorld obfuscates object and predicate names, making it a diagnostic test of planning beyond lexical memorization. We further evaluate zero-shot transfer on ProntoQA (Saparov and He, 2023) and the Trip Planning and Calendar Scheduling subsets of NATURAL PLAN (Zheng et al., 2024). Detailed benchmark descriptions are provided in Appendix G.

Evaluation. For PlanBench, generated domain–problem pairs are evaluated with Fast Downward under a 60-second timeout. An instance is counted as solved only if the generated PDDL is syntactically valid and the returned plan satisfies the goal conditions. For ProntoQA and NATURAL PLAN, we follow the original evaluation protocol and report exact-match success.

Model. Our method is built on Qwen2.5-7B (Qwen et al., 2025) initialized from pretrained weights. We use no human-annotated PDDL demonstrations and no supervised fine-tuning on planning data; learning is driven only by solver-grounded multi-role reinforcement signals described in Section 3.4.

Baselines. We compare with three zero-annotation baselines: Chain-of-Thought prompting (Wei et al., 2022), Tree-of-Thought search (Yao et al., 2023), and LLM+P (Liu et al., 2023). Unless otherwise stated, these baselines use GPT-4o as the underlying model. Appendix G gives the full baseline configurations.

4.2 Main results

Overall planning performance and zero-shot transfer. Table 1 summarizes the main in-domain and out-of-domain results. Our method achieves the best performance on all PlanBench domains, with an average success rate of 70.8%, compared with 35.5% for LLM+P and much lower averages for prompting-only baselines. The gains are largest on Mystery BlocksWorld, Logistics, and Gripper, where lexical shortcuts are less useful and small symbolic errors can invalidate the whole plan. The same model also transfers zero-shot to ProntoQA and NATURAL PLAN, suggesting that solver-grounded multi-role training improves general constraint satisfaction rather than only memorizing PlanBench-specific formats. Additional discussion is provided in Appendix H.2.

Table 2: Controlled comparison under matched solver-call budget. All methods are allowed at most $K = 6$ solver calls per test instance. Faithful denotes solver-successful and semantically faithful outputs. Drift is the conditional fraction of solver-successful outputs that fail semantic checking.

Method	Backbone	BW	MBW	Logistics	Gripper	Avg.	Avg. Calls	Faithful	Drift ↓
Qwen-CoT	Qwen2.5-7B	31	1	5	25	15.5	5.8	11.6	25.2
Qwen-ToT	Qwen2.5-7B	16	5	9	18	12.0	5.9	9.1	24.4
Qwen-LLM+P	Qwen2.5-7B	80	32	20	7	34.8	5.2	27.1	22.1
Qwen-Self-Refine+Solver	Qwen2.5-7B	85	43	36	35	49.8	4.7	37.5	24.7
GPT-4o-Self-Refine+Solver	GPT-4o	91	47	42	37	54.3	4.5	43.0	20.8
Ours	**Qwen2.5-7B**	**98**	**71**	**58**	**56**	**70.8**	**3.2**	**66.3**	**6.4**

Table 3: Semantic faithfulness evaluation on PlanBench. Left: aggregate faithfulness metrics. Right: per-domain faithful success rate. Solv. denotes solver success. Faithful requires both solver executability and semantic consistency with the reference task.

(a) Aggregate faithfulness

Method	Solv.	Faithful	Drift ↓	Goal	Schema-F1
LLM+P	35.5	28.3	20.4	84.7	77.6
Solver-only RL	39.8	25.6	35.7	73.5	68.2
Ours w/o Judge	44.1	29.4	33.3	76.8	70.5
Ours	**70.8**	**66.3**	**6.4**	**96.1**	**91.8**

(b) Per-domain faithful success

Method	BW	MBW	Logistics	Gripper	Avg.
LLM+P	75	26	12	0	28.3
Solver-only RL	51	9	22	20	25.6
Ours w/o Judge	57	13	24	24	29.4
Ours	**94**	**67**	**53**	**51**	**66.3**

Controlled comparison under matched backbone and solver budget. To rule out the possibility that the gains come from a stronger backbone or more solver access, Table 2 compares all methods under a matched $K = 6$ solver-call budget. Our method still outperforms the strongest same-backbone baseline by 21.0 points in average PlanBench success, while using fewer solver calls on average. It also achieves the highest faithful success and the lowest semantic drift, indicating that trained role specialization is more effective than prompting-based diagnostic revision alone. The controlled protocol is detailed in Appendix H.3.

4.3 Analysis

Semantic faithfulness. Because a solver-executable PDDL specification can still deviate from the original task semantics, Table 3 reports both solvability and faithful success. Our method achieves a small solvability–faithfulness gap, with 70.8% solvability and 66.3% faithful success, while reducing drift to 6.4%. The per-domain faithful-success results show that the improvement persists across all PlanBench domains, especially on Mystery BlocksWorld and the structurally brittle Logistics and Gripper domains. The reference-checking protocol and metric definitions are given in Appendix H.4.

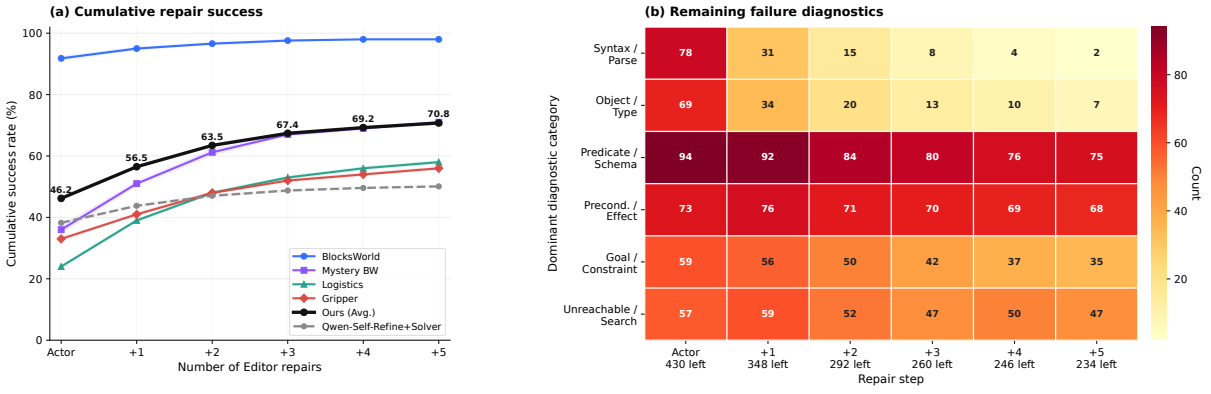

Figure 3: **Repair and diagnostic analysis.** Left: cumulative success rate over Editor repair steps. Step 0 denotes the initial Actor output, and steps 1–5 denote successive repairs. Right: diagnostic heatmap over remaining unsolved cases after each step.

Ablation and semantic drift types. Table 4 shows two diagnostic analyses. The ablation results on Mystery BlocksWorld indicate that Actor, Judge, and Editor are all necessary: removing any role causes a large drop, and using three separate 7B models collapses despite larger total parameter count. The drift-type analysis shows that our method reduces direct shortcut-like failures such as goal weakening and object/type drift; the remaining drift cases are dominated by harder action-schema errors. Detailed

Table 4: Ablation and semantic drift analysis. Left: role ablation on Mystery BlocksWorld. Right: distribution of semantic drift types among solver-successful but semantically unfaithful outputs.

(a) Role ablation

Configuration	Success	Δ
Full: Actor + Judge + Editor	**71.0**	–
Actor only	0.0	-71.0
Editor + Judge	31.5	-39.5
Actor + Judge	20.5	-50.5
Actor + Editor	0.0	-71.0
Separate models, 3×7B	0.0	-71.0

(b) Drift categories

Method	Goal weak.	Type drift	Schema drift	Constraint omit.
LLM+P	31.2	18.8	34.5	15.5
Solver-only RL	38.4	21.7	24.6	15.3
Ours w/o Judge	34.9	19.5	27.1	18.5
Ours	18.6	12.4	47.2	21.8

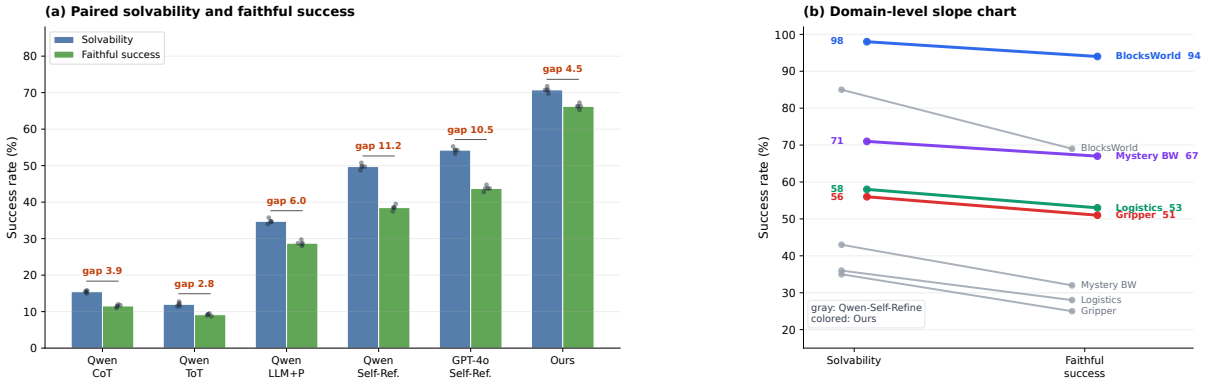

Figure 4: **Solvability–faithfulness gap analysis.** Left: paired bars compare solver success and faithful success under matched-budget settings. Right: domain-level slope chart comparing solver success and faithful success.

interpretation is deferred to Appendix H.6.

Repair dynamics and diagnostic evolution. Figure 3 shows how success accumulates across Editor repair steps. The initial Actor output solves 46.2% of instances on average, and bounded repair increases success to 70.8%. Most improvement occurs early: the first two repairs contribute 17.3 out of the total 24.6 point gain, indicating that the Editor effectively corrects many localized defects rather than relying on long iterative search. The diagnostic heatmap further shows that syntax and typing errors are removed first, while remaining failures increasingly concentrate in harder schema and transition-model errors. This pattern suggests a natural repair hierarchy: shallow formalization errors are rapidly resolved, whereas residual failures require deeper corrections to action semantics and state transitions. The full repair protocol is described in Appendix H.7.

Solvability–faithfulness gap. Figure 4 visualizes the gap between solver success and faithful success. Our method has a 4.5-point gap, compared with 12.3 points for Qwen-Self-Refine+Solver and 11.3 points for GPT-4o-Self-Refine+Solver. Thus, stronger prompting or stronger base models can improve solvability, but they do not fully prevent semantic drift. More details are provided in Appendix H.8.

Judge separation of faithful and unfaithful outputs. Finally, Figure 5 and Table 5 test whether the Judge distinguishes solver-successful but semantically unfaithful outputs from truly faithful ones. The Judge assigns low scores to unsolved outputs, intermediate scores to solved-but-unfaithful outputs, and high scores to faithful outputs, with an overall AUC of 0.89 among solver-successful candidates. This suggests that the Judge captures structural quality signals beyond raw solver executability. Candidate construction and category definitions are given in Appendix H.9.

