PROVE: Predict-to-Specify and Observe-to-Verify for Human-Aligned Evaluation of Mobile Manipulation

Bowen Lyu Yunan Liu Liaoyuan Fan Qinchuan Cheng
Jiafei Duan Cheston Tan Shijie Li

Abstract

*Judging open-world robot manipulation is hard because layouts, articulated fixtures, and valid goal configurations vary widely. On RoboCasa365, simulator predicates agree with human judgments on only 69.09% of rollouts, and a frontier VLM judge on 86.12%. We present **PROVE** (Predict-to-Specify, Observe-to-Verify), which first specifies what success should look like and then checks the outcome. PROVE refines success conditions from the instruction, the initial scene, a target-object reference image, and a generated goal image. The goal image only sharpens the conditions and never serves as evidence. On 6,500 human-annotated rollouts over 65 atomic actions, PROVE reaches **96.83%** agreement. Applied to open-source policies, it reports far lower success than native checks and changes their ranking. To reduce evaluation overhead in policy learning, we distill PROVE into **PROVE-Lite**, a 4B evaluator that needs no reference images. In one forward pass, PROVE-Lite predicts the outcome and the terminal state of each task condition. It reaches **91.20%** agreement while running about 38× faster than PROVE. Mapped to a severity-aware reward, its diagnoses improve PPO post-training over both simulator and binary rewards.*

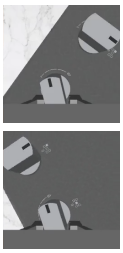
	(a) CoffeeSetupMug	(b) LowerHeat	(c) CloseBlenderLid
Instruction	Place the mug under the coffee dispenser.	Lower the heat of the kettle.	Close the blender lid securely.
Images			
RoboCasa	**SUCCESS** Mug near dispenser; gripper away.	**SUCCESS** Knob angle within the low-heat range.	**SUCCESS** Lid touches the blender; gripper away.
PROVE	**FAILURE** Mug lies on its side, opening sideways.	**FAILURE** Before vs. after: heat was raised.	**FAILURE** Lid does not fully cover the opening.
Human	**FAILURE** Mug lies on its side.	**FAILURE** Knob turned up, still in the low range.	**FAILURE** Lid does not cover the opening.

Figure 1. **Native checks accept failures that humans reject.** RoboCasa passes all three because each check tests only part of the task. In (b), only a before/after comparison shows that the knob was turned up.

1. Introduction

Mobile manipulation benchmarks decompose household tasks into atomic actions such as grasping, placing, closing doors, and turning knobs. Judging these actions reliably matters for benchmark credibility [30, 31, 52] and for policy improvement [48]. In open-world scenes, this is harder than it looks. Layouts and articulated fixtures vary across kitchens, and one instruction often admits several valid goal configurations. An evaluator should tell whether the robot succeeded and which physical conditions remain unmet. Ideally, this diagnosis should also help improve the policy.

Existing evaluators fall short. Simulation benchmarks check fixed predicates over privileged object states. These states are unavailable on physical robots, and the predicates cannot anticipate every valid outcome. On RoboCasa365 [31], they agree with human judgments on only **69.09%** of rollouts. They err in both directions, accepting failures and rejecting successes, and never explain why (Fig. 1). VLM judges [10, 43] avoid privileged state, but instruction-and-terminal-only judging omits the initial scene context. Prompted with the instruction alone, they turn plausible yet irrelevant expectations into requirements. For example, they may demand closing a cabinet door that was already open. Such a judge reaches **82.52%** agreement with our 8B backbone and **86.12%** with a frontier model, GPT-6.1 Sol [35].

Human judges do better because they separate two questions. What should success look like in this scene, and did it actually happen? They first set the standard from the task, the scene, and the expected outcome, and only then consult the evidence. VLMs understand instructions, objects, and

spatial relations [3]. Image-editing models can predict the intended post-action configuration [7, 8]. Generated images, however, contain artifacts and implausible contact. They tell us about the goal but are unreliable as a reference for matching. We therefore let predictions specify and observations verify. The goal image only sharpens the success conditions. The terminal observations alone decide whether those conditions hold, so the outcome never shapes its own standard.

Following this principle, we build **PROVE**, an evaluation harness that mirrors the human process (Fig. 2). One frozen MLLM refines success conditions in three stages. It starts from the instruction alone (**M1**) and adds the initial scene and target object (**M2**). The last stage adds a generated goal image (**M3**). On 6,500 human-annotated rollouts over 65 atomic actions, this progressive grounding raises agreement to **96.83%**. Ablations confirm that the goal image helps only as specification, and that the gain transfers across backbones. Applied to four open-source policies, PROVE reports far lower success than native checks and reorders them. When the two disagree, blind human review sides with PROVE in **95.42%** of cases.

PROVE suits offline evaluation, where each rollout is judged once. Policy training is different: thousands of rollouts must be judged on the fly. There, PROVE's multi-stage inference and reference assets become too costly. We therefore distill [14] it into **PROVE-Lite**, an evaluator that needs no reference images (Fig. 3). In a single forward pass, PROVE-Lite predicts the outcome and the terminal state of each task condition. Trained only on PROVE's pseudo-labels, it reaches **91.20%** agreement while running about 38× faster.

We validate PROVE-Lite where it is meant to be used: as the reward in policy post-training. Its diagnoses describe how far each condition is from its target. They can therefore grade failures instead of treating them alike. A narrow door gap, for instance, is penalized less than a wide-open door. We map each diagnosed state to a fixed severity. The resulting terminal reward gives failed rollouts bounded partial credit while keeping binary success dominant. In controlled PPO [42] post-training, it beats simulator rewards by **4.58** points and binary PROVE-Lite rewards by **2.00** points.

Our contributions are three-fold:

- **Predict-to-Specify, Observe-to-Verify evaluation.** PROVE grounds success conditions in the task, the scene, and a generated goal image. It verifies them against real observations only.
- **Reference-free evaluation for policy learning.** PROVE-Lite is distilled from PROVE without manual labels. Beyond the outcome, it diagnoses the terminal state of each task condition in one forward pass.
- **Diagnosis-guided terminal reward.** The reward grades failed rollouts by the physical severity of their unmet conditions. It outperforms simulator and binary rewards in PPO post-training.

2. Related Work

Evaluating Robot Manipulation. Simulation benchmarks including RoboCasa [30, 31], RLBench [18], and LIBERO [24] use simulator-state checks. BEHAVIOR [22, 45] formalizes initial and goal conditions with predicates. For physical robots, AutoEval [52] learns success classifiers. Following the broader model-as-judge paradigm [51], SuccessVQA [10] and RoboVQA [43] formulate judgments as visual questions. RoboJev [9] adapts the JEV-27B-VL judge [1] with LoRA and reads success from two calibrated tokens in one forward pass. Eval-Actions [25] pairs policy rollouts, failures included, with teleoperation data to grade execution quality. PROVE grounds evaluation criteria in scene-specific visual context and fixes them before inspecting terminal evidence.

Generated Goals in Robot Learning. SuSIE [7] and UniPi [11] generate goal images or video plans for control; Gen2Act [5] conditions robot policies on generated human videos. VoxPoser [17] grounds language-derived affordances and constraints in 3D value maps for planning. PROVE uses a generated goal to refine evaluation criteria instead. Since generated images can contain artifacts and cannot enumerate all valid outcomes, we retain textual conditions and withhold the generated image during verification.

Rewards from Foundation Models. VLM rewards use image–text similarity [4, 39], video–language matching [44], and preference feedback [46]; Eureka [28] instead uses an LLM to write reward code. LIV [27] jointly learns representations and value-based rewards. GVL [29], Robometer [23], and ProcVLM [13] estimate progress using in-context reasoning, trajectory comparisons, and procedural structure, respectively. Closest to our terminal grading, RoboReward [21] already scores partial and near-complete outcomes under an end-of-episode rubric. Our distinction is condition-level terminal diagnosis with a fixed class-to-severity mapping, learned from a teacher that separates criterion specification from verification.

Robot Failure Explanation. REFLECT [26] summarizes execution experience for failure explanation and correction. AHA [12] provides failure feedback for planning and reward refinement; Guardian [37] uses structured reasoning to detect manipulation failures. PROVE-Lite restricts diagnosis to terminal evidence: it predicts a closed set of per-condition physical states mapped directly to reward severity, without inferring an unobserved execution history.

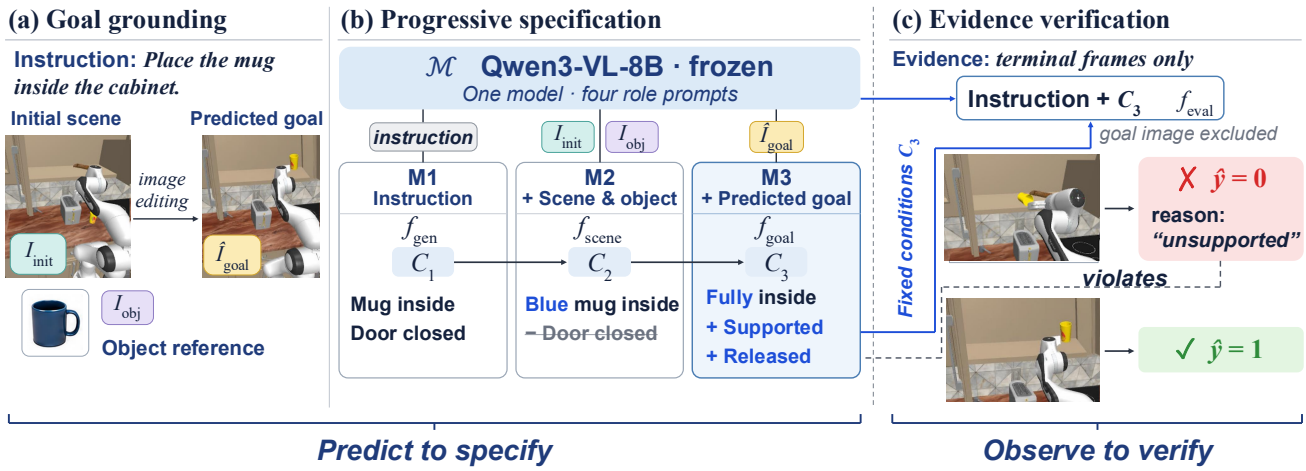


Figure 2. **Overview of PROVE.** (a) Inputs: the instruction, the initial scene I_{init}, an object reference I_{obj}, and a goal image $\hat{I}_{\text{goal}}$ predicted by image editing. (b) One frozen MLLM $\mathcal{M}$ refines the success conditions under three role prompts. Highlighted words are refined, "+" marks added conditions, and struck-out conditions are removed. (c) With f_{eval}, $\mathcal{M}$ checks the fixed conditions C_3 against terminal frames only. It outputs a label $\hat{y}$ and a reason; the goal image is never used here.

3. Evidence-Grounded Action Evaluation

3.1. Evaluation Formulation

Simulator predicates [30, 31] cannot capture the diverse layouts, articulated fixtures, and multiple valid goal configurations of open-world manipulation. Human judges instead combine task understanding, scene context, expected outcomes, and visual evidence. We abstract this process into three steps: (1) **task and scene grounding**, which establishes the intended physical outcomes in the current environment; (2) **goal anticipation**, which predicts the expected post-action configuration; and (3) **evidence-grounded verification and diagnosis**, which checks each condition against actual observations and identifies the primary deficiency when execution fails.

We instantiate this formulation on the 65 atomic mobile manipulation actions in RoboCasa365 [31], where each evaluation instance corresponds to one rollout specified by a natural-language instruction $\mathcal{I}$. Our MLLM-based evaluation framework, **PROVE**, strictly decouples goal specification from outcome verification. Before inspecting the terminal observation I_{final}, PROVE first states the required physical postconditions from $\mathcal{I}$ and progressively refines them using context that describes the goal rather than the execution outcome: the initial scene I_{init}, a target-object reference I_{obj}, and a predicted goal image $\hat{I}_{\text{goal}}$ (Section 3.2). Only after locking in these criteria does PROVE examine I_{final} to output a binary decision $\hat{y} \in \{0, 1\}$ alongside a primary failure reason ρ, ensuring that the actual outcome never shapes the standard by which it is judged (Section 3.3).

3.2. Progressive Condition Grounding

Across condition grounding and outcome verification, PROVE uses a single frozen MLLM $\mathcal{M}$ [3] instantiated with role-specific prompts: f_{gen} for initial condition generation, f_{scene} and f_{goal} for scene- and goal-guided refinement, and f_{eval} for terminal verification and failure diagnosis.

Task instructions specify the intended action but often leave scene-dependent completion criteria implicit. For example, an instruction to place an object inside a cabinet does not fully specify the target region, acceptable spatial tolerance, or required support condition. The expected physical outcomes must therefore be grounded in the actual scene before execution can be evaluated.

We progressively refine textual verification conditions through three stages (Fig. 2b). Instruction-derived semantic conditions establish the initial specification (**M1**). Scene context and object references resolve ambiguities in that specification (**M2**). Finally, a predicted goal image provides additional guidance about the intended post-action configuration (**M3**). The resulting conditions form the explicit criteria for subsequent verification.

