Automated Skill Optimization via Formal Verification for Embodied Agents

Yunhao Yang*[1] Neel P. Bhatt*[1] Kevin Wang[1] Zhangyang Wang[1] Ufuk Topcu[1]
[1] The University of Texas at Austin *Equal Contribution
Austin, Texas, USA
yunhaoyang234@utexas.edu

Abstract

Skill abstractions are high-level modules that enable embodied agents to compose complex behaviors from reusable components. Recent foundation models further extend this planning approach by generating skills from natural language, improving flexibility and accessibility. However, such skills lack formal guarantees, as their behaviors may violate safety or task constraints, limiting reliability in real-world deployment. We formalize verified skill synthesis, the problem of expanding a skill library while preserving global safety specifications expressed in temporal logic. Each skill consists of a formal local rule and a natural language contract, both produced by a foundation model. The contract serves as the planner-facing representation of the skill and the optimization target for improving plan quality. We introduce VASO, a closed-loop optimization process that verifies each local rule against the global specifications and refines the contract using model-checking feedback to improve specification compliance, without updating model weights. Evaluations on the Jackal ClearPath ground robot and a PX4 quadcopter show that plans guided by the optimized skills achieve 97.2% compliance with formal specifications using fewer than 100 optimization samples and under 20 minutes of optimization per skill.

1. Introduction

Embodied agents deployed in real-world environments often execute long-horizon tasks while satisfying safety and task-specific constraints. Skill abstractions, high-level representations (e.g., text-based instructions) encoding low-level control policies, offer a natural way to manage this complexity: they decompose behavior into reusable, high-level modules such as navigation, obstacle avoidance, and object interaction [1, 12]. By composing skills, agents can reuse structured behaviors across tasks while preserving a modular and interpretable planning interface.

Foundation models further expand this planning approach by translating natural language instructions into executable plans or robot programs that invoke available skills [6, 10, 17]. The use of foundation models reduces manual engineering and makes embodied-agent programming more accessible. However, this flexibility comes without formal guarantees: generated behaviors may violate safety constraints or fail task requirements, making them difficult to deploy in safety-critical embodied systems.

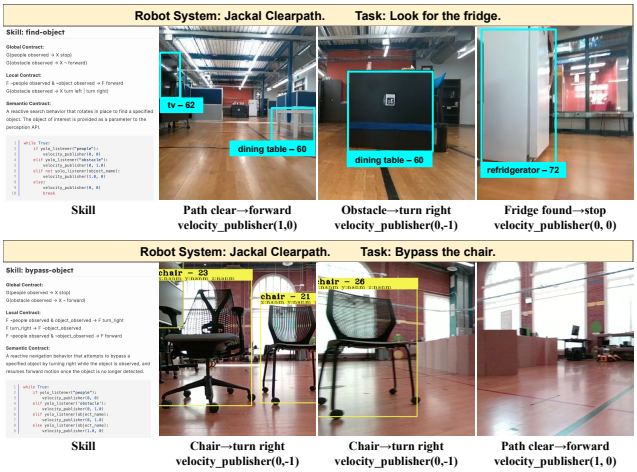

Figure 1. Real-world executions on the Jackal ground robot. We learn different new skills and complete various tasks while satisfying the externally provided safety specifications.

To address this limitation, we develop a framework for automated and verifiable skill optimization in embodied agents. Our key observation is that a generated skill should expose both a formal interface and a language interface. The formal interface specifies the intended behavior of the skill and enables verification against global safety requirements. The language interface instructs the planner how to generate executable behaviors. We therefore represent each skill with a formal local rule and a natural language instruction, which we define as semantic contract, under global temporal logic specifications provided by the system designer. This separation allows us to verify skill-level consistency before planning and then optimize the semantic

