Learning from Trials and Errors: Reflective Test-Time Planning for Embodied LLMs

Yining Hong[1], **Huang Huang**[1], **Manling Li**[2], **Li Fei-Fei**[1], **Leonidas Guibas**[1], **Jiajun Wu**[1], **Yejin Choi**[1]
[1]Stanford University [2]Northwestern University

Website: https://reflective-test-time-planning.github.io
Code: https://github.com/Reflective-Test-Time-Planning/Reflective-Test-Time-Planning

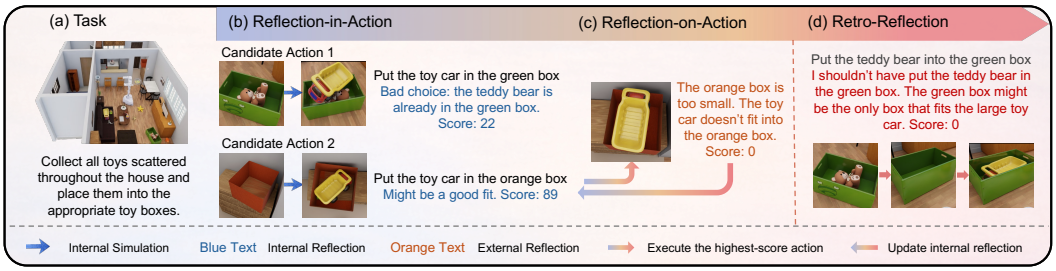

Figure 1: Conceptual overview of Reflective Test-Time Planning. The agent (a) receives a long-horizon task, (b) performs *reflection-in-action* by internally simulating and scoring candidate actions, (c) performs *reflection-on-action* by updating its beliefs and decision systems based on execution outcomes, and (d) conducts *retrospective reflection* to revise earlier decisions with hindsight.

Abstract

Embodied LLMs endow robots with high-level task reasoning, but they cannot reflect on what went wrong or why, turning deployment into a sequence of independent trials where mistakes repeat rather than accumulate into experience. Drawing upon human reflective practitioners, we introduce Reflective Test-Time Planning, which integrates two modes of reflection: *reflection-in-action*, where the agent uses test-time scaling to generate and score multiple candidate actions using internal reflections before execution; and *reflection-on-action*, which uses test-time training to update both its internal reflection model and its action policy based on external reflections after execution. We also include retrospective reflection, allowing the agent to re-evaluate earlier decisions and perform model updates with hindsight for proper long-horizon credit assignment. Experiments on our newly-designed Long-Horizon Household benchmark and MuJoCo Cupboard Fitting benchmark show significant gains over baseline models, with zero-shot generalization to photorealistic HM3D environments and real-robot experiments on a Franka Panda arm. Ablations confirm that reflection-in-action and reflection-on-action are mutually dependent, and that retrospective reflection achieves better credit assignment than step-wise external feedback at lower computational overhead. Qualitative analyses further highlight behavioral correction through reflection.

"Error isn't simple darkness, it sheds a light of its own."

— Kathryn Schulz, *Being Wrong*

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

1 Introduction

Embodied LLMs [Zitkovich et al., 2023, Kim et al., 2024, Hong et al., 2024] equip agents with task planning abilities, but they remain brittle static oracles that cannot learn from failures, turning deployment into independent trials of repeated mistakes rather than accumulated experience. Humans, in contrast, are natural reflective practitioners. Drawing on Schön's framework about reflective planning [Schön, 1992], humans fluidly alternate between two modes of reflection: through *reflection-in-action*, we engage in internal simulation, questioning whether our planned approach will actually work given what we currently understand; through *reflection-on-action*, we use the actual outcomes to reshape both our beliefs about the environment and our strategies for acting within it. An illustrative example of these reflection modes is shown in Figure 1. This bidirectional flow allows us to learn not only from outcomes, but also from the very process of engaging with an uncertain world.

Current approaches have pushed forward individual reflection mechanisms, but largely in isolation. One line of work [Shinn et al., 2023, Madaan et al., 2023] uses LLM-based verbal reflection, generating natural-language critiques of past behavior to condition future actions. While this enables reflection-on-action at the level of reasoning traces, it remains underexplored how such reflections can move beyond contextual text to become persistent learning signals. A second line of work [Zhen et al., 2024, Feng et al., 2025] supports reflection-in-action by guiding action selection through internal world models, yet raising another challenge: can such internal beliefs be updated when execution reveals mismatches between expected and actual outcomes? Classical planning and RL methods [Rubinstein, 1999, Williams et al., 2017, Schulman et al., 2017a] approach the problem from a different angle, revising behavior through scalar reward signals aggregated over many rollouts, episodes, or resets. Yet such scalar feedback is non-linguistic, noisy, and amortized across episodes, making it hard to diagnose repeated failures within just one single unfolding episode of interaction.