5 Conclusion

We studied annotation-free natural-language-to-PDDL planning, where a model must learn executable and faithful symbolic specifications from solver feedback alone. We introduced a solver-grounded multi-role reinforcement learning framework that conditions a single language model as an Actor, Judge, and Editor, separating global generation, calibrated verification, and diagnostic-conditioned repair. Across PlanBench and zero-shot transfer benchmarks, the results show that structuring solver feedback into complementary roles improves both planning success and semantic faithfulness, rather than merely increasing solver executability. These findings suggest that verifiable environments can provide scalable

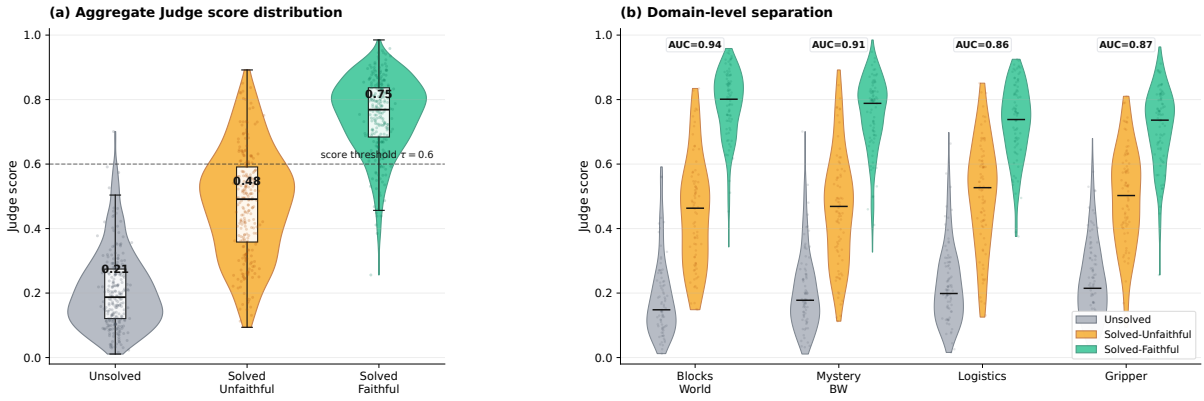

Figure 5: **Judge score distribution over post-hoc semantic categories.** Left: aggregate violin plot over all PlanBench domains. Right: domain-level grouped violin plot.

Table 5: Post-hoc Judge separation analysis. Scores are reported as mean ± standard deviation. AUC measures how well the Judge separates faithful outputs from solved-but-unfaithful outputs among solver-successful candidates.

Domain	#Cand.	Unsolved	Solved-Unfaithful	Solved-Faithful	AUC
BlocksWorld	700	0.16 ± 0.09	0.42 ± 0.13	0.80 ± 0.10	0.94
Mystery BW	830	0.18 ± 0.10	0.47 ± 0.14	0.77 ± 0.11	0.90
Logistics	800	0.20 ± 0.11	0.50 ± 0.15	0.74 ± 0.12	0.87
Gripper	780	0.21 ± 0.11	0.51 ± 0.15	0.73 ± 0.13	0.85
All	3110	0.19 ± 0.11	0.48 ± 0.15	0.76 ± 0.12	0.89

supervision for neuro-symbolic planning when feedback is converted into generation, verification, and repair signals.

A natural next step is to extend this paradigm from classical PDDL domains to richer interactive environments where symbolic constraints, tool feedback, and natural-language goals co-evolve.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution, 2025. URL https://arxiv.org/abs/2510.23595.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Yuanzhuo Wang, and Jian Guo. A survey on llm-as-a-judge. *CoRR*, abs/2411.15594, 2024. doi: 10.48550/ARXIV.2411.15594. URL https://doi.org/10.48550/arXiv.2411.15594.

Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.507. URL https://aclanthology.org/2023.emnlp-main.507/.

Cassie Huang and Li Zhang. On the limit of language models as planning formalizers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4880–4904, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.242. URL https://aclanthology.org/2025.acl-long.242/.

Owen Jiang, Cassie Huang, Ashish Sabharwal, and Li Zhang. Language model planners do not scale, but do formalizers? *arXiv preprint arXiv:2603.23844*, 2026.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Paul Saldyt, and Anil B Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=Th8JPEmH4z.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. From generation to judgment: Opportunities and challenges of LLM-as-a-judge. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng, editors, *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 2757–2791, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.138. URL https://aclanthology.org/2025.emnlp-main.138/.

Zhiming Lin, Kai Zhao, Sophie Zhang, Peilai Yu, and Canran Xiao. Cec-zero: Zero-supervision character error correction with self-generated rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40(28), pages 23612–23620, 2026.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+p: Empowering large language models with optimal planning proficiency, 2023. URL https://arxiv.org/abs/2304.11477.

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

Sadegh Mahdavi, Raquel Aoki, Keyi Tang, and Yanshuai Cao. Leveraging environment interaction for automated PDDL translation and planning with large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=RzlCqnncQv.

OpenAI. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

James Oswald, Kavitha Srinivas, Harsha Kokel, Junkyu Lee, Michael Katz, and Shirin Sohrabi. Large language models as planning domain generators. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, pages 423–431, 2024.

Qwen, An Yang, Baosong Yang, et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Abulhair Saparov and He He. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=qFVVBzXxR2V.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

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, 2023. URL https://arxiv.org/abs/2305.11014.

Joar Max Viktor Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=yb3HOXO3lX2.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. Chain of thoughtlessness? an analysis of cot in planning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=kPBEAZU5Nm.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplanner: Adaptive planning from feedback with language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=rnKgbKmelt.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=YXogl4uQUO.

Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. Llms still can't plan; can lrms? a preliminary evaluation of openai's o1 on planbench, 2024. URL https://arxiv.org/abs/2409.13373.

Pulkit Verma, Ngoc La, Anthony Favier, Swaroop Mishra, and Julie A. Shah. Teaching llms to plan: Logical chain-of-thought instruction tuning for symbolic planning, 2025. URL https://arxiv.org/abs/2509.13351.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models, 2023. URL https://arxiv.org/abs/2302.05128.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h.

Tianyi Zhang, Li Zhang, Zhaoyi Hou, Ziyu Wang, Yuling Gu, Peter Clark, Chris Callison-Burch, and Niket Tandon. Proc2pddl: Open-domain planning representations from texts. In *Proceedings of the 2nd Workshop on Natural Language Reasoning and Structured Explanations (@ ACL 2024)*, pages 13–24, 2024.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data, 2025. URL https://arxiv.org/abs/2505.03335.

Huaixiu Steven Zheng, Swaroop Mishra, Hugh Zhang, Xinyun Chen, Minmin Chen, Azade Nova, Le Hou, Heng-Tze Cheng, Quoc V. Le, Ed H. Chi, and Denny Zhou. Natural plan: Benchmarking llms on natural language planning, 2024. URL https://arxiv.org/abs/2406.04520.

Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning, 2023. URL https://arxiv.org/abs/2308.13724.

Max Zuo, Francisco Piedrahita Velez, Xiaochen Li, Michael Littman, and Stephen Bach. Planetarium: A rigorous benchmark for translating text to structured planning languages. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11223–11240, 2025.

A Reproducibility Details

Scope. This appendix consolidates all implementation- and experiment-related details required to reproduce the proposed solver-grounded multi-role reinforcement learning (RL) framework for NL-to-PDDL planning.

A.1 Task, Data, and Evaluation

Task definition. Given a natural-language planning description x, the goal is to generate a PDDL specification $y = (\mathcal{D}, \mathcal{P})$ consisting of a domain file $\mathcal{D}$ and a problem file $\mathcal{P}$ that can be verified by a PDDL solver (i.e., achieves the goal with valid syntax).

Metric. Primary metric is planning success rate (%), measured by the solver's binary verification result.

A.2 Grounded Verifier (PDDL Solver)

Solver. The environment verifier E is the Fast Downward planner with a 60-second timeout. Given a candidate specification y, the solver returns $(r, f) = E(y)$, where $r \in \{0, 1\}$ indicates goal achievement, and f contains structured diagnostics such as: syntax errors, unsatisfied preconditions, unreachable goals, and execution traces.

A.3 Model, Roles, and Parameter Sharing

Base language model. **Qwen2.5-7B** is used as the pretrained backbone (initialized from pretrained weights only; no supervised fine-tuning).

One Brain, Three Roles. A single model plays three roles via learned role conditioning: Actor ($r = A$), Judge ($r = J$), and Editor ($r = E$). Role conditioning is implemented with a learnable role embedding e_r:

$$p_\theta(y \mid x, r) = \mathrm{LM}_\theta(y \mid [e_r; x]) .$$

Parameter sharing. Parameters are partitioned as $\theta = \theta_{\text{shared}} \cup \theta_A \cup \theta_J \cup \theta_E$, where $|\theta_{\text{shared}}| \approx 0.95|\theta|$. Role-specific parameters (about 1.67% each) are lightweight projection heads.

A.4 RL Objective, Rewards, and Decoding

MDP objective. The generation-refinement process is modeled as an MDP with objective:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right],$$

where γ is the discount factor = 0.9.

Actor (generation). The Actor generates an initial PDDL specification y_0 from the natural-language input. During training, nucleus sampling with temperature $T_{\text{samp}} = 0.7$ is used, and switches to greedy decoding at inference.

Actor reward. The Actor reward combines solver success and the Judge's quality score:

$$R_A = r_{\text{solver}} + \lambda_J \cdot s_{\text{Judge}},$$

where $r_{\text{solver}} \in \{0, 1\}$ and $s_{\text{Judge}} \in [0, 1]$. $\lambda_J = 0.3$.

Judge (quality prediction). The Judge predicts a quality score:

$$s_{\text{Judge}} = \sigma(f_\theta(x, y_0, r = J)).$$

Judge reward enforces consistency with the solver outcome using threshold τ:

$$R_J = \begin{cases} +1 & s_{\text{Judge}} > \tau \ \wedge \ r_{\text{solver}} = 1, \\ -1 & s_{\text{Judge}} > \tau \ \wedge \ r_{\text{solver}} = 0, \\ +0.5 & s_{\text{Judge}} \leq \tau \ \wedge \ r_{\text{solver}} = 0, \\ -0.5 & s_{\text{Judge}} \leq \tau \ \wedge \ r_{\text{solver}} = 1. \end{cases}$$

with $\tau = 0.5$.

Component	Setting
Base model	Qwen2.5-7B
Environment	PlanBench
Solver	Fast Downward, 60s timeout
Learning rate	3×10^{-5}
Batch size	32
PPO clip	$\epsilon = 0.2$
Actor decoding (train)	nucleus sampling, temperature $T_{\text{samp}} = 0.7$
Actor decoding (test)	greedy
Actor reward weight	$\lambda_J = 0.3$
Judge threshold	$\tau = 0.5$
Editor max steps	$T_{\max} = 5$
Editor step penalty	$\beta = 0.1$
Role objective weight	$\lambda_E = 0.5$
Discount factor	$\gamma = 0.9$

Table 6: Hyperparameters and settings.

Editor (bounded refinement). When the solver fails ($r_0 = 0$), the Editor iteratively refines the specification using solver diagnostics f_t:

$$y_{t+1} \sim \pi_E(\cdot \mid x, y_t, f_t), \quad t = 0, 1, \ldots, T.$$

The maximum refinement steps is $T_{\max} = 5$.

Editor reward. The Editor trades off final success and number of edits:

$$R_E = r_{\text{solver}}(y_T) - \beta \cdot T,$$

where $\beta = 0.1$ and $T \leq T_{\max}$.

A.5 Training Protocol and Optimization

Training loop. Training is conducted in cycles. Each cycle includes: (1) Actor generates y_0 and queries the solver for (r_0, f_0); (2) Judge scores s_{Judge} and receives R_J based on solver outcome; (3) if $r_0 = 0$, Editor refines up to $T_{\max}$ steps and receives R_E based on final outcome.

Joint parameter update. All roles share the backbone and are updated with an aggregate gradient:

$$\nabla_\theta J = \nabla_\theta J_A + \lambda_J \nabla_\theta J_J + \lambda_E \nabla_\theta J_E,$$

where $\lambda_J = 0.3$ and $\lambda_E = 0.5$.

RL algorithm (PPO). Optimization uses Proximal Policy Optimization (PPO) with: learning rate 3×10^{-5}, batch size 32, and clipping ratio $\epsilon = 0.2$.

A.6 Hyperparameter Summary

Shown in Table 6

A.7 Compute Resources

All experiments were conducted on a single multi-GPU compute node with 8× NVIDIA A100 GPUs. Each GPU has 80GB memory, and the node has 64 CPU cores and 512GB system memory. The Qwen2.5-7B backbone was trained with mixed precision. Fast Downward was executed on CPU with a fixed 60-second timeout for each generated PDDL specification.

The main solver-grounded multi-role RL run uses approximately 12,000 solver interactions across 8 training cycles. A full training run takes approximately 12–16 hours on the above hardware.

B Additional Method Details

B.1 Solver Environment and Diagnostics

The solver environment $\mathcal{E}$ is implemented with Fast Downward using a 60-second timeout. Given a generated PDDL specification $y = (y^{\text{dom}}, y^{\text{prob}})$, the solver first checks whether the domain and problem

files can be parsed and grounded. If parsing and grounding succeed, the planner searches for a plan that reaches the stated goal. The returned label $b \in \{0, 1\}$ is defined as $b = 1$ when the solver returns a valid plan within the timeout and $b = 0$ otherwise.