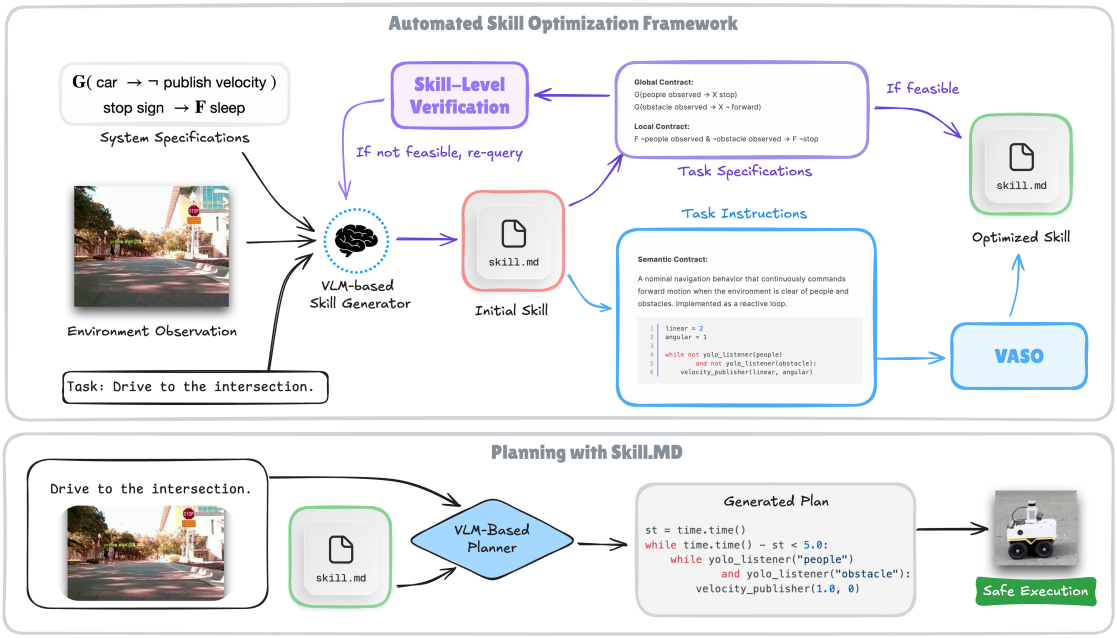

Figure 2. Overview of the automated skill optimization framework. We feed a task prompt and obtain a skill, verified for feasibility, used to generate plans, and iteratively refined through verification-guided optimization.

contract that the planner actually uses.

We formalize this setting as *verified skill synthesis*, the problem of expanding a language-generated skill library while preserving global safety specifications expressed in temporal logic. Building on this formulation, we introduce **Verification-guided Automated Skill Optimization (VASO)**, a closed-loop process that uses model checking at two levels. At the skill level, VASO verifies each local rule against the global specifications and regenerates inconsistent candidates. At the plan level, VASO verifies generated plans against both global and local specifications, converts counterexamples into textual feedback, and refines the semantic contract without updating model weights.

We evaluate our framework on real robotic platforms, including the Jackal ClearPath ground robot and a PX4 quadcopter. Optimized skills achieve over 97% compliance with formal specifications using fewer than 100 training samples and under 20 minutes per skill, demonstrating efficient and reliable skill acquisition.

Contributions.

- **Verifiable skill abstraction.** A contract-based skill representation that is formally verifiable.
- **Two-level verification.** A unified framework to ensure skill-level consistency and the constraint compliance of the plans guided by the skill.
- **VASO.** A verification-guided loop that refines skills without model fine-tuning.

2. Related Work

LLM-Based Planning. LLMs are increasingly used as planners for embodied agents, generating executable action sequences [10, 17], code-based robot policies [6, 12], or structured problem formulations for classical solvers [5, 19]. Reasoning-augmented strategies improve multi-step planning through intermediate reasoning and verbal self-reflection [16, 20, 24]. SayCan [1] grounds plan selection in physical feasibility by scoring actions against learned accordance functions, but relies on a fixed library of pre-trained skills that cannot be extended or refined. These methods produce or select plans at execution time. None generate reusable skill representations that can be improved or verified independently of any particular plan.

A separate line of work improves LLM planning quality through optimization. textual gradient methods back-propagate natural-language feedback to refine prompts [11, 25, 26]. At the parameter level, RLHF [18], DPO [14], and simulator-augmented fine-tuning [9] adapt model weights directly. However, these methods lack formal guarantees on the generated plans, and violations of safety or task constraints are detected only at execution time, if at all.

Formal Methods for LLM-Based System. Recent work has begun integrating formal verification with LLM-based planning. These works develop algorithms that converts non-verifiable LLM-generated outputs, e.g., natural language or executable code, into verifiable representations such as finite state automaton [7, 8, 13]. Complementary efforts address uncertainty-aware planning with for-

mal guarantees [2] and multimodal grounding for verifiable decision-making [22].

Furthermore, several works utilize formal verification outcomes as feedback to refine the LLM by fine-tuning the parameters (RLVF [21]) or optimizing input prompts (LAD-VF [23]). Our work differs by introducing a contract-based skill abstraction that separates formal constraints from natural language generation interfaces, enabling verification at both the skill and plan levels and supporting efficient closed-loop optimization without model fine-tuning.

3. Problem Formulation

We consider an embodied agent operating over a set of atomic propositions $\mathcal{AP} = \{a_1, \ldots, a_n\}$. Each proposition $a \in \mathcal{AP}$ is associated with an executable API that either (i) produces observations (e.g., `obstacle_observed`) or (ii) triggers actions (e.g., `forward`, `stop`). A system execution is represented as a trace

$$\sigma = \sigma_0, \sigma_1, \sigma_2, \ldots, \quad \sigma_t \subseteq \mathcal{AP},$$

where σ_t denotes the set of propositions that hold at time t.

Skill definition. A skill s is defined as a tuple $s = (\mathcal{G}, \psi_s, \mathcal{C}_s)$, where:

- $\mathcal{G} = \{\varphi_1, \ldots, \varphi_m\}$ is a set of *global specifications*, given as LTL formulas over $\mathcal{AP}$,
- ψ_s is a *local rule*, given as an LTL formula over $\mathcal{AP}$,
- $\mathcal{C}_s$ is a *semantic contract*, expressed in natural language.

The global specifications define environment-wide constraints that must always be satisfied:

$$\sigma \models \mathcal{G} \iff \forall i,\ \sigma \models \varphi_i.$$

The local rule ψ_s specifies the intended behavior of the skill, while the semantic contract $\mathcal{C}_s$ serves as an instruction for plan generation.