To address these challenges and operationalize both reflection modes in embodied settings, we introduce Reflective Test-Time Planning, a systematic framework that unifies reflection-in-action and reflection-on-action for embodied agents during test-time deployment. Concretely, the framework employs three embodied LLMs during deployment: an action generation model π_θ, an internal evaluator V_{ϕ_i}, and an external evaluator V_{ϕ_e}. During reflection-in-action, the agent samples N candidate actions via high-temperature sampling, uses V_{ϕ_i} to generate internal reflections scoring each candidate, then executes the highest-scoring action. After execution, V_{ϕ_e} generates an external reflection, providing an immediate, language-based evaluation of what happened and why.

This immediate external reflection grounds beliefs in reality, but remains inherently local—it only evaluates consequences visible at the next timestep. Many embodied failures are non-local: an action that appears successful may later block progress, and a seemingly suboptimal action may enable future success. To address this temporal credit assignment problem, we introduce retrospective reflection, where V_{ϕ_e} periodically re-evaluates earlier decisions with hindsight (e.g., at room transitions or after repeated failures). These hindsight assessments provide self-supervised signals at deployment time, enabling two forms of test-time training: (1) policy gradient for π_θ to favor actions that score well under hindsight, and (2) supervised learning for V_{ϕ_i} to anticipate what hindsight will reveal. Because these updates revise not only the action policy but also the predictive assumptions behind it, the process constitutes a form of double-loop learning [Argyris and Schön, 1977], in which agents learn not merely from outcomes but from diagnosing and correcting the underlying causes of their errors.

We evaluate our approach on two embodied benchmarks that we design to stress error-driven adaptation: (1) a Long-Horizon Household benchmark that requires failure recovery during multi-step planning across rooms, and (2) a controlled MuJoCo Cupboard Fitting benchmark that isolates geometric placement failures. Our framework achieves large gains over reflective language, RL and world-model baselines, with zero-shot generalization to photorealistic HM3D environments and real-robot experiments on a Franka Panda arm. Ablations indicate that improvement emerges only when both reflection-in-action and reflection-on-action take place, and when both action policy and internal reflection model are updated during deployment; and that retrospective reflection achieves better credit assignment than step-wise external feedback at lower computational overhead. Qualitative analyses further show that reflection reduces repetitive failure modes in practice.

2 Related Works

Test-Time Adaptation (TTA). TTA adapts models to distribution shifts during inference without source data [Sun et al., 2020, Wang et al., 2021, Liang et al., 2025]. Early methods minimize entropy, with Tent [Wang et al., 2021] updating batch-norm parameters online and later work adding calibrated objectives [Niu et al., 2022, Yang et al., 2024]. Parameter-efficient TTA uses LoRA [Hu et al., 2021, Kojima et al., 2025], bias-only tuning [Dumpala et al., 2023], or hidden-state adaptation for long-context memory [Sun et al., 2024]. In embodied settings, continual learning supports manipulation and navigation [Lesort et al., 2020, Meng et al., 2025, Hajizada et al., 2024, Mendonca et al., 2024]. We adapt models at test time via self-supervised signals from the agent's own verbal assessments.

Multimodal Embodied Large Language Models. Recent foundation models leverage large-scale robotic data for zero-shot generalization [Zitkovich et al., 2023, Kim et al., 2024, Driess et al., 2023], including RT-2's web-knowledge transfer and OpenVLA's heterogeneous embodiment support. 3D spatial grounding spans point clouds [Hong et al., 2023], 3D patches [Zhu et al., 2025], and lightweight point-cloud injection [Li et al., 2025]. Extensions incorporate multisensory interaction [Hong et al., 2024], generative world models [Zhen et al., 2024], long-term spatial–temporal memory [Hu et al., 2025], interleaved multimodal instructions [Fan et al., 2025], and chain-of-thought reasoning [Zhao et al., 2025, Mu et al., 2023]. In contrast, we treat deployment as a learning phase in which the embodied multimodal LLM reflects on actions and updates itself via test-time training.