Instruction-Derived Semantic Conditions (M1). The first stage establishes task-level expectations using semantic and physical priors. Given the task instruction $\mathcal{I}$, the MLLM $\mathcal{M}$ derives a set of concise physical postconditions:

$$C_1 = f_{\text{gen}}(\mathcal{I}). \tag{1}$$

Without scene context, M1 relies on generic language priors and may turn plausible but task-irrelevant expectations into required postconditions. For example, an instruction to place an object inside a cabinet may lead M1 to require closing a cabinet door that was already open, or moving the robot

arm out of the camera view. Neither requirement necessarily follows from the instruction.

Scene-Grounded Refinement (M2). The second stage grounds these expectations in the actual environment. It refines C_1 using the initial scene observation I_{init} and the target-object reference I_{obj}:

$$C_2 = f_{\text{scene}}(C_1, \mathcal{I}, I_{\text{init}}, I_{\text{obj}}). \tag{2}$$

The initial scene observation and object reference help identify the target object (e.g., distinguishing the target mug from visually similar containers) and establish which fixtures are already in the desired state. This prevents the evaluator from imposing unnecessary changes to the initial scene. However, because these inputs describe only the pre-action scene, they cannot fully specify the intended post-action geometry or spatial tolerances.

Goal-Guided Condition Refinement (M3). The third stage anticipates the expected post-action configuration using an image-editing generative model [7, 8]. Its candidate goal image $\hat{I}_{\text{goal}}$ provides additional spatial and physical guidance for refining the verification conditions:

$$C_3 = f_{\text{goal}}(C_2, \mathcal{I}, I_{\text{init}}, I_{\text{obj}}, \hat{I}_{\text{goal}}). \tag{3}$$

Unlike generic semantic priors or pre-action observations, $\hat{I}_{\text{goal}}$ provides a concrete visual hypothesis of the post-action arrangement, enabling $\mathcal{M}$ to clarify target spatial relationships, placement regions, and support configurations. Crucially, $\mathcal{M}$ distills only high-level geometric and relational cues from $\hat{I}_{\text{goal}}$ into the explicit textual criteria C_3, abstracting away low-level appearance variations. Because $\hat{I}_{\text{goal}}$ is used strictly during condition specification and withheld during outcome verification, it helps define the expected goal standard without ever serving as evidence of task completion.

3.3. Evidence-Grounded Outcome Verification

Once the conditions C_3 are fixed, $\mathcal{M}$ acts as an evidence verifier, checking each condition against the terminal observation I_{final} to output a binary outcome $\hat{y} \in \{0, 1\}$ along with a primary failure reason ρ:

$$(\hat{y}, \rho) = f_{\text{eval}}(\mathcal{I}, C_3, I_{\text{final}}). \tag{4}$$

Specifically, f_{eval} inspects all three camera views across six terminal frames (18 images in total). Every postcondition in C_3 must be positively confirmed by the terminal evidence; partial cues are not enough. Multiple views keep a nearby gripper that occludes a placed object from being misjudged as an incomplete release. Later frames capture transient events such as object settling or appliance activation. When an execution fails ($\hat{y} = 0$), ρ records the primary observable deficiency in the terminal state, without speculating about earlier events. The pair $(\hat{y}, \rho)$ later supervises PROVE-Lite (Section 4.1).

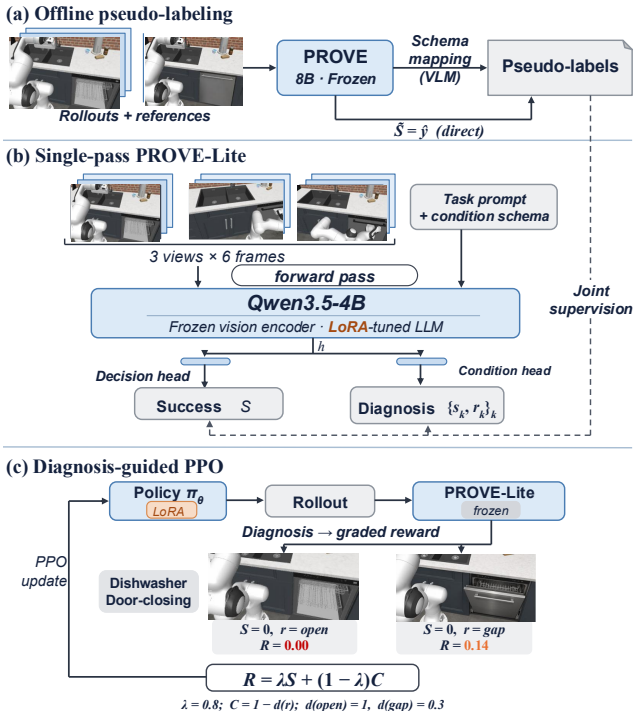


Figure 3. **Overview of PROVE-Lite.** (a) PROVE labels training rollouts offline; a VLM maps its conditions and verdict to the condition schema. (b) PROVE-Lite reads three views of six terminal frames in one forward pass. Two light heads share the hidden state $\mathbf{h}$. They predict success S and a per-condition diagnosis $\{s_k, r_k\}_k$. (c) The frozen PROVE-Lite turns the diagnosis into a graded reward $R = \lambda S + (1 - \lambda)C$ with $C = 1 - \sum_k w_k\, d(r_k)$, which drives PPO. Examples show a door-closing task.

4. Efficient Evaluation and Policy Learning

PROVE suits offline evaluation, where each rollout is judged once and accuracy outweighs cost. An evaluator is most valuable, however, as a reward signal: online post-training [48] must judge thousands of fresh rollouts inside the training loop, which PROVE's reference assets, goal-image synthesis, and multi-stage inference make too costly. We therefore distill [14] it into **PROVE-Lite**, a 4B evaluator that needs no reference images and reads out the binary outcome and the terminal state of each task condition from a single hidden state in one forward pass (Section 4.1), and convert this diagnosis into a terminal reward for PPO [42] (Section 4.2).

4.1. PROVE-Lite: Reference-Free Evaluator Distillation

Single-Pass Readout. We call **PROVE-Lite** reference-free because it needs no initial or generated reference images: no I_{init}, no I_{obj}, and no $\hat{I}_{\text{goal}}$. Its visual input is the terminal observation I_{final}: six frames from each of three camera views (18 images in total). The frames are sampled at $0, 0.2, 0.4, 0.6, 0.8$, and 1.0 s before termination. Instead of

generating text, PROVE-Lite uses the 4B VLM backbone [38] as a single-pass multimodal encoder. The prompt lists the task conditions $\mathcal{K}_\tau$ of the active action and all 44 terminal classes (Section 4.2), and ends with a task-completion query. This schema is fixed for each action, not generated per episode as in PROVE. In one forward pass with no generated tokens, PROVE-Lite takes the final-layer hidden state $\mathbf{h} \in \mathbb{R}^d$ at the query position and predicts

$$\big(S, \{(s_k, r_k)\}_{k\in\mathcal{K}_\tau}\big) = f_{\text{Lite}}(\mathcal{I}, I_{\text{final}}), \tag{5}$$

where $S \in \{0,1\}$ is the binary outcome, $s_k \in \{\texttt{SATISFIED}, \texttt{UNMET}\}$ is the binary status of condition k, and r_k is its terminal state and failure reason class.

Readout Heads. Two heads read out directly from $\mathbf{h}$:

- **DecisionHead ($\mathbf{h} \to S$).** Inspired by verbalizer-based classification [40], we initialize a linear head from the pretrained language model's `false` and `true` token embeddings to predict the binary outcome S.
- **ConditionHead ($\mathbf{h} \to \{s_k, r_k, \mathbf{e}_k\}$).** A shared two-layer MLP predicts, for each active condition k, its status s_k, its class r_k restricted to the classes valid for k (44 classes across all conditions), and an auxiliary evidence vector $\mathbf{e}_k \in \{0,1\}^{18}$ indicating which of the six frames and three views support the judgment.

The evidence targets ground each condition judgment in specific frames and views during training; the reward uses only S, s_k, and r_k.

PROVE-Guided Pseudo-Labeling. We run PROVE on training rollouts to obtain supervision without manual annotation. Related output-based distillation uses teacher-generated sequences [20] or rationales [15]; here, teacher outputs supervise structured classification heads instead of soft teacher logits. The binary target is PROVE's decision, $\tilde{S} = \hat{y}$. Condition targets need one more step. PROVE writes its conditions C_3 and its failure reason ρ in free-form, task-specific language, while the heads use the fixed schema of Section 4.2. A mapping VLM, `Qwen3-VL-8B-Instruct` [3], bridges the two. It receives the terminal frames I_{final} alongside C_3, $\hat{y}$, ρ, and the schema for action τ. For each condition $k \in \mathcal{K}_\tau$, it returns a binary status $\tilde{s}_k \in \{\texttt{SATISFIED}, \texttt{UNMET}\}$, a class $\tilde{r}_k$, and the supporting frames and views $\tilde{\mathbf{e}}_k$ directly within its structured output. Any condition not positively confirmed as satisfied is labeled `UNMET`, so silence or ambiguity from the teacher is never read as success.

Weighted Multi-Head Fine-Tuning. We jointly fine-tune LoRA adapters [16] on the language backbone and both heads, keeping the vision encoder frozen, under the objective

$$\mathcal{L} = w_s\, \mathcal{L}_{\text{CE}}(S, \tilde{S}) + \frac{w_r}{|\mathcal{K}_\tau|} \sum_{k\in\mathcal{K}_\tau} \mathcal{L}_k, \tag{6}$$

Severity tier	**Example classes (r)**	**#**	$d(r)$
Target achieved	`closed`, `inside`	12	0.0
Mild defect	`gap`, lid `offset`	2	0.3
Partial progress	`boundary`, `partial`	4	0.6
Unsecured object	`held`	2	0.8
Unachieved/unverified	`open`, `unknown`	24	1.0

Table 1. **Severity tiers over the 44 terminal classes.** Each class r is assigned a penalty $d(r)$ on an ordinal scale of distance from the target configuration; # gives the number of classes per tier. The full table of conditions, classes, and per-action weights is in the supplementary material.

$$\mathcal{L}_k = \mathcal{L}_{\text{CE}}(s_k, \tilde{s}_k) + \alpha_{\tilde{r}_k} \mathcal{L}_{\text{CE}}(r_k, \tilde{r}_k) + \beta\, \mathcal{L}_{\text{BCE}}(\mathbf{e}_k, \tilde{\mathbf{e}}_k), \tag{7}$$

where w_s and w_r balance the binary and diagnostic terms, $\alpha_{\tilde{r}_k}$ is a class-balancing weight that counteracts the dominance of frequent classes, and $\beta = 0.3$ down-weights the auxiliary evidence term.

4.2. Diagnosis-Guided Terminal Reward Design

Binary terminal rewards make no distinction between failed rollouts, regardless of their physical progress. We therefore convert **PROVE-Lite**'s terminal-state diagnosis into a graded reward that grants failed rollouts partial credit for the progress they visibly achieve.

Task Conditions. Each atomic action τ is assigned a small set of task conditions $\mathcal{K}_\tau$ with fixed weights w_k that sum to one, *e.g.*, `door_closed` for `CloseFridge`, or `object_inside` and `stable_placement` for `PickPlaceCounterToCabinet`. Each condition admits a closed set of terminal classes, *e.g.*, `closed`, `gap`, or `open` for `door_closed`, from which PROVE-Lite predicts r_k.

Diagnosis-to-Severity Mapping. We map each predicted class r_k to a severity penalty $d(r_k) \in [0,1]$ that increases with its physical distance from the target configuration (Table 1): a closed door incurs no penalty (0.0), a narrow gap a mild one (0.3), and an open door the maximum (1.0). The penalty depends only on the class and is shared by all actions that use the condition. Conditions whose target state cannot be visually confirmed are treated as unachieved (1.0). Conditions, weights, and penalties are fixed before policy training and are not tuned on evaluation outcomes.