The diagnostic field d records the most informative failure mode available from the solver pipeline. We use diagnostics from four categories: syntax and parse errors, type and object mismatches, invalid operator definitions such as missing preconditions or effects, and search-level failures such as unreachable goals. The Editor receives the raw diagnostic text together with the current specification. During training, generated specifications and solver outputs are stored as on-policy interaction data for Actor, Judge, and Editor updates.

B.2 Optimization and Hyperparameters

Decoding. During training, the Actor and Editor use stochastic decoding to maintain exploration. We use nucleus sampling with temperature $\tau_{\text{temp}} = 0.7$. During evaluation, both roles use greedy decoding for deterministic comparison. The repair horizon is fixed to $H_{\max} = 5$. If the initial Actor output succeeds, no Editor step is taken.

Actor and Editor PPO objectives. For role $\rho \in \{\text{A}, \text{E}\}$, let o_t^ρ denote the context given to the role policy, a_t^ρ the generated token sequence, and $\hat{A}_t^\rho$ the advantage computed from the corresponding role reward. PPO minimizes

$$\mathcal{L}_{\text{PPO}}^\rho = -\mathbb{E}_t \left[\min \left(q_t^\rho(\theta) \hat{A}_t^\rho, \text{clip}\left(q_t^\rho(\theta), 1 - \epsilon_{\text{ppo}}, 1 + \epsilon_{\text{ppo}} \right) \hat{A}_t^\rho \right) \right], \tag{11}$$

where

$$q_t^\rho(\theta) = \frac{\pi_\theta(a_t^\rho \mid o_t^\rho, \rho)}{\pi_{\theta_{\text{old}}}(a_t^\rho \mid o_t^\rho, \rho)}.$$

The Actor advantage is computed from R_{A}, and the Editor advantage is computed from R_{E}. We use $\epsilon_{\text{ppo}} = 0.2$.

Judge loss. The Judge is trained as a binary predictor of solver success. The binary cross-entropy term is

$$\text{BCE}(s, b) = -b \log s - (1 - b) \log(1 - s).$$

The banded penalty is

$$\ell_{\text{band}}(s, b) = (1 - b) \max(0, s - \tau_{\text{J}})^2 + \frac{1}{2} b \max(0, \tau_{\text{J}} - s)^2. \tag{12}$$

The first term penalizes false positives above the decision threshold, while the second term penalizes under-confident scores on solver-successful specifications. We use $\tau_{\text{J}} = 0.5$.

Joint update. All trainable parameters are updated with

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{PPO}}^{\text{A}} + \lambda_{\text{E}} \mathcal{L}_{\text{PPO}}^{\text{E}} + \lambda_{\text{J}} \mathcal{L}_{\text{J}}.$$

The shared backbone receives gradients from all three roles. The Actor head receives Actor gradients, the Judge head receives Judge gradients, and the Editor head receives Editor gradients.

C Judge Calibration Analysis

This section proves the calibration statement used in Section 3.5. The result is distributional and on-policy: it concerns the generated specifications encountered under the current policy.

Setup. Let $X \sim \mathcal{P}_{\text{data}}$ be a task and let $Y \sim \pi_\theta(\cdot \mid X)$ be a generated specification. The solver-success label is $B = b(X, Y) \in \{0, 1\}$, and the task-level correctness label is $C = c(X, Y) \in \{0, 1\}$. The solver and the reference correctness checker are deterministic for a fixed pair (x, y); randomness comes from task sampling and stochastic generation.

Define the Bayes-optimal solver-success predictor

$$s_J^*(x, y) = \Pr(B = 1 \mid X = x, Y = y).$$

The learned Judge has calibration error

$$\delta_J = \sup_{x,y} \left| s_J(x, y) - s_J^*(x, y) \right|. \tag{13}$$

Table 7: Hyperparameters used in the solver-grounded multi-role training procedure.

Component	Setting
Base model	Qwen2.5-7B
Solver	Fast Downward
Solver timeout	60 seconds
Actor decoding during training	nucleus sampling, $\tau_{\text{temp}} = 0.7$
Actor decoding during evaluation	greedy decoding
Editor decoding during training	nucleus sampling, $\tau_{\text{temp}} = 0.7$
Editor decoding during evaluation	greedy decoding
Maximum repair steps	$H_{\max} = 5$
PPO clip coefficient	$\epsilon_{\text{ppo}} = 0.2$
Learning rate	3×10^{-5}
Batch size	32
Judge threshold	$\tau_{\text{J}} = 0.5$
Actor shaping weight	$\lambda_{\text{J}} = 0.3$
Editor loss weight	$\lambda_{\text{E}} = 0.5$
Editor step penalty	$\beta = 0.1$

Assumption C.1 (Bounded solver–correctness disagreement)**.** *For the policy-induced distribution over (X, Y), there exists $\varepsilon < 1/2$ such that*

$$\Pr(B \neq C) \leq \varepsilon. \tag{14}$$

Lemma C.1 (Bayes-optimality of the Judge target)**.** *The minimizer of the conditional cross-entropy loss for predicting B is*

$$s_J^*(x, y) = \Pr(B = 1 \mid X = x, Y = y).$$

Proof. For a fixed pair (x, y), let $\eta = \Pr(B = 1 \mid X = x, Y = y)$. The conditional cross-entropy is

$$\ell(s; \eta) = -\eta \log s - (1 - \eta) \log(1 - s), \qquad s \in (0, 1).$$

Its derivative is $-\eta/s + (1 - \eta)/(1 - s)$, which vanishes only at $s = \eta$. The second derivative is positive on $(0, 1)$, so $s = \eta$ is the unique minimizer. □

Theorem C.2 (Judge–correctness connection)**.** *Under Assumption C.1, any Judge with calibration error δ_J satisfies*

$$\mathbb{E}_{(X,Y)} \left[\left| s_J(X, Y) - C(X, Y) \right| \right] \leq \varepsilon + \delta_J. \tag{15}$$

Moreover,

$$\Pr\left(\left| s_J(X, Y) - C(X, Y) \right| > \delta_J \right) \leq \varepsilon. \tag{16}$$

Proof. By Lemma C.1, $s_J^*(X, Y) = \Pr(B = 1 \mid X, Y)$. Since B is deterministic given (X, Y), we have $s_J^*(X, Y) = B(X, Y)$ almost surely. Therefore,

$$|s_J(X, Y) - C(X, Y)| \leq |s_J(X, Y) - s_J^*(X, Y)| + |B(X, Y) - C(X, Y)|.$$

Taking expectations yields Eq. (15). For Eq. (16), note that on the event $B = C$, the first term is at most δ_J. Thus the event $|s_J - C| > \delta_J$ can occur only when $B \neq C$, whose probability is at most ε. □

Theorem C.3 (Margin ranking bound)**.** *Let $Y_1, Y_2 \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot \mid X)$. Under Assumption C.1,*

$$\begin{aligned} \Pr\Big(& s_J(X, Y_1) \geq s_J(X, Y_2) + 2\delta_J \\ & \wedge\ C(X, Y_1) < C(X, Y_2) \Big) \leq 2\varepsilon. \end{aligned} \tag{17}$$

Proof. Let G be the event that $B(X, Y_i) = C(X, Y_i)$ for both $i = 1, 2$. By a union bound, $\Pr(G^c) \leq 2\varepsilon$. On G, the learned score differs from the binary correctness value by at most δ_J for each candidate. Therefore, a score margin of $2\delta_J$ cannot rank an incorrect candidate above a correct candidate. The mis-ranking event in Eq. (17) is contained in G^c, so its probability is at most 2ε. □

C.1 Estimating the Solver–Correctness Disagreement Rate

For a fixed checkpoint θ, the disagreement rate can be estimated by sampling tasks X_i, generating specifications Y_i, recording solver labels B_i, and evaluating task-level correctness labels C_i using a reference validator or audit protocol:

$$\widehat{\varepsilon}_\theta = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}\{B_i \neq C_i\}. \tag{18}$$

For any $\delta \in (0,1)$, Hoeffding's inequality gives

$$\varepsilon_\theta \leq \widehat{\varepsilon}_\theta + \sqrt{\frac{\log(1/\delta)}{2n}} \tag{19}$$

with probability at least $1-\delta$. For a finite set of checkpoints Θ_{ckpt}, a uniform bound is obtained by replacing δ with $\delta/|\Theta_{\text{ckpt}}|$.

D Why Separate Actor and Editor Roles

This section clarifies why the Actor and Editor are trained as separate roles rather than a single policy optimized only for final solver success.

Initial correctness. For a policy that produces an initial specification Y_0, define

$$C_{\text{init}}(\pi) = \mathbb{E}\left[c(X, Y_0)\right].$$

Here $c(X, Y_0) = 1$ means that the initial specification is correct with respect to the task-level reference semantics.

Actor-only alignment. If the Actor is trained with the initial solver label $b(X, Y_0)$, then under the bounded disagreement condition $\Pr(b(X, Y_0) \neq c(X, Y_0)) \leq \varepsilon$,

$$|\mathbb{E}[b(X, Y_0)] - C_{\text{init}}(\pi)| \leq \varepsilon.$$

Thus the initial solver label remains a distributional surrogate for initial correctness whenever solver success and task-level correctness agree on most on-policy samples.

Final-only Editor objective. Consider a monolithic repair policy that both initializes and edits, and receives only the final reward

$$R_{\text{mono}} = b(X, Y_T) - \beta T.$$

Suppose there exists a set of repairable initial specifications $\mathcal{Y}_{\text{rep}}(x)$ such that every $y_0 \in \mathcal{Y}_{\text{rep}}(x)$ can be repaired to solver success in the same number of steps. Then all initializers supported on $\mathcal{Y}_{\text{rep}}(x)$ receive the same final return, even if their initial correctness differs. The final-only objective therefore does not identify the quality of Y_0.

Proposition D.1 (Initial specification is unidentifiable under saturated repair)**.** *Assume that for every x, all $y_0 \in \mathcal{Y}_{\text{rep}}(x)$ can be repaired to $b = 1$ in exactly $T^\dagger$ steps. If $\mathcal{Y}_{\text{rep}}(x)$ contains both correct and incorrect initial specifications, then for any $\alpha \in [0,1]$ there exists an initializer with $C_{\text{init}} = \alpha$ and final return $1 - \beta T^\dagger$.*

Proof. For each x, choose $y^+(x), y^-(x) \in \mathcal{Y}_{\text{rep}}(x)$ such that $c(x, y^+(x)) = 1$ and $c(x, y^-(x)) = 0$. Define the initializer to output $y^+(x)$ with probability α and $y^-(x)$ with probability $1-\alpha$. By the saturated repair assumption, both choices obtain final return $1 - \beta T^\dagger$. The expected initial correctness is α. □

This proposition motivates the Actor–Editor decomposition. The Actor receives an initial-stage reward, which keeps global formalization tied to the initial specification. The Editor receives a final-stage repair reward, which specializes it for local diagnostic-conditioned correction.

E Proof of Reward-Hacking Scaling

This section proves Theorem 3.1.

Local hacking gain. Let $J_A(\theta)$ denote the Actor objective and $C(\theta)$ denote expected task-level correctness. Around a reference point θ_0, define

$$g_{\text{hack}}(\theta_0) = \nabla_\theta J_A(\theta_0) - a\nabla_\theta C(\theta_0), \qquad a > 0.$$

For a unit-norm perturbation $\Delta\theta$, the first-order Actor gain not explained by correctness is

$$\Delta_{\text{hack}}(\theta_0; \Delta\theta) = g_{\text{hack}}(\theta_0)^\top \Delta\theta.$$

Admissible perturbation sets. For three separate models, the Actor has its own parameter vector $\theta_A \in \mathbb{R}^{P_A}$. The admissible perturbation set is

$$\mathcal{H}_{\text{sep}} = \left\{ \Delta\theta_A \in \mathbb{R}^{P_A} : \|\Delta\theta_A\|_2 \le 1 \right\}. \tag{20}$$

For the shared-backbone architecture, write

$$\theta = (\theta_{\text{sh}}, \theta_A, \theta_J, \theta_E),$$

where $\theta_{\text{sh}} \in \mathbb{R}^p$ is the shared backbone and $\theta_A \in \mathbb{R}^h$ is the Actor head. We consider perturbations that are neutral to Judge and Editor on the shared backbone and do not modify the Judge or Editor heads:

$$\mathcal{H}_{\text{shr}} = \left\{ \Delta\theta : \|\Delta\theta\|_2 \le 1,\ \Delta\theta_J = 0,\ \Delta\theta_E = 0,\ \Delta\theta_{\text{sh}} \in \text{Null}(\Sigma_{JE}) \right\}. \tag{21}$$

Here Σ_{JE} is the Judge–Editor backbone sensitivity matrix defined below.