Reflection and Self-Improvement in AI Agents. Verbal self-reflection methods such as Reflexion [Shinn et al., 2023] store natural-language critiques to guide future actions, with extensions to self-refinement [Madaan et al., 2023, Zhang et al., 2024], tool-assisted verification [Gou et al., 2024], curiosity-driven reflection [Kauvar et al., 2024], multi-agent systems [Ng, 2024], and robotics [Huang et al., 2022]. These methods support reflection-on-action but use reflections as text rather than deployment-time parameter updates, while complementary reflection-in-action methods use internal predictive models to anticipate outcomes [Zhen et al., 2024, Feng et al., 2025, Zhen et al., 2025, Hafner et al., 2024]. Our method unifies both modes by converting reflections into self-supervised training signals for parameter updates during deployment.

Planning and Reinforcement Learning for Embodied Agents. Our work relates to, but differs from, classical planning and RL. Planning methods such as CEM [Rubinstein, 1999] and MPPI [Williams et al., 2017] optimize action sequences over fixed dynamics without deployment-time parameter updates, corresponding to single-loop learning [Argyris and Schön, 1977]: correcting actions without revising failure-producing assumptions. Model-free RL methods such as PPO [Schulman et al., 2017a] and SAC [Haarnoja et al., 2018] learn from scalar rewards across many episodes and resets, requiring reward engineering rather than single-episode adaptation. Model-based RL methods such as DreamerV3 [Hafner et al., 2023] learn world models but typically keep them fixed during deployment. In contrast, we implement double-loop learning, updating the agent's internal model within a single episode through language-diagnosed failure signals without resets or reward engineering.

3 Reflective Test-Time Planning

Consider an embodied agent with a multimodal large language model operating on task τ in a partially observable environment. At each timestep t, the model receives observation o_t, generates action a_t in natural language, and receives execution feedback e_t indicating whether the action executed successfully (e.g., object grasped, placement completed). Crucially, a positive e_t indicates successful execution but does not imply the action was strategically correct or contributes to full task completion.

Traditional multimodal embodied LLM systems keep model parameters fixed at inference, limiting adaptation to novel scenarios or recovery from failures. We depart from static inference by building an adaptive test-time framework that employs three interacting models: an action generation LLM π_θ that produces actions given observations, an internal reflection LLM V_{ϕ_i} that generates pre-action evaluations, and an external reflection LLM V_{ϕ_e} that generates post-execution assessments. These LLMs are first initialized with basic capabilities for reasoning in embodied environments through minimum supervised fine-tuning on a small set of tasks, enabling them to understand action formats, generate reflections, and process 3D observations before they can effectively learn from test-time experience. At deployment time, we introduce three reflection types: internal reflection f_i for pre-action scoring, external reflection f_e for post-execution assessment, and retrospective reflection f_r

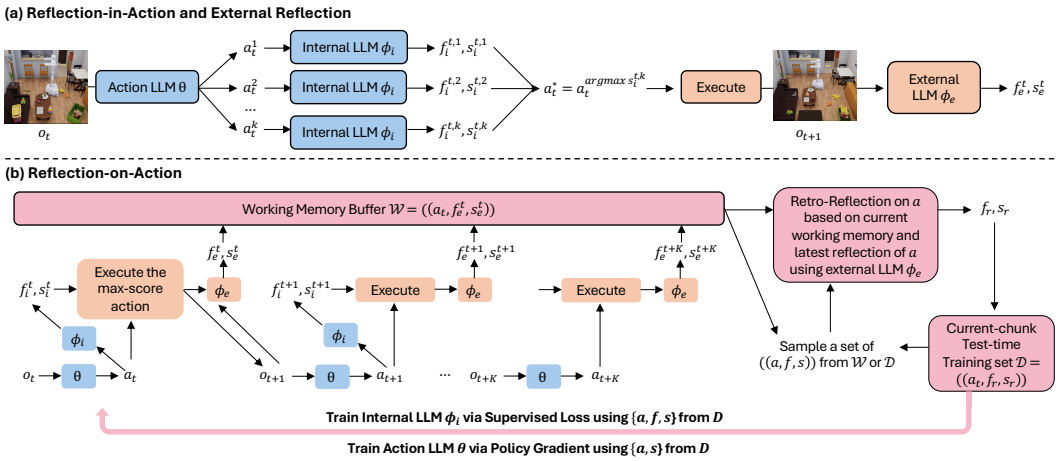

Figure 2: **Method overview.** (a) *Reflection-in-action*: multiple candidate actions are generated and scored by an internal reflection LLM prior to execution. (b) *Reflection-on-action*: iteratively invoked when working memory hits K or at key milestones. Executed actions are critiqued by an external reflection LLM and stored in a working memory buffer; at milestones, hindsight re-evaluation assigns long-horizon credit. The resulting verbal reflections form self-supervised training data to update both the internal reflection LLM (supervised loss) and the action LLM (policy gradient) via test-time training, enabling agents to learn from execution experience during deployment.