Graded Terminal Reward. We aggregate the weighted penalties into a physical completion score

$$C = 1 - \sum_{k\in\mathcal{K}_\tau} w_k\, d(r_k) \in [0,1], \tag{8}$$

and combine it with PROVE-Lite's binary outcome $S \in \{0,1\}$:

$$R = \lambda S + (1-\lambda)\, C, \tag{9}$$

#	Evaluator	Specification		Verification	Wrong acc. ↓	Succ. rej. ↓	Agree. ↑
		$I_{\text{init}}, I_{\text{obj}}$	$\hat{I}_{\text{goal}}$	$\hat{I}_{\text{goal}}$			
1	Simulator predicate (privileged state)				22.32	37.16	69.09
2	GPT-6.1 Sol, direct judge				14.50	13.42	86.12
3	M1				17.02	17.81	82.52
4	M2	✓			18.01	7.81	87.89
5	M3 (PROVE)	✓	✓		<u>5.84</u>	<u>1.22</u>	<u>96.83</u>
6	M2 + goal at verification	✓		✓	13.95	18.77	83.26
7	M3 + goal at verification	✓	✓	✓	13.26	6.91	90.42
8	PROVE, GPT-6.1 Sol backbone	✓	✓		**5.81**	**1.17**	**96.88**
9	PROVE-Lite (Qwen3.5-4B, 1 pass)				6.21	10.69	91.20

Table 2. **Human alignment on the 6,500-episode benchmark (65 atomic actions).** Rates are in %. All evaluators judge the instruction and terminal frames; checkmarks mark extra inputs. M variants use Qwen3-VL-8B unless noted. Specification raises agreement from 82.52 to 96.83 (rows 3–5). The goal image helps for specification and hurts at verification (rows 6–7). The same framework lifts GPT-6.1 Sol from 86.12 to 96.88 (rows 2, 8). Bold: best; underline: second best.

where λ caps the partial credit a failure can earn at $1 - \lambda$ of a success. Any $\lambda > 0.5$ ranks every predicted success above every predicted failure; we use $\lambda = 0.8$, so binary success remains the dominant signal. For a predicted failure ($S = 0$), C grades physical progress and adds at most 0.2. For a predicted success ($S = 1$), $C < 1$ only when the two heads disagree, *i.e.*, the ConditionHead marks some condition as `UNMET`. Such a rollout then receives less than 1. The narrow-gap failure above ($S = 0$, $d = 0.3$), for instance, receives $R = 0.14$ rather than 0. Setting $\lambda = 1$ recovers the purely binary reward $R = S$. This task-specific terminal objective is not potential-based shaping [32] and carries no policy-invariance guarantee. The reward is issued only at episode termination, and GAE [41] propagates it across action chunks [50] during PPO [42] post-training.

5. Experiments

We first measure human alignment and ablate each specification stage (Sec. 5.2). We then re-evaluate open-source policies with PROVE (Sec. 5.3). Finally, we test whether PROVE-Lite keeps this alignment at low cost (Sec. 5.4) and whether its diagnosis improves post-training (Sec. 5.5).

5.1. Experimental Setup

Data. We use three sets of rollouts. The benchmark has 6,500 rollouts of the RoboCasa365 Xiaomi-Robotics-1 policy [47] over 65 atomic actions, with 10 scenes per action and 10 rollouts per scene. Humans label 3,762 successes and 2,738 failures from the instruction, the initial scene, and the terminal multi-view frames. A second annotator independently relabels 1,000 random rollouts and matches the primary labels on 95.4% of them. All alignment and diagnosis results use this set. PROVE-Lite trains on 32,500 rollouts covering all 65 actions, with 50 scenes per action and 10 rollouts per scene, using PROVE pseudo-labels only. None of these scenes overlap with the 6,500 benchmark rollouts. Post-training uses 15 actions, each with 80 training and 20 held-out test initializations shared by all groups. The supplement gives the completion rubric and split details.

Models. PROVE runs all stages on one frozen `Qwen3-VL-8B-Instruct` [3] in BF16. Goal images come from `gpt-image-2` [36]. PROVE-Lite uses `Qwen3.5-4B` [38] with LoRA [16] ($r = 16$) on the language model and thresholds success at 0.5.

Baselines. For human alignment, we compare with native predicates and with GPT-6.1 Sol [35] judging from the instruction and terminal frames only. As an ablation, we also give $\hat{I}_{\text{goal}}$ to the verifier of M2 and M3. For distillation, we LoRA-tune `Qwen3-VL-8B` and `Qwen3.5-4B` on the same pseudo-labels to generate the verdict as text.

Policy post-training. All groups start from `RLinf-pi0-robocasa-pretrain-human300` [6, 48]. The three trained groups differ only in the terminal reward. Each trains a rank-16 policy LoRA with five PPO iterations over eight parallel environments. PROVE-Lite stays frozen when it provides the reward. Other hyperparameters are in the supplementary material.

Metrics. Against human labels, we report agreement, wrong acceptance (failures judged successful), and success rejection (successes judged failed). Diagnosis is scored against PROVE-derived condition labels, with accuracy and macro-F1 for status and class and mean absolute error for C (Eq. (8)). Post-training reports PROVE-judged success as the mean and standard deviation over four seeds.

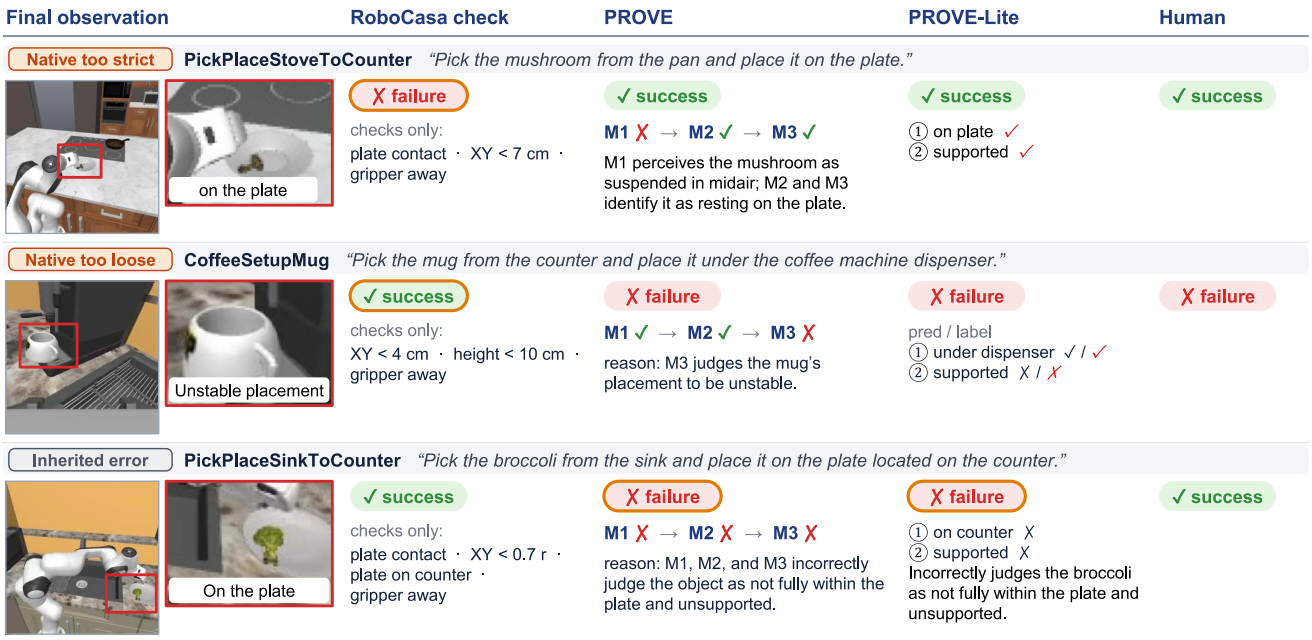


Figure 4. **Qualitative comparison.** Each row shows the final observation with a zoomed view of the evidence, followed by the verdicts of RoboCasa, PROVE, PROVE-Lite, and humans. Orange outlines mark verdicts that disagree with human review. "Supported" means stably placed on and supported by the target surface.

5.2. PROVE: Human Alignment and Ablations

Native predicates and direct judging fall short. Neither native predicates nor a frontier judge matches human outcomes (Tab. 2, rows 1–2). Native predicates agree on only 69.09% of rollouts. They accept 22.32% of failures and reject 37.16% of successes. Their checks cover only part of the task, so they err in both directions. A direct GPT-6.1 Sol judge does better at 86.12%, but still errs on about one rollout in seven.

Each specification stage helps. Agreement rises at every stage: 82.52% for M1, 87.89% for M2, and 96.83% for M3 (rows 3–5). The stages fix different errors. Instruction-only M1 rejects 17.81% of successes, mostly by imposing conditions the scene does not require. Scene grounding in M2 removes most of these, cutting success rejection to 7.81%. Its wrong acceptance stays high at 18.01%, because the initial scene cannot specify the final geometry. The goal image in M3 supplies this geometry, cutting wrong acceptance to 5.84% and success rejection to 1.22%. M3 is 27.74 points above native predicates and at the level of human inter-annotator agreement (95.4%).

Remaining errors. PROVE's remaining 206 errors concentrate on appliance activation, *e.g.*, starting a kettle or a microwave, and fine closure, *e.g.*, a refrigerator door. These cases involve transient states, occluded contact, or millimeter-scale gaps that terminal frames show poorly. Figure 4 shows typical cases. RoboCasa rejects a valid placement and accepts a tilted mug, while PROVE rejects a valid placement that PROVE-Lite inherits.

The goal image specifies; it should not verify. Using the goal image as evidence hurts both M2 and M3 (rows 6–7). Showing $\hat{I}_{\text{goal}}$ again at verification lowers M3 from 96.83% to 90.42%. Wrong acceptance rises to 13.26%, and success rejection to 6.91%. Adding the goal only at verification also lowers M2, from 87.89% to 83.26%. A generated goal shows one plausible outcome, not every valid one. Used as evidence, it likely penalizes valid alternatives and distracts from the terminal frames.

The gain comes from the framework. The improvement transfers across backbones (rows 2 and 8). With GPT-6.1 Sol as its backbone, PROVE reaches 96.88%, on par with the 8B backbone. The same model judging directly reaches only 86.12%. Progressive specification thus adds 10.76 points even to a frontier model.

5.3. Re-evaluating Open-Source Policies

Native success overstates progress and misorders policies (Tab. 3). We test three vision-language-action (VLA) policies, GR00T N1.5 [33, 34], RLDX-1 [19], and PRTS [49], and one world action model (WAM), Awomo-0.5I [2]. Native predicates report 12 to 40 points higher success than PROVE. To decide which judgment is right, annotators blind to both labels review all 8,677 disagreements. They side with PROVE in 95.42% of them. The ranking changes as well. RLDX-1 falls from second to last, and PRTS rises from third to second. Awomo-0.5I stays first, but its lead shrinks

	Success (%) / rank			Human support
Model	Native	PROVE	Disagree	N / P
GR00T N1.5	53.06 (4)	41.12 (3)	842	44 / 798
RLDX-1	69.98 (2)	40.11 (4)	2,242	117 / 2,125
PRTS	66.08 (3)	43.91 (2)	2,637	109 / 2,528
Awomo-0.5I	86.75 (1)	46.29 (1)	2,956	127 / 2,829
All	–	–	8,677	397 / 8,280

Table 3. **Policy re-evaluation.** Each policy runs on the 6,500 benchmark initializations (Sec. 5.1); ranks are in parentheses. Disagree counts rollouts where Native and PROVE differ. On these, annotators blind to both labels side with Native (N) or PROVE (P); 95.42% side with PROVE.

Evaluator	**Gen. tokens**	**Agree.**	**Lat. (s)**	**Mem. (GB)**
PROVE (M3)	386	96.83	10.21	16.95
Generative 8B	59.2	91.40	1.03	16.78
Generative 4B	33.8	89.20	0.84	9.04
PROVE-Lite, untrained	0	56.80	0.43	8.96
PROVE-Lite	0	91.20	**0.27**	8.97

Table 4. **Distillation and efficiency.** Agreement in %; latency per episode and peak memory on one NVIDIA RTX PRO 4500 GPU. Generative evaluators are LoRA-tuned on PROVE's pseudo-labels; the untrained PROVE-Lite uses the verbalizer-initialized head.

Prediction	**Space**	**Acc. (%)**	**Macro-F1**	**MAE**
Status s_k	2 per cond.	87.80	0.817	–
Class r_k	44 total	80.77	0.542	–
Completion C	$[0, 1]$	–	–	0.165

Table 5. **PROVE-Lite diagnosis against PROVE-derived condition labels**, over active conditions on the benchmark.

from 16.77 to 2.38 points.

5.4. PROVE-Lite: Distillation and Diagnosis

Distillation keeps most of the alignment. Without reference images, PROVE-Lite agrees with humans on 91.20% of rollouts across all 65 actions (Tab. 2, row 9). Its wrong acceptance is 6.21%, close to the 5.84% of PROVE, while its success rejection is higher at 10.69%. Since no benchmark scene appears in training, this measures generalization to new layouts and styles. Unlike PROVE, it never sees the initial scene. Actions judged against the initial state, such as lowering the heat in Fig. 1b, therefore exceed its input.

A single readout is much cheaper than generation. PROVE-Lite matches the generative 8B evaluator at about a quarter of its latency (Tab. 4). PROVE-Lite reaches 91.20%, against 91.40% for the generative 8B and 89.20% for the generative 4B. It needs 0.27 s per episode, against 1.03 s and 0.84 s. It is about 38× faster than PROVE and uses about half its memory. Without distillation, the same model reaches only 56.80%, so the alignment comes from training, not from the readout alone. Removing decoding makes PROVE-Lite cheap enough for the training loop.

Policy group	**Terminal reward**	**PROVE success (%)**
Base policy	–	50.17 ± 0.77
Simulator	simulator binary	52.00 ± 0.81
Binary	$R = S$	54.58 ± 0.88
Graded (ours)	$R = \lambda S + (1 - \lambda)C$	$\mathbf{56.58 \pm 0.91}$

Table 6. **PPO post-training on 15 tasks.** Mean ± s.d. over four seeds, with 20 test episodes per task and seed. All groups share the initialization and training budget; Binary and Graded use PROVE-Lite rewards with $\lambda = 0.8$ (Eq. (9)).