Assumption E.1 (Judge and Editor constrain the shared backbone)**.** *Let $g_J \in \mathbb{R}^p$ and $g_E \in \mathbb{R}^p$ denote stochastic backbone gradients from the Judge and Editor objectives. Define*

$$\Sigma_{JE} = \mathbb{E}[g_J g_J^\top] + \mathbb{E}[g_E g_E^\top]. \tag{22}$$

Assume $\text{Null}(\Sigma_{JE}) = \{0\}$*. Therefore, any Judge- and Editor-neutral perturbation in $\mathcal{H}_{\text{shr}}$ must lie in the Actor head subspace.*

Assumption E.2 (Local isotropic hacking-gradient model)**.** *There exists $\sigma > 0$ such that the restriction of $g_{\text{hack}}(\theta_0)$ to the relevant admissible subspace is isotropic Gaussian. In the separate setting,*

$$g_{\text{hack}}(\theta_0) \sim \mathcal{N}(0, \sigma^2 I_{P_A}).$$

In the shared setting,

$$\text{Proj}_A g_{\text{hack}}(\theta_0) \sim \mathcal{N}(0, \sigma^2 I_h),$$

where Proj_A *denotes projection onto the Actor head subspace.*

Lemma E.1 (Gaussian supremum over a unit ball)**.** *Let $g \sim \mathcal{N}(0, \sigma^2 I_d)$. Then*

$$\mathbb{E}\left[\sup_{\|u\|_2 \le 1} g^\top u \right] = \mathbb{E}\|g\|_2 = \Theta(\sigma\sqrt{d}). \tag{23}$$

Proof. For any fixed g, Cauchy–Schwarz gives

$$\sup_{\|u\|_2 \le 1} g^\top u = \|g\|_2,$$

with equality at $u = g/\|g\|_2$ when $g \neq 0$. Since $g \sim \mathcal{N}(0, \sigma^2 I_d)$, $\|g\|_2/\sigma$ follows a χ_d distribution, whose expectation is $\Theta(\sqrt{d})$. □

Proof of Theorem 3.1. For the separate-model setting, $\mathcal{H}_{\text{sep}}$ is the unit ball in the Actor parameter space. Therefore,

$$\sup_{\Delta\theta \in \mathcal{H}_{\text{sep}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta) = \sup_{\|\Delta\theta_A\|_2 \le 1} g_{\text{hack}}(\theta_0)^\top \Delta\theta_A = \|g_{\text{hack}}(\theta_0)\|_2.$$

By Assumption E.2 and Lemma E.1 with $d = P_A$,

$$\mathbb{E}\left[\sup_{\Delta\theta \in \mathcal{H}_{\text{sep}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta) \right] = \Theta(\sigma\sqrt{P_A}).$$

For the shared-backbone setting, Assumption E.1 gives $\Delta\theta_{\text{sh}} = 0$ for every $\Delta\theta \in \mathcal{H}_{\text{shr}}$, and by construction $\Delta\theta_J = \Delta\theta_E = 0$. Hence every admissible perturbation lies in the Actor head subspace:

$$\Delta\theta = (0, \Delta\theta_A, 0, 0), \qquad \|\Delta\theta_A\|_2 \le 1.$$

Thus,

$$\sup_{\Delta\theta\in\mathcal{H}_{\text{shr}}} \Delta_{\text{hack}}(\theta_0;\Delta\theta) = \sup_{\|\Delta\theta_A\|_2\le 1} \left(\text{Proj}_A g_{\text{hack}}(\theta_0)\right)^\top \Delta\theta_A.$$

By Assumption E.2 and Lemma E.1 with $d = h$,

$$\mathbb{E}\left[\sup_{\Delta\theta\in\mathcal{H}_{\text{shr}}} \Delta_{\text{hack}}(\theta_0;\Delta\theta)\right] = \mathbb{E}\left[\|\text{Proj}_A g_{\text{hack}}(\theta_0)\|_2\right] \lesssim \sigma\sqrt{h}.$$

This proves both claims. □

Effective-rank variant. If $\text{Null}(\Sigma_{\text{JE}})$ has dimension $k > 0$, the same proof gives a shared-architecture scaling of order

$$\Theta(\sigma\sqrt{h+k}),$$

where k is the number of Judge- and Editor-neutral backbone directions. Thus the relevant quantity is the dimension of the role-private or role-neutral subspace, rather than the total backbone parameter count.

F Proofs for One Brain Three Roles

Throughout, X denotes a task sampled from $\mathcal{D}$, Y a PDDL specification produced by the policy, and $R, C \in \{0,1\}$ are respectively the *solver success indicator* ("solvability") and the *ground-truth plan correctness indicator* ("correctness") of the final plan for (X,Y): $R = 1$ iff the PDDL solver parses Y and returns a plan that reaches its goal under the semantics encoded by (X,Y), and $C = 1$ iff that final plan is correct under the true task semantics. We write $C(\theta) = \mathbb{E}[C]$ for the expected correctness under policy π_θ.

F.1 Judge–Correctness Consistency

Deterministic oracles; randomness only from sampling. Although the planner and the validator are deterministic maps for any fixed natural-language task x and PDDL specification y, the pair (X,Y) is random because (i) tasks are sampled as $X \sim \mathcal{D}$ and (ii) specifications are sampled as $Y \sim \pi_\theta(\cdot \mid X)$ (e.g., via stochastic decoding). Hence, R and C are random variables through (X,Y), even though their dependence on (x,y) is deterministic.

Formally, there exist measurable functions $r, c : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$ such that

$$R \;=\; r(X,Y) \in \{0,1\}, \qquad C \;=\; c(X,Y) \in \{0,1\}. \tag{24}$$

We define the conditional success and correctness probabilities

$$\eta(x,y) := \mathbb{E}[R \mid X = x, Y = y] = \Pr(R = 1 \mid X = x, Y = y), \tag{25}$$

$$\kappa(x,y) := \mathbb{E}[C \mid X = x, Y = y] = \Pr(C = 1 \mid X = x, Y = y). \tag{26}$$

Under determinism, $\eta(x,y) = r(x,y)$ and $\kappa(x,y) = c(x,y)$ are $\{0,1\}$-valued. We keep the probabilistic notation because all bounds below are taken with respect to the *distribution of* (X,Y) induced by $(\mathcal{D}, \pi_\theta)$.

Bounded solver–correctness drift (on-policy). We allow "specification drift" in which the planner declares success for a generated specification while the resulting behavior is not correct under the ground-truth validator; we assume such drift is rare on the distribution induced by policies of interest.

Assumption F.1 (Bounded solver–correctness drift rate.)**.** *There exists $\varepsilon \in [0, 1/2)$ such that for every policy π_θ considered in the analysis,*

$$\varepsilon_\theta \;:=\; \Pr_{X\sim\mathcal{D},\, Y\sim\pi_\theta(\cdot|X)}\big(R \neq C\big) \;\le\; \varepsilon. \tag{27}$$

Justification and empirical results in F.2

Judge training and calibration. The Judge outputs $s_J : \mathcal{X} \times \mathcal{Y} \to (0,1)$ and is trained by conditional cross-entropy for predicting the planner outcome R:

$$\mathcal{L}(s) \;=\; \mathbb{E}\big[-R \log s(X,Y) - (1-R)\log\big(1 - s(X,Y)\big)\big], \tag{28}$$

where the expectation is taken over the joint distribution of (X,Y) used to train the Judge (typically the on-policy distribution induced by the Actor). Let $s_J^*(x,y)$ denote the Bayes pointwise minimizer of (28). We quantify approximation/calibration by

$$\delta_J \;:=\; \sup_{x,y} \big|s_J(x,y) - s_J^*(x,y)\big|. \tag{29}$$

Lemma F.1 (Bayes-optimal Judge under cross-entropy). *For every (x, y), the pointwise minimizer of (28) satisfies*

$$s_J^*(x, y) \;=\; \eta(x, y) \;=\; \Pr(R = 1 \mid X = x, Y = y). \tag{30}$$

Proof. Fix (x, y) and write $\eta = \eta(x, y)$. The conditional cross-entropy for predicting R from a score $s \in (0, 1)$ is $\ell(s; \eta) = -\eta \log s - (1 - \eta) \log(1 - s)$. Differentiating gives $\frac{\partial \ell}{\partial s}(s; \eta) = -\eta/s + (1 - \eta)/(1 - s)$, which vanishes only at $s = \eta$. Moreover, $\frac{\partial^2 \ell}{\partial s^2}(s; \eta) = \eta/s^2 + (1 - \eta)/(1 - s)^2 > 0$ for all $s \in (0, 1)$, hence $\ell(\cdot; \eta)$ is strictly convex and $s = \eta$ is the unique minimizer. □

Bounding solvability vs. correctness (distributionally). The key deterministic fact is that, since $R, C \in \{0, 1\}$ and both are deterministic functions of (X, Y), the pointwise gap $|\eta(X, Y) - \kappa(X, Y)|$ is exactly the drift indicator $\mathbb{I}\{R \neq C\}$.

Lemma F.2 (Distributional solvability–correctness gap). *Under Assumption F.1,*

$$\mathbb{E}\big[\, |\eta(X, Y) - \kappa(X, Y)| \,\big] \;=\; \Pr(R \neq C) \;\leq\; \varepsilon. \tag{31}$$

Consequently,

$$\big|\mathbb{E}[R] - \mathbb{E}[C]\big| \;\leq\; \varepsilon. \tag{32}$$

Proof. Because $R = r(X, Y)$ and $C = c(X, Y)$ are $\{0, 1\}$-valued, almost surely $|R - C| = \mathbb{I}\{R \neq C\}$. Under determinism, $\eta(X, Y) = \mathbb{E}[R \mid X, Y] = R$ and $\kappa(X, Y) = \mathbb{E}[C \mid X, Y] = C$ almost surely. Thus $|\eta(X, Y) - \kappa(X, Y)| = |R - C| = \mathbb{I}\{R \neq C\}$. Taking expectations yields (31), and (32) follows from Jensen: $|\mathbb{E}[R] - \mathbb{E}[C]| = |\mathbb{E}[R - C]| \leq \mathbb{E}[|R - C|] \leq \varepsilon$. □

Theorem F.3 (Judge–correctness consistency). *Let s_J^* be the Bayes-optimal Judge from Lemma F.1, and let s_J be any learned Judge with calibration error δ_J defined in (29). Under Assumption F.1,*

$$\mathbb{E}\big[\, |s_J(X, Y) - \kappa(X, Y)| \,\big] \;\leq\; \varepsilon + \delta_J, \tag{33}$$

$$\Pr\big(|s_J(X, Y) - \kappa(X, Y)| > \delta_J\big) \;\leq\; \varepsilon. \tag{34}$$

Moreover, if $Y_1, Y_2 \overset{\text{i.i.d.}}{\sim} \pi_\theta(\cdot \mid X)$, then the Judge's margin *controls correctness ranking up to drift:*

$$\Pr\Big(C(X, Y_1) < C(X, Y_2) \;\wedge\; s_J(X, Y_1) \geq s_J(X, Y_2) + 2\delta_J\Big) \;\leq\; 2\varepsilon. \tag{35}$$

Proof. By Lemma F.1, $s_J^*(x, y) = \eta(x, y)$.

(i) Expected consistency. By triangle inequality, $|s_J - \kappa| \leq |s_J - s_J^*| + |s_J^* - \kappa| = |s_J - s_J^*| + |\eta - \kappa|$. Taking expectations and using $\mathbb{E}[|s_J - s_J^*|] \leq \delta_J$ and Lemma F.2 gives (33).