Planning pipeline. We consider a pipeline that maps a user task prompt to executable behavior:

$$\text{task prompt } \tau \ \rightarrow\ \text{skill } s \ \rightarrow\ \text{plan } \pi_s \ \rightarrow\ \text{execution } \sigma.$$

Given a task prompt τ, a foundation model generates a skill $s = \mathcal{F}_{\text{skill}}(\tau)$. The semantic contract $\mathcal{C}_s$ is then used to generate a plan $\pi_s = \mathcal{F}_{\text{plan}}(\mathcal{C}_s, \tau)$.

Plan verification. The plan is compiled into a transition system $\mathcal{A}_{\pi_s} = (S, S_0, T, L)$, where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, $T \subseteq S \times S$ is the transition relation, and $L : S \rightarrow 2^{\mathcal{AP}}$ is the labeling function.

If there exists a trace σ induced from $\mathcal{A}_{\pi_s}$ satisfies both global and local specifications $\sigma \models \mathcal{G} \wedge \psi_s$, then we claim the local rule is feasible, i.e., not conflicting to the global specifications. A plan is correct if all its executions satisfy the specifications:$\mathcal{A}_{\pi_s} \otimes \mathcal{M} \models \mathcal{G} \wedge \psi_s$, where $\mathcal{M}$ is a transition system that captures all possible system executions.

Problem statement. Given atomic propositions $\mathcal{AP}$, global specifications $\mathcal{G}$, and a task prompt τ, our goal is to automatically construct a skill $s = (\mathcal{G}, \psi_s, \mathcal{C}_s)$ such that the plan generated from $\mathcal{C}_s$ satisfies

$$\mathcal{A}_{\pi_s} \otimes \mathcal{M} \models \mathcal{G} \wedge \psi_s,$$

while the skill itself is *feasible*, i.e.,

$$\exists \sigma \ \ (\sigma \models \mathcal{G} \wedge \psi_s).$$

We aim to produce skills that are both logically consistent with global specifications and capable of inducing plans whose executions satisfy the desired constraints.

4. Automated Skill Optimization

Given a task prompt τ, we generate a skill

$$s = (\mathcal{G}, \psi_s, \mathcal{C}_s) = \mathcal{F}_{\text{skill}}(\tau),$$

where $\mathcal{G}$ is given by the user or system designer, and the model produces a local rule ψ_s and a semantic contract $\mathcal{C}_s$.

Running example. We illustrate with a navigation task on a ground robot.

Listing 1. A sample task prompt for skill generation.

```
Task prompt:
Move forward 5 meters and then turn left.

Safety requirements:
- G (people observed -> X stop)

Output:
1. A local rule in temporal logic
2. A semantic description of the behavior
```

The model outputs a **local rule** ψ_s—an LTL formula encoding the intended behavior—and a **semantic contract** $\mathcal{C}_s$: a natural language instruction used for plan generation. This representation separates symbolic constraints from language-based generation.

4.1. Two-Level Verification

Skill-Level Verification We verify that the generated skill is logically consistent with global specifications. Let

$$\Phi_s = \left(\bigwedge_{\varphi \in \mathcal{G}} \varphi \right) \wedge \psi_s.$$

We check whether there exists a trace σ such that $\sigma \models \Phi_s$.

We construct a universal transition system $\mathcal{M} = (S, S_0, T, L)$, where:

- $S = 2^{\mathcal{AP}}$, i.e., each state corresponds to a valuation of propositions,
- $S_0 = S$,
- $T = S \times S$, i.e., all transitions are allowed,
- $L(s) = s$.

This model represents all possible behaviors over $\mathcal{AP}$.

We reduce the satisfiability problem to model checking by verifying: $\mathcal{M} \models \neg\Phi_s$.

- If the property holds, then all traces satisfy $\neg\Phi_s$, and no satisfying trace exists. The skill is **infeasible**.
- If the property does not hold, the model checker returns a counterexample trace σ such that $\sigma \models \Phi_s$, implying the skill is **feasible**.

If the skill is infeasible (i.e., no satisfying trace exists), we re-query $\mathcal{F}_{\text{skill}}$ with verification feedback to produce a revised skill: $s' = \mathcal{F}_{\text{skill}}(\tau, g)$, where g encodes the cause of infeasibility (e.g., conflicting specifications). This process continues until a feasible skill is obtained. We present an running example below.

Listing 2. An example of feedback-guided skill regeneration.

```
Original local rule:
G(!obstacle observed | person observed ->
     X forward)

Global specification:
G(person observed -> X stop)

Feedback: "The local rule conficts with
    the global specification."

Regenerated local rule:
G(! (obstacle observed | person observed)
     -> X forward)
```

Justification: This skill-level verification checks feasibility by ensuring the existence of a trace satisfying the global and local specifications. This is sufficient for two reasons. First, plans are executed sequentially, so skills do not interact concurrently and cross-skill conflicts do not arise. Second, this step only ensures logical consistency of the specification, while full compliance is verified at the plan level. By filtering out infeasible skills early, it prevents optimization on contradictory specifications.