The diagnosis is accurate enough to grade failures. Against PROVE-derived condition labels (Tab. 5), status accuracy is 87.80% (0.817 macro-F1). Terminal-class accuracy is 80.77%, with 0.542 macro-F1 over 44 classes. The predicted completion score C has a mean absolute error of 0.165. With weight $1 - \lambda = 0.2$ in Eq. (9), this amounts to only about 0.033 in reward. For a predicted success, the condition term lowers R only when the two heads disagree. This happens for 10.8% of predicted successes, whose mean reward is then 0.877. A false binary acceptance still scores at least $\lambda = 0.8$, above any predicted failure.

5.5. Diagnosis-Guided Policy Post-Training

A more reliable reward helps, and grading failures by severity helps further (Tab. 6). All groups start from one checkpoint and share the same 1,200 held-out test episodes. The binary PROVE-Lite reward beats the simulator reward by 2.58 points. The graded reward adds 2.00 more, reaching 56.58%, 6.41 points above the base policy. Its gain over the binary reward holds across seeds (paired t-test, $p = 0.031$). To avoid scoring policies with their own reward model, PROVE judges the test rollouts. It differs from PROVE-Lite in inputs, model size, and access to reference assets.

6. Conclusion

We presented **PROVE**, an evaluator for open-world robot manipulation built on one principle. Predictions specify what success should look like, and only real observations verify whether it happened. Each grounding stage, from the task to the scene to a generated goal image, sharpens the success conditions. Across 65 RoboCasa365 actions, this raises human agreement from 82.52% to **96.83%**. Re-evaluated with PROVE, open-source policies score far below their native success, and their ranking changes. Distilled into **PROVE-Lite**, the same judgment becomes cheap enough for the training loop. PROVE-Lite keeps **91.20%** agreement in a single forward pass and diagnoses the terminal state of each condition. Mapped to fixed severities, these diagnoses yield a graded reward that beats simulator and binary rewards in PPO. This suggests that knowing *how* an execution failed,

not only *whether*, is a useful learning signal.

Limitations and future work. Our experiments use one simulator, RoboCasa365. PROVE needs no simulator-specific training, so applying it to other benchmarks and to physical robots is a natural next step. Task conditions and severities are defined per action; learning them from data would extend the reward to new tasks. Scaling diagnosis-guided post-training to more tasks and longer budgets is another promising direction.

References

[1] AutoTrust AI Lab. JEV-27B-VL. Hugging Face model card. https://huggingface.co/autotrust/JEV-27B-VL, 2026. Accessed: 2026-11-17.

[2] Awomo-WAM Team. Awomo-0.5 technical report. Technical report. https://github.com/Awomo-WestlakeDI/Awomo-0.5/blob/main/Awomo_0_5.pdf, 2026.

[3] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

[4] Kate Baumli, Satinder Baveja, Feryal Behbahani, Harris Chan, Gheorghe Comanici, Sebastian Flennerhag, Maxime Gazeau, Kristian Holsheimer, Dan Horgan, Michael Laskin, et al. Vision-language models as a source of rewards. *arXiv preprint arXiv:2312.09187*, 2023.

[5] Homanga Bharadhwaj, Debidatta Dwibedi, Abhinav Gupta, Shubham Tulsiani, Carl Doersch, Ted Xiao, Dhruv Shah, Fei Xia, Dorsa Sadigh, and Sean Kirmani. Gen2Act: Human video generation in novel scenarios enables generalizable robot manipulation. *arXiv preprint arXiv:2409.16283*, 2024.

[6] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[7] Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, Homer Walke, Chelsea Finn, Aviral Kumar, and Sergey Levine. Zero-shot robotic manipulation with pretrained image-editing diffusion models. In *International Conference on Learning Representations (ICLR)*, 2024.

[8] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. InstructPix2Pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[9] Josef Liyanjun Chen. RoboJev: An open vision-language judge for robot success detection. *arXiv preprint arXiv:2610.33718*, 2026.

[10] Yuqing Du, Ksenia Konyushkova, Misha Denil, Akhil Raju, Jessica Landon, Felix Hill, Nando de Freitas, and Serkan Cabi. Vision-language models as success detectors. In *Proceedings of The 2nd Conference on Lifelong Learning Agents (CoLLAs)*, pages 120–136, 2023.

[11] Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and Pieter Abbeel. Learning universal policies via text-guided video generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[12] Jiafei Duan, Wilbert Pumacay, Nishanth Kumar, Yi Ru Wang, Shulin Tian, Wentao Yuan, Ranjay Krishna, Dieter Fox, Ajay Mandlekar, and Yijie Guo. AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In *International Conference on Learning Representations (ICLR)*, 2025.

[13] Youhe Feng, Hansen Shi, Haoyang Li, Xinlei Guo, Yang Wang, Chengyang Zhang, Jinkai Zhang, Xiaohan Zhang, Jie Tang, and Jing Zhang. ProcVLM: Learning procedure-grounded progress rewards for robotic manipulation. *arXiv preprint arXiv:2605.08774*, 2026.

[14] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[15] Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, 2023.

[16] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[17] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. VoxPoser: Composable 3d value maps for robotic manipulation with language models. In *Proceedings of The 7th Conference on Robot Learning*, pages 540–562, 2023.

[18] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J. Davison. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

[19] Dongyoung Kim, Huiwon Jang, Myungkyu Koo, Suhyeok Jang, Taeyoung Kim, Beomjun Kim, Byungjun Yoon, Changsung Jang, Daewon Choi, Dongsu Han, et al. RLDX-1 technical report. *arXiv preprint arXiv:2605.03269*, 2026.

[20] Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, 2016.

[21] Tony Lee, Andrew Wagenmaker, Karl Pertsch, Percy Liang, Sergey Levine, and Chelsea Finn. RoboReward: General-purpose vision-language reward models for robotics. *arXiv preprint arXiv:2601.00675*, 2026.

[22] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, Mona Anvari, Minjune Hwang, Manasi Sharma, Arman Aydin, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Silvio Savarese, Hyowon Gweon, Karen Liu, Jiajun Wu, and Li Fei-Fei. BEHAVIOR-1K: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Proceedings of The 6th Conference on Robot Learning*, pages 80–93, 2023.

[23] Anthony Liang, Yigit Korkmaz, Jiahui Zhang, Minyoung Hwang, Abrar Anwar, Sidhant Kaushik, Aditya Shah, Alex S. Huang, Luke Zettlemoyer, Dieter Fox, Yu Xiang, Anqi Li, Andreea Bobu, Abhishek Gupta, Stephen Tu, Erdem Bıyık, and Jesse Zhang. Robometer: Scaling general-purpose robotic reward models via trajectory comparisons. In *Robotics: Science and Systems (RSS)*, 2026.

[24] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowl-

edge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

[25] Mengyuan Liu, Juyi Sheng, Peiming Li, Ziyi Wang, Tianming Xu, Tiantian Xu, and Hong Liu. Eval-Actions: Fine-grained execution quality evaluation for robotic manipulation. *arXiv preprint arXiv:2601.18723*, 2026.

[26] Zeyi Liu, Arpit Bahety, and Shuran Song. REFLECT: Summarizing robot experiences for failure explanation and correction. In *Conference on Robot Learning (CoRL)*, pages 3468–3484, 2023.

[27] Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. LIV: Language-image representations and rewards for robotic control. In *Proceedings of the 40th International Conference on Machine Learning*, pages 23301–23320, 2023.

[28] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

[29] Yecheng Jason Ma, Joey Hejna, Ayzaan Wahid, Chuyuan Fu, Dhruv Shah, Jacky Liang, Zhuo Xu, Sean Kirmani, Peng Xu, Danny Driess, et al. Vision language models are in-context value learners. In *International Conference on Learning Representations (ICLR)*, 2025.

[30] Soroush Nasiriany, Abhiram Maddukuri, Lance Zhang, Adeet Parikh, Aaron Lo, Abhishek Joshi, Ajay Mandlekar, and Yuke Zhu. RoboCasa: Large-scale simulation of household tasks for generalist robots. In *Robotics: Science and Systems (RSS)*, 2024.

[31] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. RoboCasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations (ICLR)*, 2026.

[32] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML)*, pages 278–287, 1999.

[33] NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, et al. GR00T N1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[34] NVIDIA GEAR Team. GR00T N1.5: An improved open foundation model for generalist humanoid robots. https://research.nvidia.com/labs/gear/gr00t-n1_5/, 2025.

[35] OpenAI. Introducing GPT-6.1 Sol. https://openai.com/index/introducing-gpt-6-1-sol/, 2026. Accessed: 2026-11-17.

[36] OpenAI. Introducing ChatGPT Images 2.0. https://openai.com/index/introducing-chatgpt-images-2-0/, 2026. Model: gpt-image-2. Accessed: 2026-11-17.

[37] Paul Pacaud, Ricardo Garcia, Shizhe Chen, and Cordelia Schmid. Guardian: Detecting robotic manipulation failures with vision-language models. *arXiv preprint arXiv:2512.01946*, 2026. Version 6.

[38] Qwen Team. Qwen3.5: Towards native multimodal agents. https://qwen.ai/blog?id=qwen3.5, 2026.

[39] Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024.

[40] Timo Schick and Hinrich Schütze. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 255–269, 2021.

[41] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[43] Pierre Sermanet, Tianli Ding, Jeffrey Zhao, Fei Xia, Debidatta Dwibedi, Keerthana Gopalakrishnan, Christine Chan, Gabriel Dulac-Arnold, Sharath Maddineni, Nikhil J. Joshi, et al. RoboVQA: Multimodal long-horizon reasoning for robotics. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[44] Sumedh A Sontakke, Jesse Zhang, Sébastien M. R. Arnold, Karl Pertsch, Erdem Bıyık, Dorsa Sadigh, Chelsea Finn, and Laurent Itti. RoboCLIP: One demonstration is enough to learn robot policies. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[45] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Elliott Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. BEHAVIOR: Benchmark for everyday household activities in virtual, interactive, and ecological environments. In *Proceedings of the 5th Conference on Robot Learning*, pages 477–490, 2022.

[46] Yufei Wang, Zhanyi Sun, Jesse Zhang, Zhou Xian, Erdem Bıyık, David Held, and Zackory Erickson. RL-VLM-F: Reinforcement learning from vision language foundation model feedback. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pages 51484–51501, 2024.

[47] Xiaomi Robotics Team, Jun Guo, Piaopiao Jin, Jason Li, Peiyan Li, Yingyan Li, Futeng Liu, Wanli Peng, Optimus Qin, Yifei Su, et al. Xiaomi-Robotics-1: Scaling vision-language-action models with over 100k hours of real-world trajectories. *arXiv preprint arXiv:2607.15330*, 2026.

[48] Hongzhi Zang, Mingjie Wei, Si Xu, Yongji Wu, Zhen Guo, Yuanqing Wang, Hao Lin, Peihong Wang, Liangzhi Shi, Yuqing Xie, et al. RLinf-VLA: A unified and efficient framework for reinforcement learning of vision-language-action models. In *Robotics: Science and Systems (RSS)*, 2026.

[49] Yang Zhang, Jiangyuan Zhao, Chenyou Fan, Fangzheng Yan, Tian Li, Haitong Tang, Sen Fu, Xuan'er Wu, Qizhen Weng, Weinan Zhang, Xiu Li, Chi Zhang, Chenjia Bai, and Xuelong Li. PRTS: A primitive reasoning and tasking system via contrastive representations. *arXiv preprint arXiv:2604.27472*, 2026.

[50] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Robotics: Science and Systems (RSS)*, 2023.

[51] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

[52] Zhiyuan Zhou, Pranav Atreya, You Liang Tan, Karl Pertsch, and Sergey Levine. AutoEval: Autonomous evaluation of generalist robot manipulation policies in the real world. In *Conference on Robot Learning (CoRL)*, pages 1997–2017, 2025.

PROVE: Predict-to-Specify and Observe-to-Verify for Human-Aligned Evaluation of Mobile Manipulation

Supplementary Material

A. Overview

This supplementary material gives the details that the main paper defers. Tables, figures, and equations numbered with an S prefix are in this document; those without it are in the main paper. Section B describes the camera views, the terminal window, and the object reference. Section C describes the models, the three rollout sets, and the human annotation. Section D lists the prompts. Section E gives the task conditions, the 44 terminal classes, and their severities. Section F details the PROVE-Lite architecture and training. Section G reports extended results. Section H describes the PPO protocol. Section I shows qualitative examples.

B. Observations and Object References

B.1. Camera Views

Each rollout is recorded by three synchronized cameras: left, right, and wrist (Tab. S1). The initial scene I_{init} contains the three views before execution. It helps identify the target object and the initial state of relevant fixtures.