for hindsight re-evaluation. We combine test-time scaling (generating and scoring multiple candidate actions) for reflection-in-action, with test-time training (tuning π_θ and V_{ϕ_i}) for reflection-on-action. Figure 2 shows a method overview, and Algorithm 1 provides a detailed method breakdown.

We choose verbal reflection as the representation for all reflection types inspired by "double-loop learning" [Argyris and Schön, 1977]: by articulating what went wrong and why, the agent abstracts generalizable lessons transferable to future decisions rather than reporting that an action failed. These articulated lessons serve as supervisory signals during deployment, providing interpretable feedback to be reused later. Thus, instead of only training the action model based on outcomes during test time (single loop), we also train the internal reflection LLM to align its pre-action internal reflections with post-execution external reflections, updating the underlying reasoning process behind the action itself.

3.1 Reflection-in-Action

Humans naturally deliberate under uncertainty by *mentally* simulating and reflecting on actions. We transfer this ability to embodied agents via *reflection-in-action*: rather than greedily selecting the first plausible action, the agent samples candidates and reflects on each one before committing. We implement this through test-time scaling, where we generate N diverse candidate actions and use the internal reflection LLM to produce reflective evaluations for each, which guide action selection.

Vanilla Action Generation. Standard autoregressive generation from the action LLM π_θ produces a single action via greedy or low-temperature sampling:

$$a_t = \arg\max_{a \in \mathcal{A}} p_\theta(a|o_t) \tag{1}$$

where o_t is the current observation, with some architectures implicitly encoding historical context through memory structures (*e.g.*, Hu et al. [2025]). Greedy generation commits to actions early without reflecting on potential consequences.

Candidates Generation. Different from the above, at each decision step t, we construct an action generation prompt x_{action} containing task description τ, current observation o_t, previous action a_{t-1}, and previous external reflection f_e^{t-1} (both initialized as None; external reflection will be introduced later). We sample N diverse candidate actions:

$$a_t^k \sim p_\theta(\cdot|x_{\text{action}}; T) \quad k = 1, ..., N \tag{2}$$

Algorithm 1 Reflective Test-Time Planning

Require: Task τ, initial observation o_1; action LLM π_θ, internal reflection LLM V_{ϕ_i}, external reflection LLM V_{ϕ_e}; window size K

Initialize a_0, $f_e^0 \leftarrow$ None; buffers $\mathcal{W}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{retro}} \leftarrow \emptyset$; temperature T; max step $\mathcal{T}$
for $t = 1, \ldots, \mathcal{T}$ **do**
  // Reflection-in-Action
  Construct x_{action} from $(\tau, o_t, a_{t-1}, f_e^{t-1})$ and sample $\{a_t^k\}_{k=1}^N \sim \pi_\theta(\cdot|x_{\text{action}}; T)$
  for $k = 1, \ldots, N$ **do**
    Construct x_{internal}^k and score $f_i^{t,k}, s_i^{t,k} \leftarrow V_{\phi_i}(x_{\text{internal}}^k)$
  end for
  Select $a_t^* = a_t^{(\arg\max_k s_i^{t,k})}$; execute a_t^* and observe (o_{t+1}, e_t); break if task complete
  // Reflection-on-Action
  Construct x_{external} and generate feedback $f_e^t, s_e^t \leftarrow V_{\phi_e}(x_{\text{external}})$
  $\mathcal{W} \leftarrow \mathcal{W} \cup \{(o_t, a_t^*, f_e^t, s_e^t)\}$
  if $|\mathcal{W}| = K$ **or** hit key milestone **then**
    // Retrospective Reflection: Re-evaluate with Hindsight
    for each $(o_j, a_j, f^j, s^j) \in \mathcal{W} \cup \mathcal{D}_{\text{retro}}$ **do**
      Construct x_{retro}^j from $(\tau, a_j, \mathcal{W}, o_{t+1}, f^j, s^j)$
      Generate $f_r^j, s_r^j \leftarrow V_{\phi_e}(x_{\text{retro}}^j)$ and set $\mathcal{D}_{\text{retro}}[a_j] \leftarrow (a_j, f_r^j, s_r^j)$
    end for
    $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{retro}}$
    // Regularization: Prevent Catastrophic Forgetting
    for sampled unexplored action a_l **do**
      Construct x_{internal}^l; get original output $f_i^l, s_i^l \leftarrow V_{\phi_i}(x_{\text{internal}}^l)$; add (a_l, f_i^l, s_i^l) to $\mathcal{D}_{\text{train}}$
    end for
    // Test-Time Training
    for $(a, f, s_r) \in \mathcal{D}_{\text{train}}$ **do**
      Construct $x_{\text{internal}}, x_{\text{action}}$ from a; reward $r = 2s_r/100 - 1$
      Update internal LLM: $\ell_\phi = -\log p_{\phi_i}(f|x_{\text{internal}})$, $\phi_i \leftarrow \phi_i - \eta_\phi \nabla_{\phi_i} \ell_\phi$
      Update action LLM: $\ell_\theta = -r \log p_\theta(a|x_{\text{action}})$, $\theta \leftarrow \theta - \eta_\theta \nabla_\theta \ell_\theta$
    end for
    Clear working memory: $\mathcal{W} \leftarrow \emptyset$
  end if
end for