Plan-Level Verification Given a semantic contract $\mathcal{C}_s$, a plan is generated: $\pi_s = \mathcal{F}_{\text{plan}}(\mathcal{C}_s)$, and compiled into a transition system $\mathcal{A}_{\pi_s}$. We verify correctness via:

$$\mathcal{A}_{\pi_s} \otimes \mathcal{M} \models \mathcal{G} \wedge \psi_s.$$

If the check fails, the model checker returns a counterexample trace indicating a violation of the specifications.

Table 1. Examples of plans in NuSMV and corresponding Python implementations. The function 'max speed' returns a boolean indicating whether the current speed below a given threshold.

NuSMV	**Python**
`next(act) := case TRUE : stop; esac;`	`while True: stop()`
`next(act) := case speed < 1: stop; TRUE : forward; esac;`	`while True: if max_speed(1): stop() else: forward()`

Implementation-wise, we query $\mathcal{F}_{\text{plan}}$ to generate a plan π_s directly in NuSMV[4], a formal language compatible with model checking. The resulting NuSMV program defines the transition relation and labeling over $\mathcal{AP}$, and therefore induces the transition system $\mathcal{A}_{\pi_s}$.

After verification, we convert the same plan into executable code, e.g., Python, using a predefined, fixed rule-based translation strategy. This design ensures that the formally verified symbolic plan and the deployed controller follow the same control logic. We present some sample rule-based translations in Table 1.

4.2. Verification-Guided Automated Skill Optimization (VASO)

We formalize semantic contract refinement as a prompt optimization problem. Given a task prompt τ and a skill $s = (\mathcal{G}, \psi_s, \mathcal{C}_s)$, we generate a plan via a foundation model planner $\mathcal{F}_{\text{plan}}$:

$$\pi_s^{(k)} = \mathcal{F}_{\text{plan}}(\mathcal{C}_s^{(k)}, \tau). \tag{1}$$

Our goal is to optimize the semantic contract such that the generated plan satisfies the specifications:

$$\mathcal{C}_s^* = \arg\min_{\mathcal{C}_s} \mathcal{L}\big(\mathcal{F}_{\text{plan}}(\mathcal{C}_s, \tau)\big), \tag{2}$$

where the loss $\mathcal{L}$ is induced by formal verification feedback.

Textual gradient from verification. Given a plan $\pi_s^{(k)}$, the verifier produces a signal

$$y^{(k)} \in \{\texttt{pass}, \texttt{fail}(\varphi)\},$$

which we convert into a textual gradient $g^{(k)}$, where $g^{(k)}$ takes the form:

```
- violate 1 specification: G(person
    observed -> X stop)
- pass all specifications
```

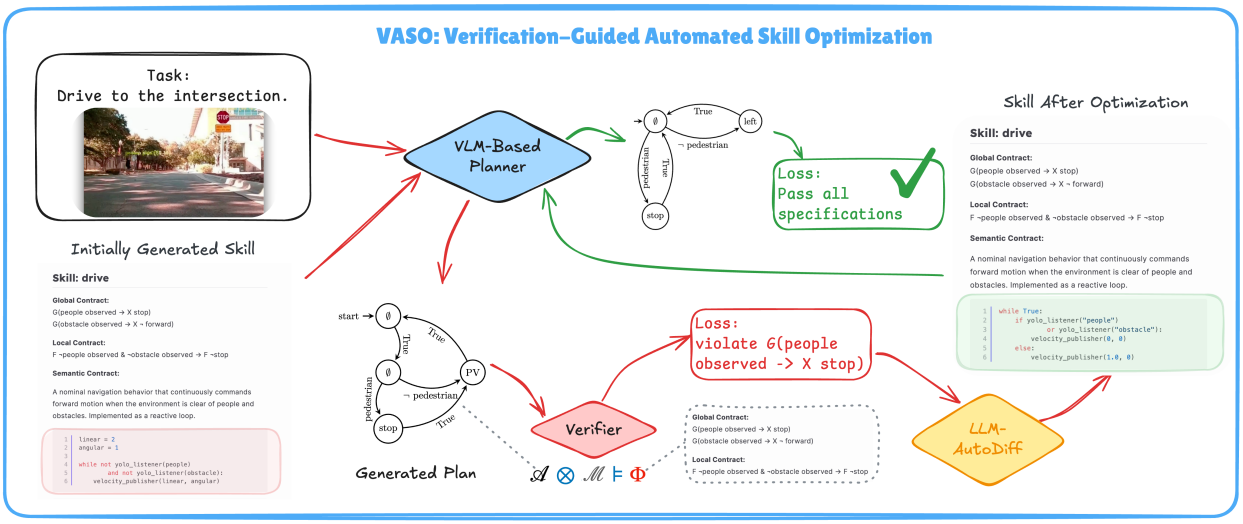

Figure 3. An overview for the verification-guided automated skill optimization pipeline. It verifies plans generated from the semantic contract, and uses the feedback to iteratively refine the contract in a closed loop.

This feedback serves as a discrete approximation of $\frac{\partial \mathcal{L}}{\partial \mathcal{C}_s}$, guiding how the semantic contract should be updated.