View	Symbol	Role
Left camera	I_{left}	Scene and object context
Right camera	I_{right}	Scene and object context
Wrist camera	I_{wrist}	Close-range manipulation view

Table S1. Camera views used by PROVE and PROVE-Lite. Camera poses follow the RoboCasa365 rendering configuration.

Each camera provides RGB images at 256×256 resolution. The simulator camera identifiers are:

- Left: `robot0_agentview_left`
- Right: `robot0_agentview_right`
- Wrist: `robot0_eye_in_hand`

For the Xiaomi robot policy, each view undergoes a 95% center crop, yielding approximately 243×243 pixels, followed by bilinear resizing to 256×256. The three views are supplied separately to the policy.

For PROVE and PROVE-Lite evaluation, no additional manual cropping is applied. The views are concatenated horizontally in left–right–wrist order before model processing, as detailed in Sec. B.2.

B.2. Terminal Window

The terminal observation I_{final} covers the last second of the episode. It is sampled at six time steps (Tab. S2). At each time step, the synchronized left, right, and wrist views, each at 256×256 resolution, are concatenated horizontally in that order to form a single 768×256 RGB image. PROVE and PROVE-Lite both receive the resulting six composite images in chronological order. No additional manual cropping is applied. The images are then processed by the model processor configured with `min_pixels`$= 65{,}536$ and `max_pixels`$= 196{,}608$.

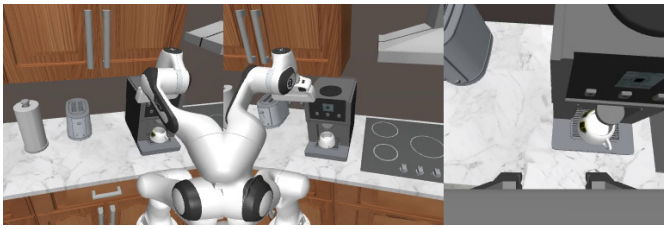


Figure S1. **Initial observation supplied to PROVE.** Left, right, and wrist views before execution.

Frame	Time before termination
F1	1.0 s
F2	0.8 s
F3	0.6 s
F4	0.4 s
F5	0.2 s
F6	0.0 s

Table S2. Temporal sampling of the terminal window. Each time step contains left, right, and wrist views.

B.3. Target-Object Reference

The task specification names the target object. Its simulator object ID identifies the target instance in the initial scene, avoiding confusion among objects of the same category. The reference I_{obj} is constructed from target-centered crops of the initial left, right, and wrist views. When the target is occluded, virtual-camera viewpoints are selected automatically as needed to obtain an unobstructed view of the same object. The reference is assembled as three horizontally arranged 384×384 panels, forming a 1152×384 RGB image. I_{obj} is used only by M2 and M3; PROVE-Lite does not receive it.

The target-appearance references are constructed from scene RGB renderings conditioned on scene metadata. Simulator segmentation information is used to identify target regions for cropping. The reference metadata records the selected view azimuths, target bounding boxes, and visible-pixel counts. These references do not reproduce the original

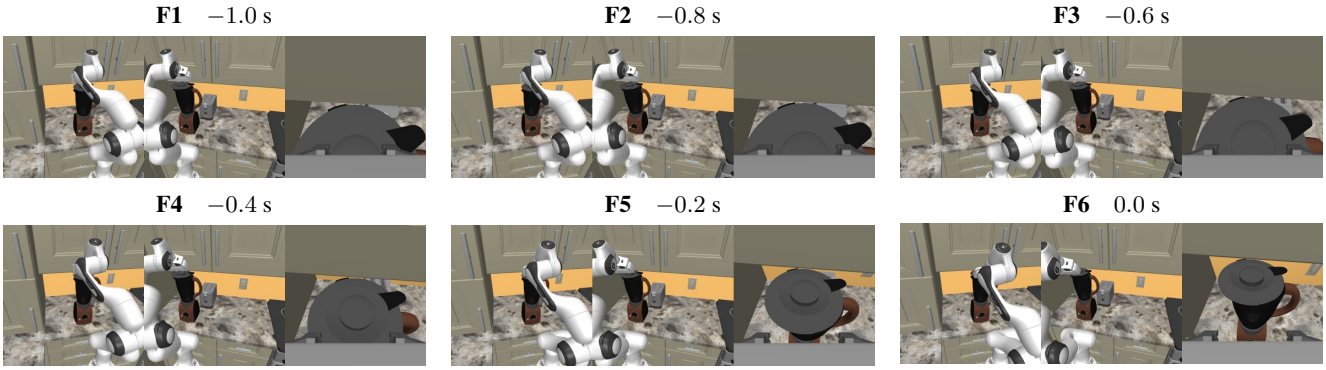


Figure S2. **Terminal observations on CloseBlenderLid.** Six composite images cover the final second of the episode, ordered chronologically from left to right and top to bottom. Each image contains synchronized left, right, and wrist views. Timestamps are relative to episode termination.

Property	Setting
Object identity	Task specification and simulator object ID
Reference time	Initial scene
Physical cameras	Left, right, wrist
Occlusion handling	Virtual-camera rendering of an unobstructed target view
Virtual-camera placement	Automatically selected based on target visibility
Reference resolution	1152×384 pixels; three horizontally arranged 384×384 panels
Number of virtual views	As needed to obtain an unobstructed target view

Table S3. Target-object reference settings.

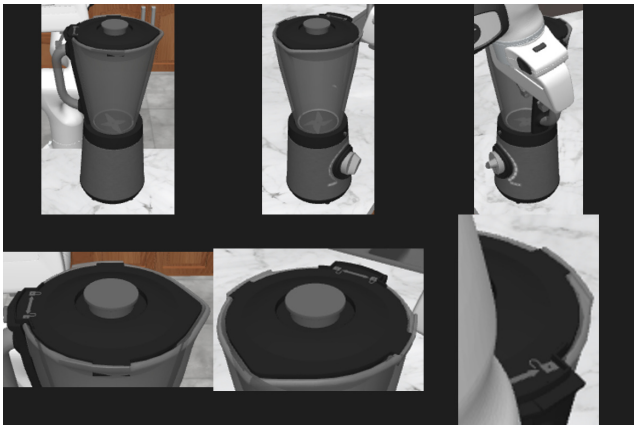


Figure S3. **Target-object reference.** An example reference image for OpenBlenderLid, showing the target blender and lid from multiple viewpoints. The reference is supplied to M2 and M3 for target identification and condition refinement.

scene poses and are used only for target appearance and identification, not as evidence of the original initial or terminal state.

Use of simulator information. Simulator metadata is used to locate and crop the target instance for I_{obj}. During condition verification, f_{eval} receives only $\mathcal{I}$, the textual conditions C_3, and the terminal RGB frames, without simulator-state inputs. PROVE-Lite and its reward computation likewise use no simulator-state inputs. On a physical robot, the same appearance reference could be obtained without privileged state, for example from an open-vocabulary detector or segmenter applied to the initial views, or from a photo of the target object. We have not evaluated this substitution, and we list it with real-robot validation as future work.

C. Models, Data, and Annotation

C.1. Models

Software environments. PROVE and PROVE-Lite use PyTorch 2.8.0 with CUDA 12.8, Torchvision 0.23.0, Transformers 5.17.0, and PEFT 0.21.0. PPO post-training runs in a separate RLinf environment with Transformers 4.53.2, PEFT 0.17.1, MuJoCo 3.3.1, and Ray 2.48.0. A separate portable evaluator manifest specifies Transformers 5.22.1 and PEFT 0.21.2.

Inference settings. The Qwen3-VL-8B evaluation runner uses BF16 and non-sampling decoding (`do_sample=False`), with a maximum of 650 new tokens per call. This limit applies to each model call; Table 4 in the main paper reports the mean number of tokens generated per rollout. The generative 4B baseline runner also uses non-sampling decoding, with a maximum of 256 new tokens. PROVE-Lite performs one forward pass without text generation and uses a success threshold of 0.5.

API usage dates. GPT-6.1 Sol was primarily accessed on November 15–16, 2026, and `gpt-image-2` on November 1–3, 2026.

Component	Model	Configuration	Role
PROVE MLLM $\mathcal{M}$	`Qwen3-VL-8B-Instruct`	Frozen, BF16	M1–M3 and verification
Goal-image generation	`gpt-image-2`	Default API settings	Predicted goal image $\hat{I}_{\text{goal}}$
Mapping VLM	`Qwen3-VL-8B-Instruct`	Frozen, BF16	Maps PROVE outputs to the condition schema
PROVE-Lite backbone	`Qwen3.5-4B`	$d = 2560$; LoRA $r = 16$, $\alpha = 32$	Single-pass evaluator
Generative baselines	`Qwen3-VL-8B`, `Qwen3.5-4B`	LoRA $r = 16$	Text-generating comparison
Direct judge	GPT-6.1 Sol	Reasoning effort: medium	Direct judge (Tab. 2, row 2); alternative PROVE backbone (row 8)
Policy	`RLinf-pi0-robocasa-pretrain-human300`	Policy LoRA $r = 16$	PPO initialization

Table S4. Models used in this work, grouped into PROVE and PROVE-Lite components, comparison methods (Tab. 2 and Tab. 4 of the main paper), and the PPO policy. The generative baselines are LoRA-tuned on the same PROVE pseudo-labels as PROVE-Lite and output the verdict as text.

C.2. Rollout Sets

We use three rollout sets (Tab. S5). In all sets, devices are selected randomly within the scene setup.

Benchmark. The benchmark has 6,500 rollouts of the RoboCasa365 Xiaomi-Robotics-1 policy: 65 actions, 10 scenes per action, and 10 rollouts per scene. Humans label 3,762 successes and 2,738 failures. The four policies re-evaluated in Sec. 5.3 run on the same 6,500 initializations.

PROVE-Lite training. PROVE-Lite trains on 32,500 rollouts: 65 actions, 50 scenes per action, and 10 rollouts per scene, with PROVE pseudo-labels only. These scenes use different layout–style combinations from the benchmark scenes, and the two sets share no scene.

PPO post-training. Post-training uses 15 actions, each with 80 training and 20 held-out test initializations. These initializations come from benchmark scenes; training and testing are separated at the initialization level rather than the scene level. None of these scenes is used to train PROVE-Lite.

C.3. Pseudo-Label Generation

PROVE labels each of the 32,500 training rollouts. The binary target is PROVE's decision, $\tilde{S} = \hat{y}$. The mapping VLM converts $(C_3, \hat{y}, \rho)$ and the terminal frames into per-condition targets $(\tilde{s}_k, \tilde{r}_k, \tilde{\mathbf{e}}_k)$ under the schema of Sec. E. Any condition not positively confirmed is labeled `UNMET`. Human labels are never used as training targets (Tab. S6).

C.4. Human Annotation

A primary annotator with relevant experience judged each benchmark rollout from the instruction, the initial scene, and the terminal multi-view frames. Full replay was available for ambiguous cases. The primary annotator followed a fixed physical-completion rubric and had no access to simulator predicates or model predictions.

Inter-annotator agreement. A second annotator independently relabeled 1,000 random benchmark rollouts and matched the primary labels on 95.4% of them (95% Wilson interval 93.9–96.5%). PROVE reaches 96.83% agreement with the benchmark reference labels, compared with 95.4% agreement between annotators on the relabeled subset.

Blind audit of policy re-evaluation. For the four policies in Sec. 5.3, annotators reviewed all 8,677 rollouts where native predicates and PROVE disagree. They saw neither label. They sided with PROVE in 8,280 cases (95.42%, 95% Wilson interval 94.96–95.84%) and with native predicates in 397. The rate is stable across policies, from 94.77% for GR00T N1.5 to 95.87% for PRTS (Tab. S16). For each disagreement, annotators watched the rollout video and recorded a binary success judgment.

C.5. Visual Quality Control

Before annotation and evaluation, an automated pass checks rendered frames for duplicated left and right buffers and for high-frequency shader noise. It flags identical pixels across views and abnormal Laplacian variance. Flagged episodes are re-rendered from saved MuJoCo states.

We manually inspected all rollout videos and target-object references for visual quality. Initial and terminal observations were extracted from the rollout videos.

D. Prompts

Table S7 summarizes the inputs and outputs of each prompt.

D.1. Shared Condition Rules

M1–M3 share the following condition-generation instruction. They ask for one to four concise, non-redundant, visually verifiable postconditions in JSON:

```
{"conditions":[{"requirement":"..."}]}
```

Set	Actions	Scenes/action	Rollouts/scene	Total	Labels
Benchmark	65	10	10	6,500	Human (3,762 S / 2,738 F)
PROVE-Lite training	65	50	10	32,500	PROVE pseudo-labels
PPO post-training	15	10	8 train + 2 test	1,200 train + 300 test inits.	Reward model / PROVE

Table S5. Rollout sets. Benchmark and PROVE-Lite training scenes are disjoint.

Target	Training	Evaluation
Outcome S	PROVE	Human labels
Status s_k	PROVE + mapping VLM	PROVE + mapping VLM
Class r_k	PROVE + mapping VLM	PROVE + mapping VLM
Evidence $\mathbf{e}_k$	Mapping VLM	Not evaluated

Table S6. Label provenance for PROVE-Lite. Condition-level evaluation labels come from the same PROVE and mapping-VLM pipeline applied to the benchmark rollouts, so they measure agreement with the teacher, not with humans.