where high temperature T encourages diversity in the generated candidates.

Internal Reflection Scoring. For each candidate, we construct an internal reflection prompt x_{internal}^k, which is identical to x_{action} except that it adds the candidate action a_t^k to be evaluated. The internal reflection LLM generates:

$$f_i^{t,k}, s_i^{t,k} = V_{\phi_i}(x_{\text{internal}}^k) \tag{3}$$

where $s_i \in [0, 100]$ is a numerical score and $f_i^{t,k}$ is natural language reflection. Because this reflection occurs prior to action execution, we refer to it as internal reflection.

Best Action Selection. We select the highest-scoring candidate:

$$a_t^* = a_t^{(\arg\max_{k \in [N]} s_i^{t,k})} \tag{4}$$

Rather than executing the first feasible action, the agent "mentally tries out" multiple options and chooses the one it internally judges as most promising.

3.2 Reflection-on-Action

Reflection-in-action has a limitation: internal reflection operates in imagination, not reality. It may score an action highly based on plausible reasoning, yet the action fails due to unforeseen physical constraints or environmental dynamics. Reflection-on-action, learning from experience after actions are executed, addresses this by grounding learning in actual execution outcomes.

3.2.1 Multi-Scale External Reflection

External Reflection Generation. After executing a_t^* and observing (o_{t+1}, e_t), we construct x_{external} by extending x_{action} with a_t^*, e_t, and (o_t, o_{t+1}). The external reflection LLM generates:

$$f_e^t, s_e^t = V_{\phi_e}(x_{\text{external}}) \tag{5}$$

where s_e^t is a score and f_e^t is language feedback assessing the immediate outcome and its cause. This provides real-time assessment based on directly observable consequences.

Working Memory Buffer. We maintain a buffer $\mathcal{W}_t = (o_j, a_j, e_j, f_e^j) \mid j = t - K + 1, ..., t$ with K steps. This buffer accumulates recent experience until reaching a key milestone (e.g., exiting a room, detecting repeated failures that need replanning) or when $|\mathcal{W}_t| = K$, at which point we trigger memory consolidation and test-time training.

Retrospective Reflection with Hindsight. A critical limitation of external reflection is that it evaluates actions based on immediate outcomes. An action may appear successful initially but later prove problematic (e.g., placing an object in an accessible compartment that blocks the only space for a larger object). To address this credit assignment problem, we introduce retrospective reflection.

Once we hit key milestone or reach the working memory limit, for each action a_j that has been reflected before (either by immediate external reflection or by previous retro-reflection), the external reflection LLM re-evaluates the action with full hindsight:

$$f_r^j, s_r^j = V_{\phi_e}(x_{\text{retro}}^j) \tag{6}$$

where the retrospective prompt x_{retro}^j includes: (1) the complete working memory window $\mathcal{W}_t$ as context; (2) a historical action a_j to be retro-reflected based on the current outcomes; (3) its most recent reflection f_{recent}^j (either f_e^j from the current working memory $\mathcal{W}_t$ if this is the first retrospective evaluation, or f_r^j from the previous retrospective round and stored in a retro-buffer $\mathcal{D}_{\text{retro}}$); and (4) the current observation o_{t+1}. After retro-revision, we update $\mathcal{D}_{\text{retro}}$ and store only the most recent retro-reflection for each action. As $\mathcal{W} \cup \mathcal{D}_{\text{retro}}$ grows larger with actions, we may subsample historical actions if necessary to keep it tractable.