(ii) High-probability consistency. On the event $\{R = C\}$ we have $\kappa = \eta = s_J^*$, hence $|s_J - \kappa| = |s_J - s_J^*| \leq \delta_J$.Therefore $\{|s_J - \kappa| > \delta_J\} \subseteq \{R \neq C\}$ and $\Pr(|s_J - \kappa| > \delta_J) \leq \Pr(R \neq C) \leq \varepsilon$, proving (34).

(iii) Ranking with a margin. Let $G := \{R(X, Y_1) = C(X, Y_1)\} \cap \{R(X, Y_2) = C(X, Y_2)\}$. By a union bound and Assumption F.1, $\Pr(G^c) \leq \Pr(R \neq C \text{ for } (X, Y_1)) + \Pr(R \neq C \text{ for } (X, Y_2)) \leq 2\varepsilon$. On G, we have $C(X, Y_i) = R(X, Y_i) = \eta(X, Y_i)$ for $i = 1, 2$, and $|s_J(X, Y_i) - \eta(X, Y_i)| \leq \delta_J$. Hence on G, $s_J(X, Y_1) \geq s_J(X, Y_2) + 2\delta_J \Rightarrow \eta(X, Y_1) \geq \eta(X, Y_2) \Rightarrow C(X, Y_1) \geq C(X, Y_2)$. Thus the event in (35) can only occur on G^c, so its probability is at most 2ε. □

F.2 Justifying the Drift-Rate Parameter ε

This subsection justifies Assumption F.1 (bounded solver–correctness drift rate) and outlines how to set ε in a deterministic planning environment.

Deterministic oracles; on-policy drift rate. The planner and the (ground-truth) validator are deterministic given a task x and a specification y:

$$R = r(x, y) \in \{0, 1\}, \qquad C = c(x, y) \in \{0, 1\}.$$

Randomness enters only through sampling $X \sim \mathcal{D}$ and stochastic generation $Y \sim \pi_\theta(\cdot \mid X)$. Define the drift indicator

$$D \;:=\; \mathbb{I}\{R \neq C\} \in \{0, 1\}. \tag{36}$$

For a fixed policy π_θ, we define the induced *on-policy drift rate*

$$\varepsilon_\theta \;\coloneqq\; \Pr_{X\sim\mathcal{D},\, Y\sim\pi_\theta(\cdot|X)}(R \neq C) \;=\; \mathbb{E}[D] \;=\; \mathbb{E}\big[|R-C|\big]. \tag{37}$$

Assumption F.1 postulates that for the policy class of interest, $\sup_\theta \varepsilon_\theta \le \varepsilon$ for some finite $\varepsilon < \frac{1}{2}$. Note that this assumption is *distributional*: it bounds the *frequency* of drift under the policy-induced distribution, rather than imposing a pointwise (conditional) noise model.

Empirical evidence from solvability vs. correctness Huang and Zhang (2025) report two evaluation metrics closely aligned with our notation: *Solvability* (whether a planner finds a plan) and *Correctness* (whether the plan is accepted under a higher-fidelity reference semantics/validator). In their experiments across domains and models (e.g. GPT-40, Llama-8B), the observed gap between the two metrics is typically small (typically well below $0.1 < 1/2$), suggesting that solver–correctness drift is limited for realistic LLM policies.

Formally, for any fixed π_θ define the induced on-policy rates

$$S(\theta) \coloneqq \Pr(R=1), \qquad K(\theta) \coloneqq \Pr(C=1),$$

where probabilities are over $(X,Y) \sim (\mathcal{D}, \pi_\theta)$. Then the observable solvability–correctness gap is always controlled by the drift rate:

$$|S(\theta) - K(\theta)| \;=\; \big|\mathbb{E}[R] - \mathbb{E}[C]\big| \;\le\; \mathbb{E}\big[|R-C|\big] \;=\; \varepsilon_\theta. \tag{38}$$

Moreover, under the common evaluation convention that correctness is defined only for solver-produced plans (so $C \le R$ almost surely), the gap equals the (false-positive) drift probability:

$$\varepsilon_\theta \;=\; \Pr(R=1, C=0) \;=\; \Pr(R=1) - \Pr(C=1) \;=\; S(\theta) - K(\theta). \tag{39}$$

Thus, the small empirical solvability–correctness gaps reported by Huang and Zhang provide direct evidence that ε_θ is small in practice for realistic policies, and in particular that taking a uniform constant $\varepsilon < \frac{1}{2}$ is reasonable.

Estimating ε_θ and choosing a uniform ε. For a fixed policy π_θ, we can estimate ε_θ by sampling $\{X_i\}_{i=1}^n \sim \mathcal{D}$, generating $Y_i \sim \pi_\theta(\cdot \mid X_i)$, and computing $R_i = r(X_i, Y_i)$ and $C_i = c(X_i, Y_i)$, yielding the empirical drift rate

$$\widehat{\varepsilon}_\theta \;\coloneqq\; \frac{1}{n}\sum_{i=1}^n \mathbb{I}\{R_i \neq C_i\}. \tag{40}$$

By Hoeffding's inequality, for any $\delta \in (0,1)$, with probability at least $1-\delta$,

$$\varepsilon_\theta \;\le\; \widehat{\varepsilon}_\theta + \sqrt{\frac{\log(1/\delta)}{2n}}. \tag{41}$$

To obtain a uniform bound over a finite set of checkpoints Θ_{ckpt}, we may set

$$\varepsilon \;\coloneqq\; \sup_{\theta\in\Theta_{\text{ckpt}}} \widehat{\varepsilon}_\theta \;+\; \sqrt{\frac{\log(|\Theta_{\text{ckpt}}|/\delta)}{2n}}, \tag{42}$$

so that (by a union bound) $\sup_{\theta\in\Theta_{\text{ckpt}}} \varepsilon_\theta \le \varepsilon$ holds with probability at least $1-\delta$. Our theory only requires that ε is finite and satisfies $\varepsilon < \frac{1}{2}$, i.e., drift is not the majority behavior on-policy.

F.3 Why Actor–Editor: Actor-only vs. Editor-only vs. Actor–Editor

We compare three training designs for PDDL formalization: *Actor-only*, *Editor-only* (a single monolithic Editor-like policy), and *Actor–Editor*. All results are stated for deterministic planner/validator semantics; all probabilities are over (X,Y) induced by task sampling $X \sim \mathcal{D}$ and stochastic generation $Y \sim \pi(\cdot \mid X)$.

Setup. Let $R = r(X,Y) \in \{0,1\}$ be solver success and $C = c(X,Y) \in \{0,1\}$ be true correctness. For any policy that produces an *initial* specification Y_0, define

$$C_{\text{init}}(\pi) \;\coloneqq\; \mathbb{E}\big[C(X,Y_0)\big] \in [0,1]. \tag{43}$$

Assumption F.2 (Bounded solver–correctness drift rate)**.** *There exists $\varepsilon \in [0, 1/2)$ such that for every policy π considered in this comparison,*

$$\Pr_{X\sim\mathcal{D},\, Y\sim\pi(\cdot|X)}\big(r(X,Y) \neq c(X,Y)\big) \;\le\; \varepsilon. \tag{44}$$

A generic optimization-to-alignment lemma.

Lemma F.4 (Generic gap bound)**.** *Let $C(\theta) \in [0,1]$ and $J(\theta) \in \mathbb{R}$ satisfy*

$$\big|J(\theta) - (a\,C(\theta) + b)\big| \le \Delta \qquad \forall \theta, \tag{45}$$

for some $a > 0$, $b \in \mathbb{R}$, $\Delta \ge 0$. If $\theta_J \in \arg\max_\theta J(\theta)$ and $\theta_C \in \arg\max_\theta C(\theta)$, then

$$C(\theta_C) - C(\theta_J) \;\le\; \frac{2\Delta}{a}. \tag{46}$$

Proof. From (45), for any θ, $J(\theta) \ge aC(\theta) + b - \Delta$ and $J(\theta) \le aC(\theta) + b + \Delta$. Using optimality of θ_J and θ_C yields $aC(\theta_J) + b + \Delta \ge J(\theta_J) \ge J(\theta_C) \ge aC(\theta_C) + b - \Delta$, hence $a(C(\theta_C) - C(\theta_J)) \le 2\Delta$. □

Actor-only: solver reward aligns initial correctness on-policy

Definition. Actor-only samples $Y_0 \sim \pi_A(\cdot \mid X)$ and maximizes

$$J_{\mathrm{A-only}}(\pi_A) \;\coloneqq\; \mathbb{E}\big[r(X, Y_0)\big]. \tag{47}$$

Theorem F.5 (Actor-only initial-alignment under drift rate)**.** *Under Assumption F.2, for all π_A,*

$$\big|J_{\mathrm{A-only}}(\pi_A) - C_{\mathrm{init}}(\pi_A)\big| \;\le\; \varepsilon. \tag{48}$$

Consequently any maximizer $\pi_A^\star \in \arg\max_{\pi_A} J_{\mathrm{A-only}}(\pi_A)$ satisfies

$$C_{\mathrm{init}}(\pi_A^\star) \;\ge\; \sup_{\pi_A} C_{\mathrm{init}}(\pi_A) - 2\varepsilon. \tag{49}$$

Proof. Since $r, c \in \{0,1\}$,

$$\big|J_{\mathrm{A-only}}(\pi_A) - C_{\mathrm{init}}(\pi_A)\big| = \big|\mathbb{E}[r(X,Y_0) - c(X,Y_0)]\big| \le \mathbb{E}\big[|r-c|\big] = \Pr(r \ne c) \le \varepsilon,$$

proving (48). Apply Lemma F.4 with $a = 1$, $b = 0$, $\Delta = \varepsilon$ to obtain (49). □

Editor-only: final-outcome objective cannot constrain initial correctness

Monolithic Editor-only baseline. A monolithic policy initializes $Y_0 \sim q(\cdot \mid X)$ and iteratively repairs:

$$Y_{t+1} \sim \pi_E^{\mathrm{mono}}(\cdot \mid X, Y_t, F_t, t), \qquad t = 0, \dots, T-1.$$

The objective depends only on the final solver outcome:

$$J_{\mathrm{mono}}(q, \pi_E^{\mathrm{mono}}) \;\coloneqq\; \mathbb{E}\big[r(X, Y_T) - \beta T\big]. \tag{50}$$

Assumption F.3 (Editor-saturated repair)**.** *There exist sets $\mathcal{Y}_{\mathrm{good}}(x) \subseteq \mathcal{Y}$ and constants $T^\dagger \in \mathbb{N}$, $\beta \ge 0$ such that:*

1. *(*Non-catastrophic initializations*)* $\Pr(Y_0 \in \mathcal{Y}_{\mathrm{good}}(X)) = 1$.
2. *(*Uniform repair success*) There exists a repair policy $\pi_E^\dagger$ such that for all x and all $y_0 \in \mathcal{Y}_{\mathrm{good}}(x)$,*

$$\Pr_{\pi_E^\dagger}\big(r(x, Y_{T^\dagger}) = 1 \mid X = x, Y_0 = y_0\big) = 1, \quad \textit{and} \quad T = T^\dagger \textit{ a.s.}$$

Assumption F.4 (Nontrivial good-set)**.** *For every x there exist $y^+(x), y^-(x) \in \mathcal{Y}_{\mathrm{good}}(x)$ such that $c(x, y^+(x)) = 1$ and $c(x, y^-(x)) = 0$.*

Theorem F.6 (Editor-only unidentifiability of initial correctness)**.** *Under Assumptions F.3 and F.4, for every $\alpha \in [0,1]$ there exists a monolithic policy $(q^\alpha, \pi_E^\dagger)$ such that*

$$J_{\mathrm{mono}}(q^\alpha, \pi_E^\dagger) = 1 - \beta T^\dagger \quad \textit{and} \quad C_{\mathrm{init}}(q^\alpha) = \alpha. \tag{51}$$

In particular, J_{mono} admits globally optimal policies with arbitrarily low C_{init}.

Proof. Fix $\alpha \in [0,1]$ and define $q^\alpha(\cdot \mid x)$ by $Y_0 = y^+(x)$ with probability α and $Y_0 = y^-(x)$ with probability $1 - \alpha$. By Assumption F.3, for any $y_0 \in \mathcal{Y}_{\mathrm{good}}(x)$, repair by $\pi_E^\dagger$ yields $r(x, Y_{T^\dagger}) = 1$ a.s. and $T = T^\dagger$ a.s., hence $\mathbb{E}[r(X,Y_T) - \beta T \mid X = x, Y_0 = y_0] = 1 - \beta T^\dagger$ independent of y_0. Taking expectation over X and $Y_0 \sim q^\alpha(\cdot \mid X)$ gives $J_{\mathrm{mono}}(q^\alpha, \pi_E^\dagger) = 1 - \beta T^\dagger$.