The rules forbid constraints the task does not imply, such as visibility requirements, perfect centering, unobservable history, arbitrary arm retreat, and RGB thresholds. They also state that the generated goal image is fallible guidance, not proof of success.

> You assess physical robot-task completion. Conditions must describe only instruction-essential physical results, at most four concise nonredundant requirements. No visibility requirements, all-camera requirements, perfect centering, invisible history, arbitrary cleanup, robot retreat, numerical thresholds inferred from RGB, or future guarantees. Reference images are not achieved outcome evidence. A WM prediction is fallible goal guidance, never proof, mandatory appearance template or truth. Preserve valid earlier conditions; change only a specific instruction/grounding error. Never alter conditions to obtain a desired class balance or method ranking. Generate conditions, NOT an episode success judgement. Real final evaluation images and labels are withheld. Return JSON `{"conditions":[{"requirement":"..."}]}`. If refining, keep previous wording unchanged unless the additional allowed input identifies a concrete needed correction. Do not invent new requirements merely because more input is present.

D.2. M1: Instruction-Derived Conditions

M1 receives only the instruction $\mathcal{I}$. It lists the physical postconditions that must hold at the end of the episode for $\mathcal{I}$ to be satisfied. Without scene context, it may add plausible but unrequired conditions, such as closing a door that was already open. The M1 prompt is summarized below.

M1 sends the shared instruction and the applicable task rules in a single user message, followed by the task name and instruction. This stage has no separate system message.

```
[shared instruction]
[applicable task rules]
Task: [task name]
Instruction: [task instruction]
```

D.3. M2: Scene-Grounded Refinement

M2 receives $\mathcal{I}$, C_1, the initial views I_{init}, and the object reference I_{obj}. It identifies the target instance, removes conditions that the initial scene already satisfies or that the instruction does not require, and makes object references specific. It does not see any post-action image. The M2 prompt is summarized below.

M2 prepends the initial scene and target-object reference to the M1 message and appends C_1 as JSON. The image preamble is translated from Chinese.

> Initial scene and object reference, not evidence of completion:
> `[initial image] [object reference]`
> `[M1 message]`
> `previous_conditions:` [C_1 as JSON]

D.4. Goal-Image Generation

`gpt-image-2` edits the initial scene to depict one plausible post-action configuration for the task instruction $\mathcal{I}$. The generation request asks the model to preserve the target object's identity, camera viewpoints, panel order, and unrelated scene elements.

In the standard M1–M3 pipeline, M3 uses the generated goal image $\hat{I}_{\text{goal}}$ to refine the success conditions C_2. Terminal verification checks the resulting conditions against the recorded terminal observations. The ablations in Tab. 2 also supply the generated goal image during verification.

The generation request is summarized below.

> Edit the initial scene to show a plausible successful outcome for the task instruction. Apply the changes required by the instruction while preserving the target object's identity, camera viewpoints, panel order, and unrelated scene elements. Keep the resulting scene consistent across the three views.
>
> **Task instruction.** `[instruction]`

D.5. M3: Goal-Guided Refinement

M3 receives $\mathcal{I}$, C_2, I_{init}, I_{obj}, and $\hat{I}_{\text{goal}}$. It uses the goal image to sharpen spatial relations, placement regions, and support requirements, for example by adding that an object must be fully inside, supported, and released. The prompt states that $\hat{I}_{\text{goal}}$ shows one possible outcome, may contain artifacts, and must not be treated as evidence. Only geometric and relational cues enter the textual conditions C_3. The M3 prompt is summarized below.

M3 places the generated goal image before the initial scene and target-object reference, and supplies C_2 as the

Stage	Input	Output
M1 (f_{gen})	$\mathcal{I}$	C_1
M2 (f_{scene})	$\mathcal{I}, C_1, I_{\text{init}}, I_{\text{obj}}$	C_2
Goal generation	$\mathcal{I}, I_{\text{init}}, I_{\text{obj}}$	$\hat{I}_{\text{goal}}$
M3 (f_{goal})	$\mathcal{I}, C_2, I_{\text{init}}, I_{\text{obj}}, \hat{I}_{\text{goal}}$	C_3
Verification (f_{eval})	$\mathcal{I}, C_3, I_{\text{final}}$	$\hat{y}, \rho$
Mapping VLM	$C_3, \hat{y}, \rho, I_{\text{final}}$, schema of τ	$\tilde{s}_k, \tilde{r}_k, \tilde{\mathbf{e}}_k$
PROVE-Lite	$\mathcal{I}, I_{\text{final}}$, schema of τ	$S, \{s_k, r_k, \mathbf{e}_k\}$
Direct judge (GPT-6.1 Sol)	$\mathcal{I}, I_{\text{final}}$	Binary outcome

Table S7. Inputs and outputs of each prompted stage.

previous conditions. The image preambles are translated from Chinese.

> The WM prediction is a fallible goal reference: `[generated goal image]`
> Initial scene and object reference, not evidence of completion: `[initial image] [object reference]`
> `[M1 message]`
> `previous_conditions:` [C_2 as JSON]

D.6. Terminal Verification

f_{eval} receives $\mathcal{I}$, the fixed conditions C_3, and the terminal observation, but not $\hat{I}_{\text{goal}}$. The 18 view images (six timestamps $\times$ three camera views) are organized as six composite input images, each horizontally concatenating the synchronized left, right, and wrist views in that order. Every condition must be positively confirmed; partial cues are not enough. Multiple views help avoid mistaking occlusion by a nearby gripper for incomplete release. A temporal rule uses later frames for transient events such as settling or appliance activation. On failure, ρ names the primary observable deficiency without speculating about earlier events.

> Verify every fixed condition against real terminal evidence only. Unconfirmed conditions mean FAILURE. Describe the primary visible deficiency; do not invent execution history. Return JSON `{"label":"SUCCESS|FAILURE","reason":"..."}`.

The user message appends a JSON object with keys `instruction` and `fixed_conditions`, followed by the terminal views. Each view is preceded by the line `Observed F{i} {view} {relative_seconds:+.3f}s`. The response is a JSON object with `label` (`SUCCESS` or `FAILURE`) and `reason` (a string).

D.7. Mapping VLM

The mapping VLM receives C_3, $\hat{y}$, ρ, the terminal frames, and the condition schema of action τ. For each condition in $\mathcal{K}_\tau$, it returns a status, a class from the valid classes of that condition, and the supporting frames and views. A condition is `SATISFIED` only if the evidence positively confirms it; otherwise it is `UNMET`. The binary target stays PROVE's verdict; the mapping only derives the per-condition diagnosis. The output must report each active condition exactly once:

```
{"checks":[{"condition_code":...,
  "status":"SATISFIED"|"UNMET",
  "observation_code":...,
  "evidence":[{"frame":6,"view":"left"}],
  "visual_note":...}]}
```

An output is rejected if a condition is missing or repeated, a class is not valid for its condition, a frame or view is out of range, or a `SATISFIED` condition has no evidence. Failed mapping attempts are retried up to three times. All rollouts ultimately yield valid structured annotations, giving a final mapping failure rate of 0%.

D.8. PROVE-Lite Input Template

The user turn reads as follows; the chat template adds the role tokens.

> Evaluate the instruction using only real terminal observations. Inspect the task-specific condition schema. Determine overall success and the terminal state of each active condition. Do not infer unobserved history.
> `{"instruction": ..., "condition_schema": {"goal": ..., "conditions": {...}}}`
> Six composite images in chronological order, containing 18 view images in total; each composite arranges the left, right, and wrist views horizontally in that order.
> Is the task successfully completed? Inspect the active conditions.

The prompt lists the conditions of the active action and all 44 terminal classes. For each active condition, the schema specifies its rule, definition, and valid classes. The hidden state at the last input position feeds both heads.

D.9. Direct-Judge Baseline

GPT-6.1 Sol receives $\mathcal{I}$ and the terminal observation, comprising 18 view images from six timestamps and three camera views. The views at each timestamp are concatenated horizontally in left–right–wrist order, yielding six composite images supplied in chronological order. The model answers whether the task succeeded. It receives no initial scene, object reference, goal image, or condition list. The prompt is given below in English translation. Placeholders are filled with the task name, instruction, and terminal-image metadata for each rollout.

You are a visual evaluator of robot-task completion. Determine whether the current trajectory succeeds or fails, and explain the decision using concrete visual evidence. No labeled examples are provided; only the real terminal images constitute completion evidence.
Success must be supported by the required physical relation or state in the real images. Do not invent closure, correct placement, or button activation when the target is outside the field of view or fully occluded. Partial visibility may support reasonable inference across views; a complete object contour is not required. The absence of an initial image or the full action history is not itself evidence of failure.
For placement tasks, assess the task-specific location, orientation, and containment requirements. Failure due to continued grasping requires clear evidence that the closed gripper actually holds or supports the object. Proximity, contact, occlusion, or not observing release is insufficient. Do not require the arm to retreat.
F1–F6 are terminal observations; F1 is not the initial state. Judge only the available views and frames. Do not invent missing observations.
Return only the required JSON. Do not use tools, browse, read other files, or output a reasoning trace. If the evidence is insufficient to confirm success, return FAILURE and set `evidence_insufficient` to true. The label and explanation must express the same conclusion.
Task: `[task name]`.
Instruction: `[episode instruction]`.
Attachments: real F1–F6, each identified by its simulation step and time relative to termination.
Camera order: left, right, wrist.

The structured response contains `label`, `reason`, `target_visible`, `failure_basis`, `grasp_status`, `grasp_evidence`, and `evidence_insufficient`. The label is either `SUCCESS` or `FAILURE`; the explanation is returned in Chinese.

E. Task Conditions, Terminal Classes, and Severities

E.1. Conditions and the 44 Terminal Classes

The schema has 10 generic conditions (Tab. S8). Each condition keeps the identifier of the action family it was first written for, but its meaning is generic. These identifiers are used consistently in the schemas provided to the mapping VLM and PROVE-Lite. For example, `rack_target` covers any part, setting, or robot pose that must reach an instructed target, and `kettle_active` covers any device that must reach an instructed on, off, or running state.

Table S9 lists all 44 terminal classes. Each class belongs to one condition and has a fixed penalty $d(r)$. A class with $d(r) = 0$ has status `SATISFIED`; every other class, including `unknown`, has status `UNMET`. The tiers match Tab. 1: 12 classes at $d = 0$, 2 at 0.3, 4 at 0.6, 2 at 0.8, and 24 at 1.0. The same table supervises the ConditionHead and computes C for the reward.

E.2. Per-Action Conditions and Weights

For action τ with conditions $\mathcal{K}_\tau$, the completion score is $C = 1 - \sum_k w_k d(r_k)$ with $\sum_k w_k = 1$. Table S10 groups all 65 benchmark actions by their conditions and weights. Placement actions put 0.7 on reaching the target and 0.3 on stable release. All other actions except AdjustWaterTemperature and CoffeeSetupMug use a single condition. Table S11 lists the 15 PPO actions.

Condition	Meaning
`door_closed`	A door, drawer, or appliance head is fully closed
`lid_seated`	A lid is seated on its opening
`mug_aligned`	The mug is under the dispenser
`object_inside`	The object is fully inside the instructed container
`object_counter`	The object is on the instructed surface or region
`stable_placement`	The object is released and rests stably
`rack_target`	A part, setting, or robot pose reaches the instructed target
`kettle_active`	A device reaches the instructed on, off, or running state
`microwave_active`	An appliance reaches the instructed running state
`spout_target`	The faucet spout reaches the instructed direction

Table S8. The 10 generic task conditions.

E.3. Reward Walkthrough

For PickPlaceCounterToCabinet, $\mathcal{K}_\tau = \{\texttt{object_inside}, \texttt{stable_placement}\}$ with weights 0.70 and 0.30. Suppose the object rests stably on the shelf but straddles the cabinet rim. Then $r_1 = \texttt{boundary}$ with $d = 0.6$ and $r_2 = \texttt{supported}$ with $d = 0$, so

$$C = 1 - (0.70 \times 0.6 + 0.30 \times 0) = 0.58. \quad \text{(S1)}$$

With $S = 0$ and $\lambda = 0.8$, the reward is $R = 0.2 \times 0.58 = 0.116$. Table S12 gives the single-condition CloseFridge case.

F. PROVE-Lite Architecture and Training

F.1. Input

The input is one user turn. It states the task, then gives the instruction, the conditions of the active action, and all 44 terminal classes in JSON format. The schema specifies the action goal and, for each active condition, its rule, definition, and valid classes. The terminal observation contains 18 view images from six timestamps and three camera views. At each timestamp, the synchronized left, right, and wrist views are concatenated horizontally in that order, yielding six composite images supplied in chronological order. The turn ends with the query "Is the task successfully completed? Inspect the active conditions." The tokenized input contains exactly six composite images and at most 8,192 tokens. The auxiliary evidence vector retains 18 positions, one for each timestamp–view pair. Section D.8 gives the text.