3.2.2 Test-Time Training Dataset Construction.

We construct training dataset $\mathcal{D}_{\text{train}}$ with two types of data:

Retro-supervised pairs: For any action a_j that has been retrospectively evaluated, we create training pairs using the retrospective reflection f_r^j and s_r^j:

$$\mathcal{D}_{\text{retro}} = (a_j, f_r^j, s_r^j) \tag{7}$$

These pairs use hindsight-corrected reflections and scores for training both the internal LLM V_{ϕ_i} and action LLM π_θ.

Regularization pairs: To prevent catastrophic forgetting, we randomly sample unexplored actions a_l, construct x_{internal}^l and use the internal LLM's current predictions:

$$\mathcal{D}_{\text{reg}} = (a_l, f_i^l, s_i^l) \tag{8}$$

where $f_i^l, s_i^l = V_{\phi_i}(x_{\text{internal}}^l)$ represents the model's current output for randomly sampled action a_l. This anchors the model to its existing knowledge for actions not updated by recent experience, preventing distribution shift caused by training exclusively on retrospectively evaluated actions.

3.2.3 Test-Time Training

Internal Reflection LLM Training via Supervised Learning. We train the internal reflection LLM V_{ϕ_i} to predict retrospective reflections using standard supervised learning. The objective minimizes negative log-likelihood over the combined dataset $\mathcal{D}_{\text{train}} = \mathcal{D}_{\text{retro}} \cup \mathcal{D}_{\text{reg}}$:

$$\mathcal{L}_{\text{internal}}(\phi_i) = \mathbb{E}(x_{\text{internal}}, f, s) \sim \mathcal{D}_{\text{train}}[-\log p_{\phi_i}(f|x)] \tag{9}$$

Where we construct x_{internal} from each action a. We perform E epochs of test-time training:

$$\phi_i^{(e+1)} = \phi_i^{(e)} - \eta_\phi \nabla_{\phi_i} \mathcal{L}_{\text{internal}} \tag{10}$$

Action LLM Training via RL. The action LLM π_θ is updated using policy gradient with retrospective scores as rewards. We convert the retrospective score $s_r \in [0, 100]$ to a reward signal $r = 2 * (s_r/100) - 1$ mapping scores to $[-1, 1]$. For each training example, we compute the log-probability of the executed action sequence:

$$\log p_\theta(a|x_{\text{action}}) = \sum_{i=1}^{|a|} \log p_\theta(a_i|a_{<i}, x_{\text{action}}) \tag{11}$$

	Baselines						Reflective Test-Time Planning Ablations					
	Reflexion	Self-Refine	ReflectVLM	PPO	DreamerV3	3DLLM-Mem	w/o RIA/ROA	w/o RIA	w/o ROA	w/o Act.	w/o Int.	**Ours**
Fitting	8.51%	10.6%	2.12%	0%	4.26%	10.6%	0%	33.5%	6.38%	25.5%	12.8%	**44.7%**
Selection	8.82%	11.8%	5.88%	2.94%	11.8%	14.7%	17.6%	5.88%	11.8%	26.5%	8.82%	**32.4%**
Preparation	15.9%	12.7%	14.3%	7.94%	11.1%	9.52%	11.1%	3.17%	19.0%	20.6%	17.5%	**31.7%**
Hybrid	6.45%	9.68%	6.45%	3.23%	12.9%	9.68%	12.9%	3.23%	12.9%	16.1%	9.68%	**25.8%**
Average	9.92%	11.20%	7.19%	3.53%	10.02%	11.13%	10.40%	11.45%	12.52%	22.18%	12.20%	**33.65%**

Table 1: Baseline comparisons and ablations of Reflective Test-Time Planning on long-horizon household tasks. RIA denotes reflection-in-action and ROA denotes reflection-on-action. Act. denotes action model loss; Int. denotes internal reflection model loss.