Moreover, by Assumption F.4, $c(X, y^+(X)) = 1$ and $c(X, y^-(X)) = 0$, so

$$C_{\mathrm{init}}(q^\alpha) = \mathbb{E}[c(X, Y_0)] = \mathbb{E}[\alpha \cdot 1 + (1-\alpha)\cdot 0] = \alpha.$$

□

Actor–Editor: restores initial alignment and improves final solvability

Actor reward with a calibrated Judge. Actor–Editor samples $Y_0 \sim \pi_A(\cdot \mid X)$ and uses an Actor reward

$$R_A \;\coloneqq\; r(X, Y_0) + \lambda_J s_J(X, Y_0), \qquad \lambda_J \geq 0, \tag{52}$$

with objective $J_A(\pi_A) \coloneqq \mathbb{E}[R_A]$. We assume the Judge is calibrated to the solver label $r(X, Y)$.

Assumption F.5 (Judge calibration to the solver)**.** *There exists $\delta_J \geq 0$ such that*

$$\sup_{x,y} \big| s_J(x, y) - r(x, y) \big| \leq \delta_J. \tag{53}$$

Lemma F.7 (Judge is correctness-consistent on-policy)**.** *Under Assumptions F.2 and F.5, for any policy π generating (X, Y),*

$$\mathbb{E}\big[\, |s_J(X, Y) - c(X, Y)| \,\big] \;\leq\; \delta_J + \varepsilon. \tag{54}$$

Moreover,

$$\Pr\big(|s_J(X, Y) - c(X, Y)| > \delta_J\big) \leq \varepsilon. \tag{55}$$

Proof. By triangle inequality, $|s_J - c| \leq |s_J - r| + |r - c|$. Taking expectation gives $\mathbb{E}|s_J - c| \leq \mathbb{E}|s_J - r| + \mathbb{E}|r - c| \leq \delta_J + \Pr(r \neq c) \leq \delta_J + \varepsilon$. For the high-probability statement, on the event $\{r = c\}$ we have $|s_J - c| = |s_J - r| \leq \delta_J$, so $\{|s_J - c| > \delta_J\} \subseteq \{r \neq c\}$ and $\Pr(|s_J - c| > \delta_J) \leq \Pr(r \neq c) \leq \varepsilon$. □

Theorem F.8 (Actor objective aligns initial correctness in Actor–Editor)**.** *Under Assumptions F.2 and F.5, for all π_A,*

$$\Big| J_A(\pi_A) - (1 + \lambda_J)\, \mathrm{C}_{\mathrm{init}}(\pi_A) \Big| \;\leq\; (1 + \lambda_J)\varepsilon + \lambda_J \delta_J. \tag{56}$$

Consequently, any maximizer $\pi_A^\star \in \arg\max_{\pi_A} J_A(\pi_A)$ satisfies

$$\mathrm{C}_{\mathrm{init}}(\pi_A^\star) \;\geq\; \sup_{\pi_A} \mathrm{C}_{\mathrm{init}}(\pi_A) \;-\; \frac{2\big((1 + \lambda_J)\varepsilon + \lambda_J \delta_J\big)}{1 + \lambda_J}. \tag{57}$$

Proof. By definition,

$$J_A(\pi_A) = \mathbb{E}[r(X, Y_0)] + \lambda_J\, \mathbb{E}[s_J(X, Y_0)].$$

Subtract $(1 + \lambda_J)\mathbb{E}[c(X, Y_0)]$ and apply triangle inequality:

$$\Big| J_A - (1 + \lambda_J)\mathrm{C}_{\mathrm{init}} \Big| \leq \big|\mathbb{E}[r - c]\big| + \lambda_J \big|\mathbb{E}[s_J - c]\big| \leq \mathbb{E}|r - c| + \lambda_J\, \mathbb{E}|s_J - c|.$$

Assumption F.2 gives $\mathbb{E}|r - c| = \Pr(r \neq c) \leq \varepsilon$, and Lemma F.7 gives $\mathbb{E}|s_J - c| \leq \delta_J + \varepsilon$, yielding (56). Apply Lemma F.4 with $a = 1 + \lambda_J$, $b = 0$, and $\Delta = (1 + \lambda_J)\varepsilon + \lambda_J \delta_J$ to obtain (57). □

Editor monotonicity for final solvability. Let the Editor iteratively refine Y_t for $t \leq T_{\max}$. Define

$$R_{\mathrm{final}} \;\coloneqq\; \mathbb{I}\{\exists t \leq T_{\max} : r(X, Y_t) = 1\}. \tag{58}$$

Lemma F.9 (Repair is weakly monotone in solver success)**.** *If the Editor action space includes a no-op (i.e., it can keep $Y_{t+1} = Y_t$), then for any fixed Actor policy π_A and any Editor policy π_E,*

$$\mathbb{E}[R_{\mathrm{final}}] \;\geq\; \mathbb{E}[r(X, Y_0)]. \tag{59}$$

Proof. Pointwise, the event $\{r(X, Y_0) = 1\}$ implies $\{\exists t \leq T_{\max} : r(X, Y_t) = 1\}$, since the Editor can keep $Y_t = Y_0$. Thus $R_{\mathrm{final}} \geq r(X, Y_0)$ almost surely, and taking expectations yields (59). □

Theorem F.10 (Why Actor–Editor)**.** *Assume Assumption F.2.*

1. *(**Actor-only**) Maximizing $J_{\mathrm{A-only}}$ yields an initial-correctness guarantee within 2ε of the optimum (Theorem F.5).*
2. *(**Editor-only**) In the saturated-repair regime (Assumptions F.3 and F.4), the monolithic objective admits globally optimal policies with arbitrarily low $\mathrm{C}_{\mathrm{init}}$ (Theorem F.6).*
3. *(**Actor–Editor**) With a calibrated Judge (Assumption F.5), maximizing the Actor objective J_A enforces initial-correctness alignment up to $\mathcal{O}(\varepsilon + \delta_J)$ (Theorem F.8), while the Editor weakly improves final solvability (Lemma F.9).*

F.4 Scaling of Local Reward Hacking: Separate vs. Shared Parameters

This subsection provides a proof of the scaling claim in Theorem 3.2 with the *worst-case first-order reward-hacking gain* under a local linearization.

Local linearization and hacking gain. Fix a reference point θ_0 and a constant $a > 0$. Let $J_A(\theta)$ denote the Actor objective and $C(\theta)$ the (true) plan-correctness objective. Define the *reward-hacking gradient*

$$g_{\text{hack}}(\theta_0) \;\coloneqq\; \nabla_\theta J_A(\theta_0) - a\,\nabla_\theta C(\theta_0). \tag{60}$$

For a perturbation $\Delta\theta$ with $\|\Delta\theta\|_2 \le 1$, the first-order (local) hacking gain is

$$\Delta_{\text{hack}}(\theta_0; \Delta\theta) \;\coloneqq\; g_{\text{hack}}(\theta_0)^\top \Delta\theta. \tag{61}$$

Given an admissible perturbation set $\mathcal{H}$, the worst-case local hacking gain is

$$\sup_{\Delta\theta \in \mathcal{H}} \Delta_{\text{hack}}(\theta_0; \Delta\theta). \tag{62}$$

Architectures and admissible perturbation sets.

- **Separate models.** The Actor has its own parameters $\theta_A \in \mathbb{R}^{p_A}$. A unit-norm perturbation can move freely in the Actor space:

$$\mathcal{H}_{\text{sep}} \;\coloneqq\; \big\{\Delta\theta_A \in \mathbb{R}^{p_A} :\; \|\Delta\theta_A\|_2 \le 1\big\}. \tag{63}$$

- **Shared backbone.** Parameters decompose as

$$\theta = (\theta_{\text{sh}}, \theta_A, \theta_J, \theta_E),$$

  where $\theta_{\text{sh}} \in \mathbb{R}^p$ is the shared backbone, and $\theta_A \in \mathbb{R}^h$ is the Actor head (with $h \ll p$). We define admissible perturbations as those that are *(Judge,Editor)-neutral on the backbone* and do not modify the Judge/Editor heads:

$$\mathcal{H}_{\text{shr}} \;\coloneqq\; \Big\{\Delta\theta :\; \|\Delta\theta\|_2 \le 1,\; \Delta\theta_J = 0,\; \Delta\theta_E = 0,\; \Delta\theta_{\text{sh}} \in \text{Null}(\Sigma_{JE})\Big\}, \tag{64}$$

  where $\Sigma_{JE} \in \mathbb{R}^{p\times p}$ is defined in Assumption F.6 below.

Assumption F.6 (Judge and Editor constrain the backbone (covariance form))**.** *At θ_0, let $g_J \in \mathbb{R}^p$ and $g_E \in \mathbb{R}^p$ denote the (random) backbone gradients associated with the Judge and Editor objectives under their respective training distributions (e.g., per-sample or per-trajectory stochastic gradients). Define the (backbone) second-moment / Gram matrix*

$$\Sigma_{JE} \;\coloneqq\; \mathbb{E}\big[g_J g_J^\top\big] + \mathbb{E}\big[g_E g_E^\top\big] \in \mathbb{R}^{p\times p}. \tag{65}$$

Assume Σ_{JE} is full rank on the backbone, equivalently

$$\text{Null}(\Sigma_{JE}) = \{0\}. \tag{66}$$

Assumption F.6 is a high-rank *coverage* condition: Judge and Editor gradients collectively "see" all backbone directions (in second moment), so there is no nonzero backbone direction that is simultaneously neutral to both.

Assumption F.7 (Isotropic hacking gradient)**.** *There exists $\sigma > 0$ such that the restriction of $g_{\text{hack}}(\theta_0)$ to the relevant parameter subspace is isotropic Gaussian. Concretely:*

1. *In the* separate *setting, $g_{\text{hack}}(\theta_0) \in \mathbb{R}^{p_A}$ satisfies*

$$g_{\text{hack}}(\theta_0) \sim \mathcal{N}(0, \sigma^2 I_{p_A}). \tag{67}$$

2. *In the* shared *setting, the projection of $g_{\text{hack}}(\theta_0)$ onto the Actor-head subspace $\mathbb{R}^h$ satisfies*

$$\text{Proj}_A\, g_{\text{hack}}(\theta_0) \sim \mathcal{N}(0, \sigma^2 I_h). \tag{68}$$

A Gaussian supremum lemma

Lemma F.11 (Supremum of an isotropic Gaussian over a unit ball)**.** *Let $g \sim \mathcal{N}(0, \sigma^2 I_d)$ in $\mathbb{R}^d$ and let $\mathbb{B}_d := \{u \in \mathbb{R}^d : \|u\|_2 \le 1\}$. Then*

$$\mathbb{E}\Big[\sup_{u \in \mathbb{B}_d} g^\top u\Big] = \mathbb{E}\big[\|g\|_2\big] = \sigma\,\mathbb{E}\big[\|Z\|_2\big], \qquad Z \sim \mathcal{N}(0, I_d). \tag{69}$$

Moreover, for all $d \ge 1$,

$$\sigma\sqrt{d-1} \;\le\; \mathbb{E}\big[\|g\|_2\big] \;\le\; \sigma\sqrt{d}, \tag{70}$$

and in particular $\mathbb{E}[\|g\|_2] = \Theta(\sigma\sqrt{d})$.

Proof. For any fixed g, the Cauchy–Schwarz inequality gives $\sup_{\|u\| \le 1} g^\top u = \|g\|_2$, achieved by $u = g/\|g\|_2$ if $g \neq 0$. Taking expectation yields the first equality in (69).