#	Condition	Class r	Observable terminal state	Status	$d(r)$
1	door_closed	closed	Flush against the body	SAT.	0.0
2		gap	A visible gap remains	UNMET	0.3
3		open	Still clearly open	UNMET	1.0
4		unknown	Cannot be confirmed from the frames	UNMET	1.0
5	lid_seated	seated	Aligned with the opening and closed	SAT.	0.0
6		offset	Off the opening or tilted	UNMET	0.3
7		open	Opening still exposed	UNMET	1.0
8		unknown	Cannot be confirmed from the frames	UNMET	1.0
9	mug_aligned	aligned	Mug opening under the nozzle	SAT.	0.0
10		offset	Mug off the nozzle	UNMET	0.6
11		absent	No target mug under the nozzle	UNMET	1.0
12		unknown	Cannot be confirmed from the frames	UNMET	1.0
13	object_inside	inside	Fully inside the target region	SAT.	0.0
14		boundary	Straddles the boundary	UNMET	0.6
15		outside	Not in the target region	UNMET	1.0
16		unknown	Cannot be confirmed from the frames	UNMET	1.0
17	object_counter	supported	Supported at the target	SAT.	0.0
18		held	Still supported by the gripper	UNMET	0.8
19		off_counter	Away from the target	UNMET	1.0
20		unknown	Cannot be confirmed from the frames	UNMET	1.0
21	stable_placement	supported	Rests stably on the target surface	SAT.	0.0
22		held	Still grasped	UNMET	0.8
23		falling	Visibly tips or falls	UNMET	1.0
24		uncertain_support	Support occluded or ambiguous	UNMET	1.0
25		off_counter	Away from the required location	UNMET	1.0
26		unknown	Cannot be confirmed from the frames	UNMET	1.0
27	rack_target	reached	At the target	SAT.	0.0
28		partial	Progress, but short of the target	UNMET	0.6
29		wrong_direction	No correct progress, or moved the wrong way	UNMET	1.0
30		unknown	Cannot be confirmed from the frames	UNMET	1.0
31	kettle_active	indicator_on	Direct state evidence, e.g., indicator, flame, or water flow	SAT.	0.0
32		switch_active	Reliable evidence from the control, e.g., a switch or knob	SAT.	0.0
33		visible_off	Clearly in the opposite state	UNMET	1.0
34		no_visible_activation	No evidence of the required state	UNMET	1.0
35		unknown	Cannot be confirmed from the frames	UNMET	1.0
36	microwave_active	display_running	Display supports the target state	SAT.	0.0
37		interior_active	Interior or other direct cue supports it	SAT.	0.0
38		visible_off	Evidence of the opposite state	UNMET	1.0
39		no_visible_activation	No evidence of the required state	UNMET	1.0
40		unknown	Cannot be confirmed from the frames	UNMET	1.0
41	spout_target	reached	At the instructed direction	SAT.	0.0
42		partial	Same direction, short of the target	UNMET	0.6
43		wrong_direction	Wrong direction or no correct progress	UNMET	1.0
44		unknown	Cannot be confirmed from the frames	UNMET	1.0

Table S9. All 44 terminal classes with their status and penalty. SAT. denotes SATISFIED. Only the classes of an active condition are valid for that condition.

F.2. Heads

PROVE-Lite reads the final-layer hidden state $\mathbf{h} \in \mathbb{R}^{2560}$ at the last input position. Two heads read from $\mathbf{h}$ (Sec. 4.1).

- **DecisionHead** is a linear layer $\mathbb{R}^{2560} \to \mathbb{R}^2$. Its two rows are copied from the lm_head rows of the single tokens false and true. At initialization, it reproduces the base model's next-token scores restricted to these two tokens. The lm_head is then removed.
- **ConditionHead** is LayerNorm, Linear(2560, 512), GELU, Dropout(0.1), and Linear(512, 10×64). For each of the 10 conditions, the 64 outputs hold 2 status logits, 44 class logits, and 18 evidence logits.

Only the conditions of the active action are read and trained. Class logits outside the valid classes of a condition are masked before the softmax.

F.3. Training Configuration

All trainable parameters are optimized in one stage with Eqs. (6) and (7). $\mathcal{L}_k$ is averaged over the active conditions of each rollout. The class weight α_r is the square root of the inverse class frequency among the valid classes of a condition, capped at 3 and normalized to mean one. The evidence term uses a positive-class weight of 4. Table S13

Conditions and weights w_k	#	Actions
`rack_target` 1.0	18	AdjustToasterOvenTemperature, LowerHeat, NavigateKitchen, OpenBlenderLid, OpenCabinet, OpenDishwasher, OpenDrawer, OpenElectricKettleLid, OpenFridge, OpenFridgeDrawer, OpenMicrowave, OpenOven, OpenStandMixerHead, OpenToasterOvenDoor, PreheatOven, SlideDishwasherRack, SlideOvenRack, SlideToasterOvenRack
`object_counter` 0.7; `stable_placement` 0.3	12	CheesyBread, CoffeeServeMug, PickPlaceCabinetToCounter, PickPlaceCounterToSink, PickPlaceCounterToStove, PickPlaceDrawerToCounter, PickPlaceFridgeDrawerToShelf, PickPlaceMicrowaveToCounter, PickPlaceSinkToCounter, PickPlaceStoveToCounter, PickPlaceToasterOvenToCounter, PickPlaceToasterToCounter
`object_inside` 0.7; `stable_placement` 0.3	10	MakeIcedCoffee, PackDessert, PickPlaceCounterToBlender, PickPlaceCounterToCabinet, PickPlaceCounterToDrawer, PickPlaceCounterToMicrowave, PickPlaceCounterToOven, PickPlaceCounterToStandMixer, PickPlaceCounterToToasterOven, PickPlaceFridgeShelfToDrawer
`door_closed` 1.0	9	CloseCabinet, CloseDishwasher, CloseDrawer, CloseFridge, CloseFridgeDrawer, CloseMicrowave, CloseOven, CloseStandMixerHead, CloseToasterOvenDoor
`kettle_active` 1.0	9	StartCoffeeMachine, TurnOffSinkFaucet, TurnOffStove, TurnOnBlender, TurnOnElectricKettle, TurnOnSinkFaucet, TurnOnStove, TurnOnToaster, TurnOnToasterOven
`lid_seated` 1.0	2	CloseBlenderLid, CloseElectricKettleLid
`microwave_active` 1.0	2	TurnOffMicrowave, TurnOnMicrowave
`rack_target` 0.7; `kettle_active` 0.3	1	AdjustWaterTemperature
`mug_aligned` 0.7; `stable_placement` 0.3	1	CoffeeSetupMug
`spout_target` 1.0	1	TurnSinkSpout

Table S10. Conditions and weights of all 65 benchmark actions, grouped by condition set.



Figure S4. **Multi-condition reward example.** Earlier and final observations from the same `PickPlaceCounterToCabinet` rollout. In this manually inspected example, the object straddles the cabinet rim but is supported. The containment condition is unmet ($r_1 =$ `boundary`, $d_1 = 0.6$), while stable placement is satisfied ($r_2 =$ `supported`, $d_2 = 0$). With weights $w_1 = 0.7$ and $w_2 = 0.3$, $C = 0.7(1 - 0.6) + 0.3(1 - 0) = 0.58$. For $S = 0$ and $\lambda = 0.8$, the resulting reward is $R = 0.8S + 0.2C = 0.116$.

lists the configuration.

PROVE-Lite is trained in parallel on eight NVIDIA RTX 6000 Ada Generation GPUs.

G. Extended Results

G.1. Confusion Counts and Chance-Corrected Agreement

Table S14 expands Tab. 2 into counts on the 3,762 human successes and 2,738 human failures. It also reports balanced accuracy and Cohen's κ with the human labels, which correct for the 58:42 class ratio. The ranking of evaluators is the same under all three measures.

G.2. Remaining PROVE Errors

Of PROVE's 206 errors, 160 are wrong acceptances and 46 are success rejections (Tab. S14). PROVE therefore errs mainly by accepting failures, which matches the hard cases in the main paper: transient activation and millimeter-scale closure gaps that look complete in terminal frames. PROVE-Lite's 572 errors split the other way, with 170 wrong acceptances and 402 success rejections. Without the initial scene and the goal image, it rejects more valid outcomes. For a finer breakdown, we group ten actions into appliance activation (TurnOnElectricKettle and TurnOnMicrowave), fine closure (CloseFridge, CloseDishwasher, and CloseBlenderLid), placement (CoffeeSetupMug, PickPlaceCounterToCabinet, and PickPlaceSinkToCounter), and other actions (SlideDishwasherRack and TurnSinkSpout). Of PROVE's 206 errors over all 65 actions, 157 (76.21%) fall in the appliance-activation and fine-closure groups. Within these ten actions, PROVE-Lite makes 88 errors, of which 72 (81.82%) fall in the fine-closure and placement groups.

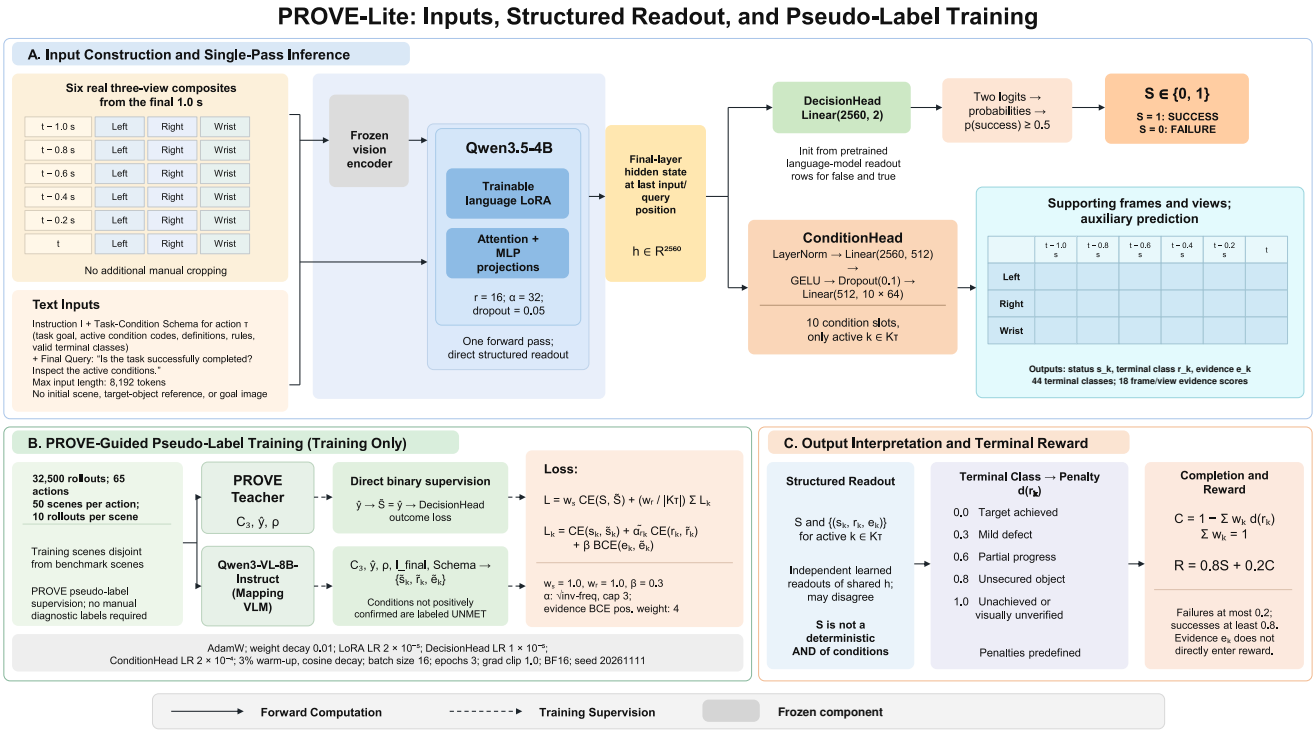


Figure S5. **PROVE-Lite architecture and training.** A shared query-position feature feeds the DecisionHead and ConditionHead for binary outcome prediction and structured condition diagnosis. PROVE-derived pseudo-labels supervise joint training, and predicted terminal classes determine the graded reward.

Action	Conditions and weights	Steps
CoffeeSetupMug	`mug_aligned` 0.7; `stable_placement` 0.3	600
TurnSinkSpout	`spout_target` 1.0	300
CloseDishwasher	`door_closed` 1.0	300
SlideDishwasherRack	`rack_target` 1.0	450
TurnOnMicrowave	`microwave_active` 1.0	450
TurnOnElectricKettle	`kettle_active` 1.0	450
PickPlaceCounterToCabinet	`object_inside` 0.7; `stable_placement` 0.3	750
CloseBlenderLid	`lid_seated` 1.0	900
CloseFridge	`door_closed` 1.0	900
PickPlaceSinkToCounter	`object_counter` 0.7; `stable_placement` 0.3	900
PickPlaceCabinetToCounter	`object_counter` 0.7; `stable_placement` 0.3	450
PickPlaceStoveToCounter	`object_counter` 0.7; `stable_placement` 0.3	450
PickPlaceToasterOvenToCounter	`object_counter` 0.7; `stable_placement` 0.3	450
PickPlaceToasterToCounter	`object_counter` 0.7; `stable_placement` 0.3	600
PickPlaceMicrowaveToCounter	`object_counter` 0.7; `stable_placement` 0.3	750

Table S11. Conditions, weights, and episode horizons of the 15 PPO actions.