Semantic contract update. We update the semantic contract using a foundation model conditioned on the gradient:

$$\mathcal{C}_s^{(k+1)} = \mathcal{F}_{\text{skill}}\left(\mathcal{C}_s^{(k)},\ g^{(k)},\ \tau\right). \tag{3}$$

The overall refinement process alternates between:

$$\text{(Forward)}: \quad \mathcal{C}_s^{(k)} \to \pi_s^{(k)} \to \mathcal{A}_{\pi_s^{(k)}}, \tag{4}$$

$$\text{(Backward)}: \quad \mathcal{A}_{\pi_s^{(k)}} \to g^{(k)} \to \mathcal{C}_s^{(k+1)}. \tag{5}$$

This procedure can be interpreted as a form of *textual gradient descent*, where:

1. the verifier provides a structured loss signal,
2. the textual gradient $g^{(k)}$ approximates the descent direction that minimize specification violation,
3. the language model performs the update step.

Implementation-wise, we adapt `LLM-AutoDiff` [25] to our setting with default parameters.

Importantly, this optimization does not require parameter updates to the foundation model, enabling efficient refinement with minimal data.

5. Demonstration

We demonstrate our automated skill optimization framework on two platforms with distinct dynamics: the *Clearpath Jackal* ground robot and a *PX4* quadrotor drone. In both settings, the system generates skills from high-level tasks, verifies them against formal specifications, and refines them through verification feedback, enabling reliable and compliant behavior without manual skill design.

Algorithm 1 Automated Skill Optimization

Require: Task prompt τ, global specifications $\mathcal{G}$

Generate skill $s = \mathcal{F}_{\text{skill}}(\mathcal{G}, \psi_s, \mathcal{C}_s)$
Construct a universal model $\mathcal{M}$
Verify feasibility of ψ_s via skill-level verification
while violating more than x% of specifications **do**
  Generate plan $\pi_s = \mathcal{F}_{\text{plan}}(\mathcal{C}_s)$
  Verify $\mathcal{A}_{\pi_s} \otimes \mathcal{M} \models \mathcal{G} \wedge \psi_s$
  Obtain feedback g from verification
  Update $\mathcal{C}_s \leftarrow \mathcal{F}_{\text{skill}}\left(\mathcal{C}_s^{(k)},\ g^{(k)},\ \tau\right)$
end while
return optimized skill s

Experimental Setup. Each robot has platform-specific control APIs, perceptual signals, and temporal logic specifications. We use `GPT-5-nano` as our skill generation model $\mathcal{F}_{\text{skill}}$ and `GPT-4o-mini` as our plan generation model $\mathcal{F}_{\text{plan}}$. For the Jackal, we control the robot via velocity commands and object detections, while for the drone, we control it based on velocity in the NED frame and onboard state estimation. The specifications encode global safety constraints that must be satisfied during all the executions. We present the detailed information below, which are the system information we included in the task prompt.

```
=====Jackal (Ground Robot)======
# Atomic Propositions
forward, backward, stop, turn_left, turn_right,
     people_observed, obstacle_observed,......

# APIs
def velocity_publisher(linear, angular):
    """Send linear and angular velocity
```

```
    commands."""
def yolo_listener(object_name) -> bool:
    """Return True if the specified object is
    observed."""

# Specifications
G(people_observed -> X stop)
G(obstacle_observed -> X not forward)
```

```
======PX4 Drone (Aerial Robot)======
# Atomic Propositions
max_altitude, linear_velocity

# APIs
def altitude() -> float:
    """Return current altitude in meters."""
def set_velocity_ned(north, east, down):
    """Set velocity in NED frame."""

# Proposition Mapping
max_altitude := altitude() < 10
linear_velocity := sqrt(north^2 + east^2 + down
    ^2) <= 1

# Specifications
G(max_altitude)
G(linear_velocity)
```

Ground robot navigation. First, consider a simple navigation task on the Jackal platform: *go straight for 5 meters and turn left by 90 degrees*. The framework initially generates a `drive` skill, which passes the skill-level verification, indicating the consistency between the global specifications and the skill-specific rule.

We then pass the skill into the planner $\mathcal{F}_{\text{plan}}$ to produce executable plans. However, the generated plan fails to satisfy the local rule $F(\neg\texttt{people\_observed} \wedge \neg\texttt{obstacle\_observed}) \rightarrow F(\neg\texttt{stop})$, as the robot remains in a stop state indefinitely once a person or obstacle is detected.

We then apply our VASO optimization to refine the semantic contract. After 10 iterations of refinement, the optimized skill satisfies all specifications and enables correct task execution. The initial and refined skills, along with the corresponding plans, are shown in Figure 4, and the execution of the plan guided by the optimized skill is shown in Figure 5.

Aerial navigation. Second, we consider a task on a PX4 quadcopter: *fly in a square-shaped trajectory*. The generated plan guided by the initial skill violates the global specification $G(\texttt{max\_altitude} < 10)$ due to a subtle edge case that can be easily overlooked by human inspection. In particular, when the altitude reaches 10 meters, the drone continues to ascend slightly until the next control command is issued, resulting in a small but critical violation of the safety constraint.

After two iterations of skill optimization, the refined skill satisfies all specifications and executes the task safely. The initial and optimized skills, along with their corresponding plans and executions, are shown in Figures 4 and 5.