The upper bound in (70) follows from Jensen: $\mathbb{E}\|g\|_2 \le \sqrt{\mathbb{E}\|g\|_2^2} = \sqrt{\mathbb{E}[g^\top g]} = \sigma\sqrt{d}$. The lower bound $\mathbb{E}\|Z\|_2 \ge \sqrt{d-1}$ for $Z \sim \mathcal{N}(0, I_d)$ is classical (equivalently for a χ_d random variable), and follows for example from standard gamma-function bounds for $\mathbb{E}\chi_d = \sqrt{2}\,\Gamma\big(\frac{d+1}{2}\big)/\Gamma\big(\frac{d}{2}\big)$. Multiplying by σ yields (70). □

Main result

Theorem F.12 (Scaling of local reward-hacking gain: separate vs. shared)**.** *Under Assumptions F.6 and F.7:*

1. ***Three separate models.*** *With $\mathcal{H}_{\text{sep}}$ as in (63),*

$$\mathbb{E}\Big[\sup_{\Delta\theta \in \mathcal{H}_{\text{sep}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta)\Big] \;=\; \Theta\big(\sigma\sqrt{P_A}\big). \tag{71}$$

2. ***One shared backbone.*** *With $\mathcal{H}_{\text{shr}}$ as in (64),*

$$\mathbb{E}\Big[\sup_{\Delta\theta \in \mathcal{H}_{\text{shr}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta)\Big] \;\le\; \sigma\sqrt{h}, \tag{72}$$

   hence the expected local reward-hacking gain is $\mathcal{O}(\sigma\sqrt{h})$ and does not scale with the backbone dimension p.

Proof. We prove the two items.

(1) Separate models. In the separate setting, $\Delta\theta$ ranges over the Actor parameter space $\mathbb{R}^{P_A}$, and by (61)–(63),

$$\sup_{\Delta\theta \in \mathcal{H}_{\text{sep}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta) = \sup_{\|\Delta\theta\|_2 \le 1} g_{\text{hack}}(\theta_0)^\top \Delta\theta = \|g_{\text{hack}}(\theta_0)\|_2.$$

By Assumption F.7 (separate case) and Lemma F.11 with $d = P_A$,

$$\mathbb{E}\Big[\sup_{\Delta\theta \in \mathcal{H}_{\text{sep}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta)\Big] = \mathbb{E}\|g_{\text{hack}}(\theta_0)\|_2 = \Theta(\sigma\sqrt{P_A}),$$

proving (71).

(2) Shared backbone. By Assumption F.6, $\text{Null}(\Sigma_{JE}) = \{0\}$, hence $\Delta\theta_{\text{sh}} = 0$ for all $\Delta\theta \in \mathcal{H}_{\text{shr}}$ by (64). Moreover $\Delta\theta_J = \Delta\theta_E = 0$ by definition of $\mathcal{H}_{\text{shr}}$. Therefore any $\Delta\theta \in \mathcal{H}_{\text{shr}}$ lies entirely in the Actor-head subspace, so we can write $\Delta\theta = (0, \Delta\theta_A, 0, 0)$ with $\|\Delta\theta_A\|_2 \le 1$. Thus,

$$\sup_{\Delta\theta \in \mathcal{H}_{\text{shr}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta) = \sup_{\|\Delta\theta_A\|_2 \le 1} \big(\text{Proj}_A g_{\text{hack}}(\theta_0)\big)^\top \Delta\theta_A = \big\|\text{Proj}_A g_{\text{hack}}(\theta_0)\big\|_2.$$

By Assumption F.7 (shared case), $\text{Proj}_A g_{\text{hack}}(\theta_0) \sim \mathcal{N}(0, \sigma^2 I_h)$. Applying Lemma F.11 with $d = h$ yields

$$\mathbb{E}\Big[\sup_{\Delta\theta \in \mathcal{H}_{\text{shr}}} \Delta_{\text{hack}}(\theta_0; \Delta\theta)\Big] = \mathbb{E}\big\|\text{Proj}_A g_{\text{hack}}(\theta_0)\big\|_2 \le \sigma\sqrt{h},$$

which is (72). The bound depends on h but not on p. □

Remark (effective-rank variant). If $\mathrm{Null}(\Sigma_{JE})$ has dimension $k > 0$ (i.e., Judge+Editor do not constrain k backbone directions), then the same proof yields a shared-architecture bound of order $\sigma\sqrt{h+k}$, where k is the number of unconstrained backbone directions. This makes explicit that the scaling depends on the *dimension of the (Judge,Editor)-neutral subspace*, not directly on the backbone size p.

G Detailed Experimental Setup

G.1 Benchmarks

PlanBench. PlanBench (Valmeekam et al., 2023) evaluates whether language models can solve classical planning problems described in natural language while preserving the underlying symbolic transition structure. Each instance is derived from a PDDL planning problem and converted into a natural-language prompt. We report results on four domains: BlocksWorld, Mystery BlocksWorld, Logistics, and Gripper. BlocksWorld tests basic object manipulation and state-transition reasoning. Mystery BlocksWorld preserves the same transition dynamics but systematically renames objects and predicates, thereby removing lexical cues and testing whether the model recovers the underlying symbolic structure. Logistics and Gripper introduce more brittle object typing, movement constraints, and action-schema dependencies.

Zero-shot transfer benchmarks. To evaluate whether solver-grounded training improves planning behavior beyond the PlanBench domains, we additionally test zero-shot transfer on ProntoQA (Saparov and He, 2023) and NATURAL PLAN (Zheng et al., 2024). ProntoQA requires multi-hop logical inference from explicitly stated facts and rules. NATURAL PLAN evaluates realistic natural-language planning under constraints; we use the Trip Planning and Calendar Scheduling subsets. In NATURAL PLAN, all external information needed for planning, such as flight connectivity or calendar availability, is provided in-context, so performance reflects planning and constraint satisfaction rather than tool invocation.

G.2 Evaluation Protocol

PDDL-based evaluation. For PlanBench, each model output is parsed as a PDDL domain–problem pair. We use Fast Downward with a 60-second timeout as the external planner. An instance is counted as successful only when both generated files are syntactically valid, the planner returns a plan within the timeout, and the plan satisfies the stated goal conditions. Outputs that fail parsing, grounding, type checking, or goal achievement are counted as failures.

Non-PDDL evaluation. For ProntoQA and NATURAL PLAN, we follow the original benchmark protocols and report exact-match success against the gold label or gold plan. We apply the same answer-normalization rules across all methods. No benchmark-specific supervised examples are used for training our method.

G.3 Model and Training Setting

Backbone. Our framework uses Qwen2.5-7B (Qwen et al., 2025) initialized from pretrained weights. The model is conditioned into three roles: Actor, Judge, and Editor. The Actor generates the initial PDDL specification, the Judge predicts a solver-calibrated quality score, and the Editor repairs failed specifications using solver diagnostics.

Annotation-free training. No human-written PDDL demonstrations, gold domain files, gold problem files, or supervised planning traces are used to train our method. The only external feedback used during training is produced by the symbolic planner and its diagnostics. This ensures that the reported performance reflects solver-grounded learning rather than supervised imitation of annotated PDDL.

Inference. At inference time, the Actor first generates an initial specification. If the specification fails solver verification, the Editor performs bounded diagnostic-conditioned repair. Unless otherwise specified, we use the same maximum repair horizon as in training. The system returns the first solver-executable specification found within the repair budget; if no repair succeeds, the final edited specification is submitted for evaluation.

G.4 Baselines

Chain-of-Thought prompting. The CoT baseline uses chain-of-thought prompting (Wei et al., 2022) to elicit intermediate reasoning before producing the final plan or formal specification. We use an out-of-domain prompt and do not provide task-specific PDDL annotations.

Tree-of-Thought search. The ToT baseline follows the search-based reasoning framework of Yao et al. (2023). The language model expands and evaluates intermediate reasoning states using breadth-first search. The search process does not access symbolic execution feedback during thought expansion.

LLM+P. LLM+P (Liu et al., 2023) translates natural-language planning descriptions into PDDL, invokes a classical planner, and then maps the resulting plan back to natural language when needed. In our setup, the baseline uses fixed domain assumptions and in-context PDDL examples, but no additional human-annotated training data.

Backbone choice. The main comparison uses GPT-4o for CoT, ToT, and LLM+P, providing strong off-the-shelf zero-annotation baselines. To separate algorithmic gains from backbone effects and solver-call budgets, we additionally report matched-backbone and matched-budget comparisons in Appendix H.1, where all controlled baselines use Qwen2.5-7B and the same maximum number of solver calls as our method.

H Additional Experimental Details and Discussion

H.1 Fair Baseline Comparison under Matched Backbone and Solver Budget

The main comparison in Table 1 follows prior planning baselines with strong prompting and neuro-symbolic pipelines. However, to rule out confounding factors from different backbone models and different verifier-access budgets, we conduct an additional controlled comparison. All methods in this subsection use the same Qwen2.5-7B backbone as our method and are allowed the same maximum number of solver calls at inference time. Since our default inference procedure consists of one initial Actor generation followed by at most five Editor repairs, the maximum solver-call budget is

$$K = H_{\max} + 1 = 6.$$

For prompting baselines that do not use solver diagnostics, we allow up to K independently decoded candidates and select the first solver-successful output if one exists. For the self-refine baseline, the model receives the raw solver diagnostic after each failed attempt and revises its previous PDDL specification for at most five rounds. This gives all baselines comparable access to solver verification while isolating the effect of trained role specialization.

Controlled baselines. We evaluate four same-backbone baselines:

- **Qwen-CoT** uses chain-of-thought prompting with up to six sampled candidates. The solver is used only for candidate selection, not for textual feedback.
- **Qwen-ToT** uses tree-of-thought search with Qwen2.5-7B as both generator and evaluator. We cap the final solver-checked candidates at six.
- **Qwen-LLM+P** prompts Qwen2.5-7B to translate the natural-language task into PDDL and invokes the planner on each candidate. We allow up to six independently sampled formalizations.
- **Qwen-Self-Refine+Solver** uses the same backbone and the same solver-call budget as our method, but has no trained Actor, Judge, or Editor. It revises its previous PDDL output using the solver diagnostic through prompting alone.

None of these baselines uses supervised PDDL annotations or task-specific training.

Table 8 shows that the advantage of our method is not caused by using a stronger backbone or by having more access to the symbolic solver. When all methods use Qwen2.5-7B and the same maximum solver-call budget, CoT and ToT remain weak, especially on Mystery BlocksWorld and Logistics. This indicates that sampling more reasoning traces or searching over textual thoughts does not reliably recover valid symbolic transition models. Qwen-LLM+P performs well on BlocksWorld but degrades sharply on Mystery BlocksWorld, Logistics, and Gripper, suggesting that in-context PDDL formalization remains sensitive to domain familiarity and brittle action-schema construction.

Table 8: Planning success rate (%) under matched backbone and matched solver-call budget. All Qwen baselines use Qwen2.5-7B and at most $K = 6$ solver calls per test instance. Avg. denotes the mean across the four PlanBench domains.

Method	Backbone	Max Calls	BW	MBW	Logistics	Gripper	Avg.
Qwen-CoT	Qwen2.5-7B	6	31	1	5	25	15.5
Qwen-ToT	Qwen2.5-7B	6	16	5	9	18	12.0
Qwen-LLM+P	Qwen2.5-7B	6	80	32	20	7	34.8
Qwen-Self-Refine+Solver	Qwen2.5-7B	6	85	43	36	35	49.8
GPT-4o-Self-Refine+Solver	GPT-4o	6	91	47	42	37	54.3
Ours	**Qwen2.5-7B**	**6**	**98**	**71**	**58**	**56**	**70.8**

The strongest same-backbone baseline is Qwen-Self-Refine+Solver, which uses the same solver diagnostics available to our Editor at inference time. It improves over Qwen-LLM+P by revising syntactic and type-level mistakes, reaching 49.8% average success. However, it still remains 21.0 percentage points behind our method. This gap suggests that solver diagnostics alone are useful but insufficient: without a trained Judge and diagnostic-conditioned Editor, prompting-based repair often fixes local errors while leaving global predicate structure, goal preservation, or action semantics unstable. The GPT-4o self-refine baseline provides an even stronger untrained reference, but it still underperforms our Qwen2.5-7B method. Therefore, the improvement is better explained by the learned multi-role training procedure rather than by backbone strength or additional solver calls.

Table 9: Budget efficiency and semantic faithfulness under the matched $K = 6$ solver-call setting. Avg. Calls is the average number of solver invocations actually used before success or budget exhaustion. Faithful denotes the fraction of all tasks that are both solver-successful and semantically faithful according to the reference checker. Drift is the conditional fraction of solver-successful outputs that fail semantic checking.