Terminal state	d	S	C	R
Wide-open door	1.0	0	0.0	0.00
Narrow gap	0.3	0	0.7	0.14
Fully closed	0.0	1	1.0	1.00

Table S12. CloseFridge reward with $w = 1$ and $\lambda = 0.8$.

Parameter	Value
Backbone	`Qwen3.5-4B`, $d = 2560$, BF16
Trainable	Language LoRA and both heads
Frozen	Vision encoder
LoRA	$r = 16$, $\alpha = 32$, dropout 0.05
LoRA targets	Attention and MLP projections
Loss weights	$w_s = 1.0$, $w_r = 1.0$, $\beta = 0.3$
Class weight cap	3
Optimizer	AdamW, weight decay 0.01
Learning rate	LoRA 2×10^{-5}; DecisionHead 1×10^{-5}; ConditionHead 2×10^{-4}
Schedule	3% linear warm-up, cosine decay
Batch size, epochs	16, 3
Gradient clipping	1.0
Seed	20261111
Success threshold	0.5

Table S13. PROVE-Lite training configuration.

G.3. Diagnosis and Head Consistency

Against PROVE-derived labels, status accuracy is 87.80% (0.817 macro-F1), class accuracy is 80.77% (0.542 macro-

#	Evaluator	TP	TN	FP	FN	Agree. (95% CI)	Bal. acc.	κ
1	Simulator predicate	2,364	2,127	611	1,398	69.09 (67.96–70.20)	70.26	0.390
2	GPT-6.1 Sol, direct judge	3,257	2,341	397	505	86.12 (85.26–86.94)	86.04	0.717
3	M1	3,092	2,272	466	670	82.52 (81.58–83.43)	82.59	0.645
4	M2	3,468	2,245	493	294	87.89 (87.08–88.66)	87.09	0.749
5	M3 (PROVE)	3,716	2,578	160	46	96.83 (96.38–97.23)	96.47	0.935
6	M2 + goal at verification	3,056	2,356	382	706	83.26 (82.33–84.15)	83.64	0.662
7	M3 + goal at verification	3,502	2,375	363	260	90.42 (89.68–91.11)	89.92	0.802
8	PROVE, GPT-6.1 Sol backbone	3,718	2,579	159	44	96.88 (96.43–97.27)	96.51	0.936
9	PROVE-Lite	3,360	2,568	170	402	91.20 (90.49–91.86)	91.55	0.822

Table S14. Confusion counts against human labels on the 6,500 benchmark rollouts. Success is the positive class. The rates in Table 2 of the main paper are computed from these counts; intervals are 95% Wilson intervals. Row numbers follow Tab. 2.

F1), and the MAE of C is 0.165. Among rollouts predicted successful, 10.8% have $C < 1$; this subset has a mean reward of 0.877.

The condition-level metrics are evaluated against PROVE-derived labels. Their downstream utility is assessed by comparing binary and graded rewards from the same frozen evaluator. Graded achieves higher success under both PROVE and native-predicate evaluation (Sec. H.3).

Table S15 varies the success threshold of the Decision-Head. For thresholds from 0.3 to 0.6, agreement stays within 1.2 percentage points of the result at the default threshold of 0.5. Lower thresholds reduce success rejection while increasing wrong acceptance.

Threshold	Agree.	Wrong acc.	Succ. rej.
0.3	90.18	14.97	6.06
0.4	91.15	8.14	9.36
0.5	91.20	6.21	10.69
0.6	90.02	5.44	13.29
0.7	88.37	4.38	16.91

Table S15. PROVE-Lite on the benchmark under different success thresholds (%).

G.4. Timing Protocol

Latency and memory in Tab. 4 are measured on one NVIDIA RTX PRO 4500 GPU. We timed 200 rollouts per evaluator, processing one rollout at a time. Timing includes image loading and preprocessing. For PROVE, the timer covers condition generation and terminal verification using precomputed goal images. Goal-image generation is outside the timed interval. The PROVE timing script uses 10 separate warm-up rollouts.

G.5. Decomposing Policy Disagreements

Table S16 decomposes disagreements into native-only successes, where native predicates accept a rollout and PROVE rejects it, and PROVE-only successes, where the reverse holds. Native-only successes outnumber PROVE-only successes for every policy, explaining the higher success rates reported by native predicates.

The decomposition also clarifies the ranking reversal between PRTS and RLDX-1. Although native predicates accept more RLDX-1 rollouts, PROVE rejects a similar number of native-only successes for the two policies (2,039 for PRTS and 2,092 for RLDX-1) while accepting substantially more PROVE-only successes for PRTS (598 versus 150). Consequently, PROVE accepts 2,854 PRTS rollouts and 2,607 RLDX-1 rollouts. GR00T N1.5 and RLDX-1 differ by 66 accepted rollouts under PROVE, corresponding to 1.02 percentage points.

Across all 8,677 disagreements, blind human review supports PROVE in 8,280 cases (95.42%; 95% Wilson interval: 94.96–95.84%). The support rate ranges from 94.77% to 95.87% across the four policies.

H. PPO Protocol

H.1. Groups and Data

All groups start from `RLinf-pi0-robocasa-pretrain-human300` (Tab. S18). The PPO benchmark covers 15 actions and four seeds (42–45). For each action, rollouts 0–7 in 10 scenes form 80 training initializations, and rollouts 8–9 form 20 test initializations. These 20 test initializations are fixed by rollout index and shared across all groups and seeds. Each trained policy collects eight trajectories per update over five updates, so 40 training trajectories per action, group, and seed. Each group is tested on $15 \times 4 \times 20 = 1{,}200$ episodes. The reward is given only at termination, and GAE propagates it across action chunks. The Binary group uses $R = S$, i.e., $\lambda = 1$; the Graded group uses $R = 0.8S + 0.2C$, i.e., $\lambda = 0.8$. Both groups use predictions from the same frozen PROVE-Lite: Binary uses the success prediction alone, whereas Graded combines it with the condition-based completion score.

H.2. Hyperparameters and Hardware

The three PPO training groups run concurrently on three separate NVIDIA H100 GPUs, one per reward group. A frozen

Policy	Native S	PROVE S	Native-only S	PROVE-only S	Disagree	N / P	Side with PROVE (95% CI)
GR00T N1.5	3,449	2,673	809	33	842	44 / 798	94.77 (93.06–96.08)
RLDX-1	4,549	2,607	2,092	150	2,242	117 / 2,125	94.78 (93.78–95.63)
PRTS	4,295	2,854	2,039	598	2,637	109 / 2,528	95.87 (95.04–96.56)
Awomo-0.5I	5,639	3,009	2,793	163	2,956	127 / 2,829	95.70 (94.91–96.38)
All	17,932	11,143	7,733	944	8,677	397 / 8,280	95.42 (94.96–95.84)

Table S16. Policy disagreements decomposed. Success counts are out of 6,500 rollouts per policy and follow from Tab. 3. Native-only S: native accepts, PROVE rejects; PROVE-only S: the reverse.

	PROVE-judged		Native-predicate	
Group	Succ. (%)	Δ	Succ. (%)	Δ
Base policy	50.17 ± 0.77	–	50.58 ± 0.88	–
Simulator	52.00 ± 0.81	$+1.83$	52.67 ± 0.90	$+2.09$
Binary	54.58 ± 0.88	$+4.41$	54.08 ± 1.03	$+3.50$
Graded	$\mathbf{56.58 \pm 0.91}$	$\mathbf{+6.41}$	$\mathbf{56.83 \pm 1.45}$	$\mathbf{+6.25}$

Table S17. PPO results on 1,200 held-out episodes per group (mean ± s.d. across four seeds). Δ: gain over the base policy in percentage points.

Group	Update	Terminal reward
Base policy	None	–
Simulator	PPO, 5 iterations	Native predicate, $\{0, 1\}$
Binary	PPO, 5 iterations	$R = S$
Graded	PPO, 5 iterations	$R = \lambda S + (1 - \lambda)C$

Table S18. PPO groups. Binary uses PROVE-Lite's outcome S. Graded uses S and completion score C, with $\lambda = 0.8$.

Parameter	Value
Optimizer	AdamW
Learning rate	Policy 5×10^{-6}; value 1×10^{-4}
Discount γ	0.99
GAE λ	0.95
Clip range	0.2
Value clip	0.2
Entropy coefficient	0
Global batch size	8
PPO epochs per update	1
Gradient-norm clipping	1.0
Policy LoRA	$r = 16$, all linear layers
Parallel environments	8
Action chunks per step	5
Adam (β_1, β_2)	(0.9, 0.95)
Weight decay	0.01
Episode horizon	300–900 steps per action

Table S19. PPO hyperparameters.

PROVE-Lite model is deployed on a fourth NVIDIA H100 GPU and serves reward requests asynchronously. Failed reward requests or invalid reward outputs trigger retries.

H.3. Aggregate Results Across Seeds

Table S17 expands Table 6 of the main paper. The graded reward also ranks first under native predicates. Under native-predicate evaluation, Graded exceeds Binary by 2.75 percentage points and Simulator by 4.17 percentage points.

Evaluation coverage. We evaluate each group on 15 actions with 20 episodes per action under each of four random seeds (42–45), giving 1,200 episodes per group.

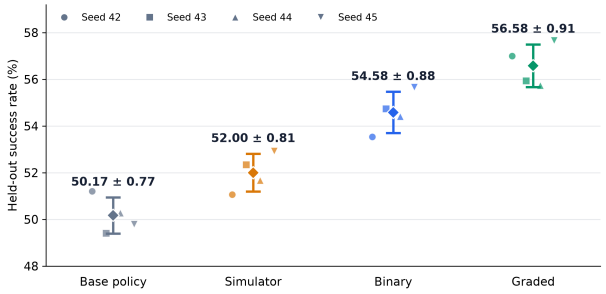


Figure S6. **PPO results.** Final held-out success rates across four seeds (42–45). Small markers show individual seeds; diamonds and error bars indicate the mean and sample standard deviation.

I. Qualitative Examples

Instruction. Turn on the toaster oven by setting the timer.



Initial scene | Target-object reference | Generated goal image

M1 (C_1) — Instruction only

Toaster oven timer is set to a non-zero value
Toaster oven power indicator light is illuminated
Toaster oven control knob or button is in the 'on' position
Toaster oven door is closed *(removed in M2)*

M2 (C_2) — Scene and object

Toaster oven timer is set to a non-zero value
Toaster oven power indicator light is illuminated *(removed in M3)*
Toaster oven control knob or button is in the 'on' position

M3 (C_3) — Goal-guided refinement

Toaster oven timer is set to a non-zero value
Toaster oven control knob or button is in the 'on' position

Figure S7. **Progressive specification for turning on a toaster oven.** M2 drops the closed-door requirement after incorporating the initial scene and target-object reference. M3 drops the power-indicator requirement after incorporating the generated goal image, retaining the timer and control-position conditions. Bold text marks the conditions removed at the next stage.



(a) Open the fridge doors. (b) Fully open the fridge drawer.

Figure S8. **Artifacts in generated goal images.** The generation review flagged (a) a duplicate handle at the door edge and (b) a lower food item replaced by a duplicate of the elongated item above. PROVE uses generated goal images to refine success conditions and verifies those conditions using recorded terminal observations.

(a) CoffeeSetupMug `t00-s01-r04`

Human: **FAILURE** PROVE: **FAILURE** PROVE-Lite: **SUCCESS**



PROVE. The mug is not resting on the coffee-machine tray.

PROVE-Lite. `mug_aligned`: SATISFIED / `aligned`; `stable_placement`: SATISFIED / `supported`.

Evaluation error. PROVE-Lite falsely accepts the rollout.

(b) PickPlaceSinkToCounter `t05-s05-r07`

Human: **SUCCESS** PROVE: **FAILURE** PROVE-Lite: **SUCCESS**



PROVE. The cheese extends beyond the plate; its explanation also questions the plate location.

PROVE-Lite. `object_counter`: SATISFIED / `supported`; `stable_placement`: SATISFIED / `supported`.

Evaluation error. PROVE falsely rejects a human-labeled success; PROVE-Lite agrees with the human.

Figure S9. **Failure cases of PROVE and PROVE-Lite.** Each case shows the final left, right, and wrist views, the human verdict, and the evaluator predictions. (a) PROVE-Lite incorrectly accepts a failed mug-placement rollout. (b) PROVE incorrectly rejects a successful placement rollout. In the placement case in Figure 4 of the main paper, both PROVE and PROVE-Lite incorrectly reject a valid outcome. Here, PROVE-Lite correctly accepts the rollout despite PROVE's false rejection. The explanations and condition outputs are model predictions.