where the sum is over action tokens. We construct x_{action} from each action a. The REINFORCE loss is:

$$\ell_\theta = -r \cdot \log p_\theta(a|x_{\text{action}}) \tag{12}$$

This gradient increases the probability of actions with positive rewards and decreases the probability of actions with negative rewards. We accumulate gradients over all examples in $\mathcal{D}_{\text{train}}$ over several RL_steps:

$$\theta^{(s+1)} = \theta^{(s)} - \eta_\theta \nabla_\theta \sum_{(x_{\text{action}}, f, s_r) \in \mathcal{D}_{\text{train}}} \ell_\theta(x_{\text{action}}, f, s_r) \tag{13}$$

4 Experiments on Long-Horizon Household Tasks

4.1 Long-Horizon Household Task Construction

To evaluate our framework on tasks requiring multi-step reasoning and failure recovery, we construct Long-Horizon Household Tasks based on the BEHAVIOR-1K [Li et al., 2024] environments. Inspired from household scenarios, we define four task categories: (1) **Fitting**, where objects must be packed or placed into constrained containers or surfaces, stressing geometry, capacity and occlusion failures; (2) **Selection**, where the agent compares and retrieves the most suitable item (*e.g.*, in terms of preferences or sizes). Failures occur when choices prove suboptimal upon discovering better alternatives; (3) **Preparation**, where tasks require sequential constraints and dependencies (e.g., assembling or nested placement), stressing sequential dependency and non-local failures; and (4) **Hybrid**, where multiple modes appear within a single episode, stressing mixed spatial, relational, and occlusion failures.

Task Generation & Validation. We employ GPT-5 to generate task specifications through a structured prompting procedure using BEHAVIOR-1K scene graphs. Each generated task instance includes a task description, 3–7 relevant rooms, new objects with placement specifications, and a complete trajectory with interleaved actions, reflections, and scores. Since scene graphs provide object properties such as 3D bounding boxes, GPT-5 can deduce potential failures (e.g., size mismatches, occlusion) during generation. Generated trajectories are then executed in OmniGibson physics simulation to validate consistency with actual scene dynamics, rejecting instances where annotated outcomes conflict with simulator results. Full prompting and validation details are in Appendix P.

Finetuning, Evaluation & Implementation. Each validated trajectory yields (observation, action, reflection, score) tuples that form SFT pairs to initialize our three embodied LLMs before test-time deployment. Given SFT data, we build on LLaVA-3D [Zhu et al., 2025] and train a single unified model on all three modes (action generation, internal reflection, external reflection) for cross-mode learning, then instantiate three copies at deployment as π_θ, V_{ϕ_i}, and V_{ϕ_e}. During evaluation, the agent is given only the scene configuration and task description and must autonomously generate and execute its trajectory step by step. Finetuning and evaluation sets have no overlap in task descriptions, scene configurations, or object placements. A task succeeds if all required objects reach target locations within the action budget. Full model architecture, training hyperparameters, and baseline implementation details are provided in Appendix P.7- P.9.

4.2 Experimental Results & Analysis

Results Analysis. Table 1 shows substantial improvements of our model over both baselines and ablations across all task categories. Fitting tasks benefit most (44.7% vs. 10.6% for 3DLLM-Mem, 2.1% for ReflectVLM, and 0% for PPO), as their tight spatial constraints demand iterative refinement

#Data	1 Epoch	2 Epoch	3 Epoch	w/o RIA/ROA
25%	15.6	20.3	22.1	1.3
50%	18.7	24.8	19.6	4.4
100% (Ours)	33.6	20.7	16.1	10.4
200%	24.5	19.6	9.1	13.4
300%	20.7	17.5	7.5	15.9

Table 2: SFT scaling analysis. Too little data fails to produce correct formats; too much overfits to SFT distribution, destabilizing TTT.

Method	Fit	Sel.	Prep.	Hyb.	Avg.	Overhead
TTT on Ext. Refl.	22.4	17.6	12.7	9.69	15.6	$\sim 4\times$
TTT on Retro (Ours)	44.7	32.4	31.7	25.8	33.7	$1\times$

Table 3: Retrospective vs. immediate external reflection as TTT signal. Retro-reflection achieves better credit assignment at lower overhead, as external reflection triggers at every step.