Method	Avg. Calls	Solv. (%)	Faithful (%)	Drift (%) $\downarrow$
Qwen-CoT	5.8 ± 0.4	15.5 ± 2.5	11.6 ± 2.0	25.2 ± 3.0
Qwen-ToT	5.9 ± 0.4	12.0 ± 2.0	9.1 ± 1.5	24.4 ± 3.0
Qwen-LLM+P	5.2 ± 0.3	34.8 ± 3.5	27.1 ± 3.0	22.1 ± 2.5
Qwen-Self-Refine+Solver	4.7 ± 0.3	49.8 ± 4.0	37.5 ± 3.5	24.7 ± 3.0
GPT-4o-Self-Refine+Solver	4.5 ± 0.3	54.3 ± 4.0	43.0 ± 3.5	20.8 ± 2.5
Ours	$\mathbf{3.2 \pm 0.2}$	$\mathbf{70.8 \pm 4.5}$	$\mathbf{66.3 \pm 4.0}$	$\mathbf{6.4 \pm 1.0}$

Table 9 further shows that our method is not simply spending the solver budget more aggressively. Although all methods are allowed up to six solver calls, our method uses only 3.2 calls on average because many instances are solved by the initial Actor output or by early Editor repairs. In contrast, CoT, ToT, and LLM+P frequently exhaust the budget without producing executable specifications. Self-refine baselines use diagnostics more effectively, but their semantic drift remains high: many repaired specifications become solver-executable without fully preserving the intended task semantics. Our method has the smallest gap between solvability and faithful success, and its drift rate is substantially lower than all matched-budget baselines. This supports the default design choice: the learned Judge and Editor do not merely increase the chance of passing the solver, but also stabilize the symbolic specification so that solver success remains aligned with task-level faithfulness.

These controlled comparisons address two possible alternative explanations. First, the gain is not due to backbone choice, since all same-backbone Qwen2.5-7B baselines remain below our method. Second, the gain is not due to a larger verifier budget, since every method is capped at the same $K = 6$ solver calls, and our method uses fewer calls on average. The remaining performance gap is therefore attributable to the trained multi-role decomposition: the Actor learns global PDDL construction, the Judge suppresses solver-facing shortcuts, and the Editor learns targeted diagnostic-conditioned repair.

H.2 Discussion of Main Planning and Transfer Results

Table 1 shows that LLM+P performs strongly on BlocksWorld but drops sharply on Logistics and Gripper. This pattern is consistent with the fact that BlocksWorld is a canonical planning domain frequently appearing in PDDL tutorials and planning examples, whereas Logistics and Gripper require more careful object typing, action preconditions, and movement constraints. Mystery BlocksWorld is especially diagnostic because it preserves the same transition dynamics as BlocksWorld while removing lexical

cues through systematic renaming. The large gain on Mystery BlocksWorld therefore suggests that our method is not simply relying on familiar predicate names, but is learning a solver-grounded mapping from task descriptions to symbolic structure.

For zero-shot transfer, ProntoQA evaluates multi-hop logical inference from explicit facts and rules, while NATURAL PLAN evaluates realistic planning tasks expressed in natural language. The Trip Planning and Calendar Scheduling subsets require satisfying multiple constraints using information provided in-context. The improvement on these benchmarks indicates that the learned roles transfer beyond PDDL syntax: the Actor learns to propose structured solutions, the Judge learns to score consistency, and the Editor learns to refine outputs under constraint feedback.

H.3 Controlled Baseline Protocol

The controlled comparison in Table 2 is designed to isolate the effect of trained role specialization. All methods are allowed at most

$$K = H_{\max} + 1 = 6$$

solver calls per instance, matching our default inference procedure: one Actor generation followed by at most five Editor repairs.

For CoT, ToT, and LLM+P variants without diagnostic repair, we allow up to K independently decoded candidate outputs and select the first solver-successful specification. For self-refine baselines, the model receives the raw solver diagnostic after each failed attempt and revises the previous PDDL specification for up to five rounds. This gives prompting-based repair access to the same type of solver feedback available to our Editor at inference time, but without any role-specific training.

Average solver calls are computed as the number of solver invocations used before either the first successful specification is found or the budget is exhausted. The faithful and drift columns are computed using the same post-hoc reference checker described in Appendix H.4. The result that our method uses 3.2 calls on average while achieving the highest faithful success indicates that the trained Actor and Editor reduce both search cost and semantic drift.

H.4 Semantic Faithfulness Protocol

The standard PlanBench metric evaluates whether the generated PDDL specification is solver-executable. However, solver success alone does not guarantee that the generated specification preserves the original natural-language task. A model may weaken goals, omit constraints, alter object types, or distort action schemas while still producing a solvable PDDL instance. We therefore evaluate semantic faithfulness with a post-hoc reference checker that is used only for evaluation and never for training.

For each task x_i, let

$$y_i = (y_i^{\text{dom}}, y_i^{\text{prob}})$$

be the generated PDDL specification and let $\hat{\pi}_i$ be the plan returned by Fast Downward when y_i is solvable. The solver-success indicator is

$$S_i = \mathbf{1}\{\mathcal{E}(y_i) = 1\}. \tag{73}$$

The reference checker $\mathcal{V}_{\text{ref}}$ verifies three conditions:

1. **Goal preservation:** generated goal conditions are semantically equivalent to the reference goal after canonicalizing object and predicate names;
2. **Object and type preservation:** generated objects and type assignments preserve the entity structure of the original task;
3. **Action-schema consistency:** generated action schemas preserve the reference transition semantics, measured by canonical precondition/effect matching and plan replay under the reference transition model.

The faithful-success indicator is then

$$C_i = \mathbf{1}\{S_i = 1 \wedge \mathcal{V}_{\text{ref}}(x_i, y_i, \hat{\pi}_i) = 1\}. \tag{74}$$

We report

$$\text{Solvability} = \frac{1}{N}\sum_{i=1}^{N} S_i, \qquad \text{FaithfulSuccess} = \frac{1}{N}\sum_{i=1}^{N} C_i, \tag{75}$$

and the conditional drift rate

$$\text{Drift} = \frac{\sum_{i=1}^{N} S_i(1 - C_i)}{\sum_{i=1}^{N} S_i}. \tag{76}$$

A lower drift rate means that fewer solver-successful outputs are achieved through semantic shortcuts.

H.5 Discussion of Faithfulness and Drift Results

The faithfulness results in Table 3 show that solver-only RL increases raw solvability but also creates substantial specification drift. This confirms the central concern that raw solver success is an exploitable proxy when the model controls the generated formal specification. In contrast, the full model improves solvability and faithful success simultaneously, while maintaining a much lower drift rate.

The per-domain faithful-success results show that the advantage is not restricted to easier domains. On Mystery BlocksWorld, where lexical cues are removed, our method retains 67% faithful success. On Logistics and Gripper, faithful success remains lower because small mistakes in typing, movement constraints, or precondition/effect structure can distort the intended transition system. Nevertheless, our method remains substantially better than LLM+P and solver-only variants, suggesting that the multi-role design improves specification quality rather than merely increasing solver executability.

H.6 Ablation and Drift-Type Discussion

The ablation results in Table 4(a) isolate the contribution of each role on Mystery BlocksWorld. The Actor-only setting collapses because the policy must search a large symbolic specification space using sparse solver feedback. Adding the Judge without the Editor improves success but remains limited because the system lacks a targeted repair mechanism. Adding the Editor without the Judge also fails, suggesting that diagnostic repair alone can still drift toward solver-facing shortcuts. The full model succeeds because the Actor handles global formalization, the Judge supplies a calibrated quality signal, and the Editor performs bounded local correction.

The separate-model baseline with 3×7B parameters also collapses. This result is consistent with the analysis in Section 3.7: fully separate role models expose more role-private directions through which the Actor can improve solver-facing reward without being constrained by Judge calibration or Editor repair behavior. The shared-backbone design reduces these degrees of freedom and stabilizes cross-role credit assignment.

Table 4(b) categorizes the remaining unfaithful but solver-successful outputs. LLM+P and solver-only RL contain many direct shortcut failures, especially goal weakening and object/type drift. Our method reduces these shortcut-like errors substantially. Among the few remaining drift cases, action-schema drift becomes the largest category, indicating that the residual failures are mostly subtle transition-model mistakes rather than systematic goal removal or object manipulation.

H.7 Repair and Diagnostic Protocol

For each test instance, the Actor first generates an initial PDDL specification y_0, which is verified by the solver. If verification fails, the Editor performs up to $H_{\max} = 5$ sequential repairs, each conditioned on the task, the current specification, and the latest solver diagnostic. Let $S_i^{(h)} \in \{0, 1\}$ denote whether instance i has been solved by step h, where $h = 0$ corresponds to the initial Actor output and $h \in \{1, \ldots, 5\}$ denotes the number of Editor repairs. We report the cumulative success rate

$$\text{CSR}(h) = \frac{1}{N} \sum_{i=1}^{N} S_i^{(h)}, \qquad h = 0, 1, \ldots, 5, \tag{77}$$

and the marginal repair gain

$$\Delta(h) = \text{CSR}(h) - \text{CSR}(h - 1), \qquad h \geq 1. \tag{78}$$

Diagnostics are grouped into six categories: *syntax/parse error*, *object/type mismatch*, *predicate/schema mismatch*, *precondition/effect mismatch*, *goal/constraint drift*, and *unreachable planning search*. The diagnostic heatmap in Figure 3 reports the dominant diagnostic type among remaining unsolved cases after each repair step.

Averaged across domains, the initial Actor solves 46.2% of instances. Cumulative success rises to 56.5%, 63.5%, 67.5%, 69.3%, and 70.8% after one to five repairs. Thus, the Editor contributes 24.6 absolute points beyond the initial Actor output. The first two repairs account for 17.3 points, or about 70% of the total repair gain, which supports using a bounded repair horizon rather than an unbounded search loop.

H.8 Solvability–Faithfulness Gap Discussion

The solvability–faithfulness gap is defined as

$$\text{Gap} = \text{Solvability} - \text{FaithfulSuccess}. \tag{79}$$

This metric measures how tightly solver success aligns with task-level semantic correctness. A large gap means that many solver-successful outputs are semantically unfaithful.

Figure 4 shows that our method has the smallest gap. The Qwen-Self-Refine+Solver baseline improves raw solvability by using solver diagnostics, but its faithful success remains much lower, indicating that prompting-based repair often fixes local syntax or type errors without preserving the deeper transition semantics. The GPT-4o self-refine baseline has a similar issue: stronger language modeling improves executable output generation, but does not fully suppress specification drift. The smaller gap of our method suggests that the Actor, Judge, and Editor jointly maintain a closer connection between executable PDDL and the original natural-language task.

H.9 Judge Candidate Construction and Interpretation

To evaluate whether the Judge distinguishes faithful outputs from solver-successful but unfaithful outputs, we collect a candidate pool from model-generated specifications during inference. The pool includes initial Actor outputs, intermediate Editor repairs, and final returned specifications. Each candidate y is assigned two post-hoc labels:

$$S(y) \in \{0,1\}, \qquad C(y) \in \{0,1\},$$

where $S(y) = 1$ means the specification is solver-executable, and $C(y) = 1$ means the specification is both solver-executable and semantically faithful under the reference checker.

We partition candidates into three groups:

$$\begin{aligned} \mathcal{U} &= \{y : S(y) = 0\}, \\ \mathcal{D} &= \{y : S(y) = 1,\ C(y) = 0\}, \\ \mathcal{F} &= \{y : S(y) = 1,\ C(y) = 1\}, \end{aligned} \tag{80}$$

corresponding to unsolved, solved-but-unfaithful, and solved-and-faithful outputs. For each candidate, we record the Judge score $s_J(x, y) \in [0, 1]$.

If the Judge only predicted raw solver acceptance, solved-but-unfaithful and faithful outputs would receive similar scores because both pass the solver. Instead, Figure 5 and Table 5 show a clear ordering:

$$\mathcal{U} < \mathcal{D} < \mathcal{F}.$$

This indicates that the Judge captures structural quality signals correlated with semantic faithfulness, even though it is not trained with human semantic labels. The separation is strongest on BlocksWorld and weaker on Logistics and Gripper, where subtle action-schema errors can still pass solver checks under distorted specifications.