These two demonstrations illustrate how the framework systematically identifies and corrects both obvious and subtle violations through verification-guided skill optimization. In the ground navigation task, the framework resolves persistent liveness failures, while in the aerial setting it captures edge-case safety violations that are easily overlooked by human inspection. In both cases, a small number of refinement iterations suffices to obtain skills that satisfy all specifications and enable reliable execution. Additional examples of optimized skills and executions on the Jackal platform are provided in Figure 1.

6. Quantitative Analysis

We quantitatively evaluate the skill optimization framework on both the Jackal ground robot and the PX4 quadcopter. We conduct the evaluation over a test set of 11 temporal logic specifications and 400 generated plans. With minimal training overhead—less than 10 minutes per skill—our framework achieves over 95% specification compliance rate, consistently outperforming current zero-shot planning models and fine-tuning-based planning approaches, while incurring significantly lower computational cost.

Skill Generation and Skill-Level Verification We first evaluate the feasibility of automatically generated skills and the effectiveness of verification-guided regeneration. Starting from raw outputs of $\mathcal{F}_{\text{skill}}$, **73%** of the generated skills satisfy all contracts. An failure example is presented in Listing 2, where the local rule is conflicting to the global specification.

By incorporating verification feedback and re-querying the model, as presented in Listing 2, the feasibility improves to **95%** after one iteration and reaches **99%** after two iterations. These results demonstrate that most invalid skills can be efficiently repaired with minimal interaction, and that the contract-based verification provides strong guidance for rapid convergence.

End-to-End Planning with Verified Skills. We evaluate the effectiveness of verified skills in end-to-end planning using a safety score defined as the average percentage of a generated plan satisfying each specification. Formally, given a plan π and a set of specifications $\{\varphi_i\}_{i=1}^{N}$, the safety score is computed as

$$\text{Safety}(\pi) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\pi \models \varphi_i],$$

and averaged over all generated plans.

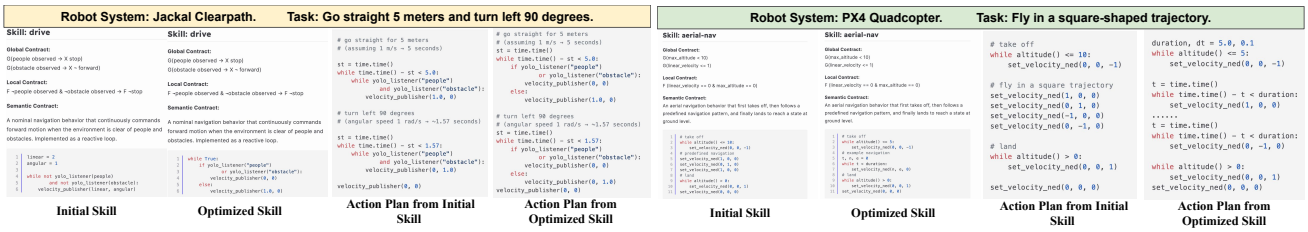

Figure 4. Comparison between initial and refined skills. Refinement improves the generated plans, leading to behaviors that better satisfy the specifications.

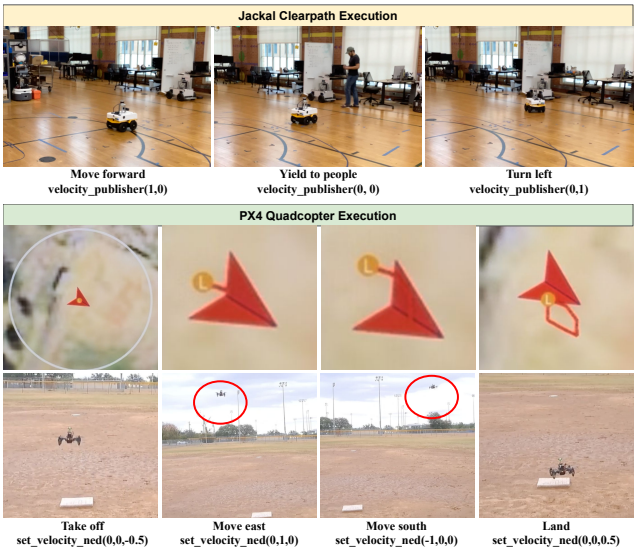

Figure 5. Real-world executions of plans guided by refined skills in Figure 4. The resulting behaviors consistently satisfy task and safety requirements.

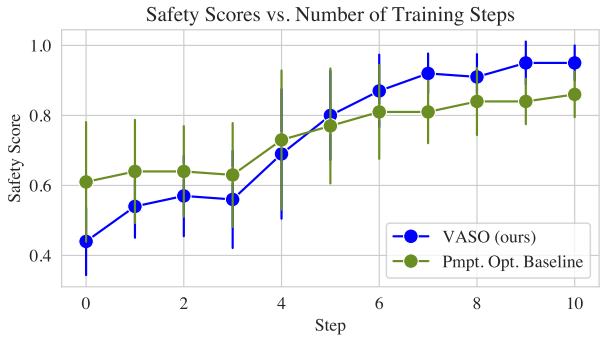

Figure 6. We compare VASO with a baseline prompt optimization method that does not use structured skill representations. VASO rapidly converges to over 90% safety score within 7 steps, while the baseline achieves around 85% after 10 steps.

First, we examine how the safety scores of the generated plans improve over skill optimization iterations. As shown in Figure 6, our VASO framework rapidly increases the safety score, **surpassing 90% within 7 training steps**. In contrast, the baseline prompt optimization method (LLM-AutoDiff [25]), which starts from natural language planning instructions without leveraging the `skill.md` representation, converges more slowly and plateaus at around 85% even after 10 steps. This result highlights that incorporating verifiable skill structure not only improves final performance but also significantly accelerates convergence.

We next evaluate generalization across all specifications. Using 11 temporal logic specifications, which are presented as the global and local contracts in Figure 4 and Figure 1. We generate 100 testing plans from the initial skills and another 100 plans from the optimized skills after 7 training steps. The results, summarized in Figure 7, show that our VASO approach consistently achieves approximately 95% safety score across all specifications and both robotic platforms. This result demonstrates that the optimized skills **generalize reliably across diverse constraints and embodiments**, rather than overfitting to specific tasks or environments. In contrast, plans generated from the initial skills exhibit substantially lower and more variable safety scores, showing the necessity of the optimization process.

Finally, we analyze the efficiency-performance trade-off by varying the training sample size and comparing against multiple baselines, including

- `LAD-VF` [25]: Prompt optimization via textual gradient with 10 steps per sample.
- `RLVF` [21]: Fine-tuning using formal verification feedback for 100 epochs.
- `ICL` [3]: In-context learning with handcrafted program examples that satisfy all the specifications.

As shown in Figure 8, the VASO framework achieves the highest overall safety score at larger sample sizes while maintaining substantially lower training time. Notably, VASO obtains **better performance with orders-of-magnitude less computation** compared to fine-tuning-based methods, which require more training iterations. This highlights the advantage of leveraging structured skill representations and verification feedback, enabling efficient and scalable improvement without expensive retraining.

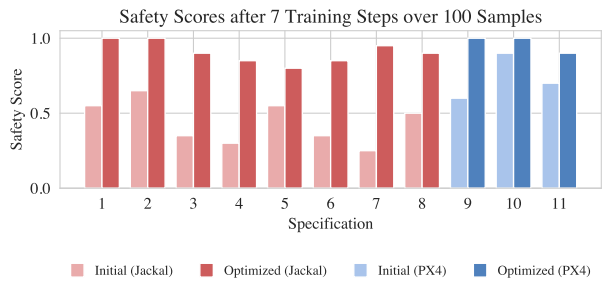

Figure 7. Safety score across 11 specifications from figures 1 and 4. VASO raises the average safety scores from 40% to above 95% safety score across specifications on both platforms.

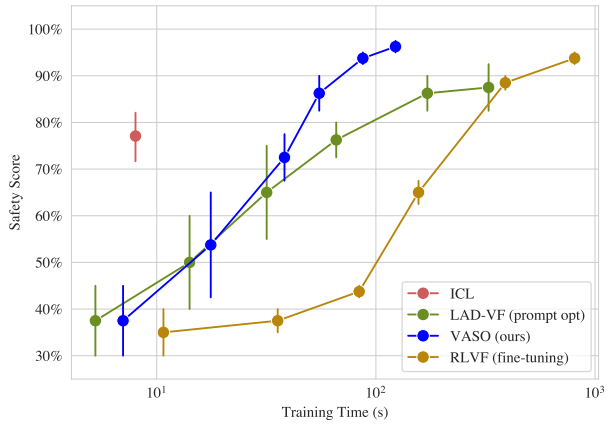

Figure 8. We compare VASO with baseline learning methods for planning tasks. VASO achieves the highest safety score while requiring significantly less training time compared to other learning-based methods.

Planning Baseline Comparison. We compare our framework against a diverse set of planning baselines, including both pretrained end-to-end planners and learning-based approaches. The first group consists of zero-shot planners such as LLM-based planners. These methods directly generate plans from high-level task descriptions without explicit verification or structured skill representations. The second group includes learning-based approaches that adapt planners using data, such as prompt optimization, in-context learning, and fine-tuning methods.

We evaluate all methods using two metrics: (1) **safety score (SS)**, defined as the percentage of specifications satisfied by generated plans, and (2) **task completion rate (TC)**, measuring successful execution of the task. As shown in Table 2, our framework consistently outperforms all baselines across both metrics.

In particular, pretrained planners often achieve reasonable task completion but suffer from low safety compliance, while learning-based methods improve safety at the cost of substantial training overhead. In contrast, our framework achieves the best performance-safety trade-off, achieving the highest safety score and strong task completion without expensive fine-tuning. These results highlight the importance of integrating structured skill representations and formal verification into the planning loop.

Table 2. Planning baseline comparison on safety score (SS) and task completion (TC) rate.

Method	**SS (%)**	**TC (%)**
Pretrained End-to-End Planners		
Direct Query GPT-4o-mini	67.5	77.8
Direct Query GPT-5.4	77.5	85.0
LLM+P [5]	79.5	88.3
ReAct-style Agent [15]	73.2	86.5
Learning-Based Methods		
ICL (few-shot prompting) [3]	83.1	80.9
LLM-AutoDiff (prompt opt) [25]	87.5	81.6
LLM-Planner [17]	90.3	89.3
RoboInstruct [9]	82.5	67.0
RLVF (fine-tuning, 100 epochs) [21]	93.5	91.0
Ours	**97.2**	**85.1**

Ablation. Our results isolate the main components of the framework. Figure 6 demonstrates the necessity of the structured `skill.MD` representation through comparison with a baseline that performs prompt optimization on natural language instructions. Figure 7 reflects the benefit of iterative optimization by showing the gap between initial and optimized skills. Finally, Table 2 captures the role of skill-mediated planning, as opposed to direct end-to-end generation. Together, these results provide a practical ablation of the major components of our framework.

7. Conclusion

We presented an automatic skill optimization framework for foundation model planning. By introducing a structured skill representation, our approach treats skills as both interpretable planning primitives and verifiable units of behavior. Building on this representation, we develop a closed-loop pipeline that formally verifies the plans derived from the skill and refines the skill via the verification feedback. Empirical results show that verification-guided skill optimization significantly outperforms existing LLM-based planning baselines and LLM fine-tuning approaches in both compliance and convergence efficiency. As future directions, we aim to enable **uncertainty-aware skill discovery**, quantifying and reducing perceptual uncertainties during the execution under real-world noise.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 1, 2

[2] Neel P Bhatt, Yunhao Yang, Rohan Siva, Daniel Milan, Zhangyang Wang, and Ufuk Topcu. Know where you're uncertain when planning with multimodal foundation models: A formal framework. In *Eighth Conference on Machine Learning and Systems*, Santa Clara, CA, USA, 2025. 3

[3] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022. 7, 8

[4] Alessandro Cimatti et al. NuSMV 2: An opensource tool for symbolic model checking. In *Computer Aided Verification*, pages 359–364, NY, USA, 2002. Springer. 4

[5] B. Liu et al. Llm+p: Empowering large language models with optimal planning proficiency, 2023. 2, 8

[6] Ishika Singh et al. Progprompt: Generating situated robot task plans using large language models, 2022. 1, 2

[7] Danil S Grigorev, Alexey K Kovalev, and Aleksandr I Panov. Verifyllm: Llm-based pre-execution task plan verification for robots. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 18489–18496. IEEE, 2025. 2

[8] Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. Large language models can solve real-world planning rigorously with formal verification tools. 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 3434–3483, 2025. 2

[9] Zichao Hu, Junyi Jessy Li, Arjun Guha, and Joydeep Biswas. Robo-instruct: Simulator-augmented instruction alignment for finetuning codellms, 2024. 2, 8

[10] Zichao Hu, Francesca Lucchetti, Claire Schlesinger, Yash Saxena, Anders Freeman, Sadanand Modak, Arjun Guha, and Joydeep Biswas. Deploying and evaluating llms to program service mobile robots. *IEEE Robotics Autom. Lett.*, 9 (3):2853–2860, 2024. 1, 2

[11] Omar Khattab, Arnav Singhvi, et al. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. 2

[12] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE, 2023. 1, 2

[13] Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. Grounding complex natural language commands for temporal tasks in unseen environments. In *Conference on Robot Learning*, pages 1084–1110. PMLR, 2023. 2

[14] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741, 2023. 2

[15] Devjeet Roy, Xuchao Zhang, Rashi Bhave, Chetan Bansal, Pedro Las-Casas, Rodrigo Fonseca, and Saravan Rajmohan. Exploring llm-based agents for root cause analysis. In *Companion proceedings of the 32nd ACM international conference on the foundations of software engineering*, pages 208–219, 2024. 8

[16] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[17] Chan Hee Song, Jiaman Wu, Clay Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models, 2022. 1, 2, 8

[18] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. Learning to summarize from human feedback. *arXiv preprint arXiv:2009.01325*, 2020. 2

[19] Ruoyu Wang, Zhipeng Yang, Zinan Zhao, Xinyan Tong, Zhi Hong, and Kun Qian. Llm-based robot task planning with exceptional handling for general purpose service robots. In *2024 43rd Chinese Control Conference (CCC)*, pages 4439–4444. IEEE, 2024. 2

[20] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. 2

[21] Yunhao Yang and Neel P. Bhatt et al. Fine-tuning language models using formal methods feedback: A use case in autonomous systems. In *Conference on Machine Learning and Systems*, CA, USA, 2024. mlsys.org. 3, 7, 8

[22] Yunhao Yang, Cyrus Neary, and Ufuk Topcu. Multimodal pretrained models for verifiable sequential decision-making: Planning, grounding, and perception. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 2011–2019, New Zealand, 2024. ACM. 3

[23] Yunhao Yang, Junyuan Hong, Gabriel Jacob Perin, Zhiwen Fan, Li Yin, Zhangyang Wang, and Ufuk Topcu. Lad-vf: Llm-automatic differentiation enables fine-tuning-free robot planning from formal methods feedback. *International Conference on Robotics and Automation*, 2026. 3

[24] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 2

[25] Li Yin and Zhangyang Wang. Llm-autodiff: Auto-differentiate any llm workflow. *arXiv preprint arXiv:2501.16673*, 2025. 2, 5, 7, 8

[26] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024. 2