and continuous adjustment of 3D geometric understanding based on execution feedback. The ablation studies reveal that RIA and ROA are mutually dependent: removing either causes performance degradation, and sometimes removing just one component performs worse than removing both. Without RIA, Preparation falls to 3.17% and Hybrid to 3.23%, below the 11.1% and 12.9% achieved by removing both components. RIA without ROA produces overconfident yet inaccurate action scores with no hindsight correction, while ROA without RIA wastes learning on poorly chosen actions that fail to reveal true scene affordances. Together they form a virtuous cycle: best-of-N selection yields higher-quality trajectories for learning, and TTT refines the internal model, leading to better future action selection. The loss ablations confirm the same mutual dependency between the action loss and internal reflection loss: without the internal loss, Hybrid drops to 9.68% and Selection to 8.82%, both below their respective w/o ROA baselines. Table 5 shows zero-shot generalization to photorealistic HM3D environments. Our model outperforms all baselines by a large margin despite training exclusively on synthetic BEHAVIOR-1K scenes. The qualitative example in Figure 5(a) further illustrates continual learning and active perception driven by accumulated scene experience. Test-time cost analysis in Appendix C shows even with substantially more test-time compute, baselines continue to repeat failures. Retro-reflection quality, search-based baselines, efficiency improvements, commercial LLM baselines, and generalization to human instructions are in Appendices D, H– K.

Necessity and Reliability of SFT Initialization and External LLM. A key design question is how much our framework relies on the quality of GPT-5-generated initialization data and the fixed external LLM. Table 2 analyzes the role of SFT initialization and establishes three points. (1) SFT is necessary but minimal. GPT-5-generated data serves only as a minimal bootstrap: it teaches the model to produce structured action formats and reflections, not to solve tasks. Without any initialization, test-time reflection yields near-zero performance regardless of whether few-shot in-context examples are provided, confirming necessity (Appendix G); the inverted-U pattern shows that too little data produces malformed outputs while too much overfits to GPT-5's reasoning distribution and destabilizes TTT. (2) Gains come from reflection, not initialization. The initialization-only variant achieves only 10.4% average success rate against 33.7% for the full model. While GPT-5 provides format, the reasoning, recovery, and adaptation abilities that drive performance are acquired entirely through test-time interaction with the environment. (3) The external LLM provides a reliable and independent learning signal. Table 2 shows that even with minimal SFT initialization, V_{ϕ_e} consistently drives meaningful performance gains across all data fractions and epoch counts. This is because V_{ϕ_e} acts as a post-hoc describer of directly observed outcomes grounded in e_t from the simulator, unlike π_θ and V_{ϕ_i} which must reason about unobserved futures. Human annotators confirm high factual correctness, causal quality, and usefulness of GPT-5 data and external reflection LLM for even out-of-domain scenarios (Appendix F&L). However, Table 3 shows that applying this signal at every step yields only 15.6% at $\sim 4\times$ overhead, demonstrating that reliable description alone is insufficient without long-horizon credit assignment via retrospective reflection at key milestones.

5 Experiments on the Cupboard Fitting Task



Figure 4: The Cupboard Fitting Task.

Cupboard Fitting Task Design. The Cupboard Fitting task serves as a complementary benchmark to Long-Horizon Household Tasks. While BEHAVIOR provides realistic multi-room environments with complex semantic reasoning challenges, it also introduces environmental uncertainties that make it difficult to isolate learning mechanisms. We design Cupboard Fitting as a controlled MuJoCo environment where agents learn from

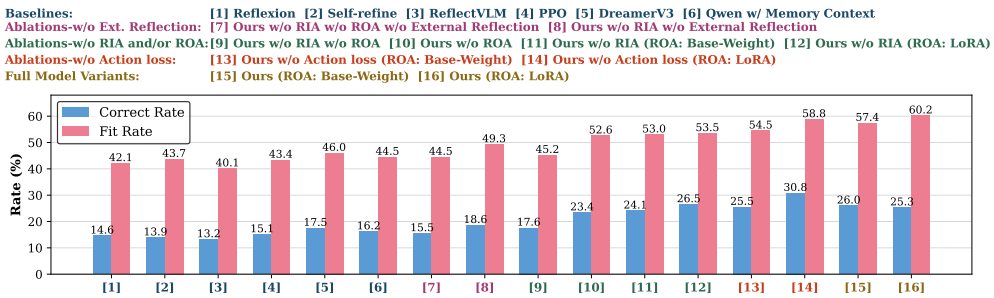

Figure 3: **Cupboard Fitting results**. Blue bars show correct-placement rate, pink bars show fit rate. RIA means Reflection-in-action; ROA means Reflection-on-action. "W/o external reflection": we don't use external reflection as the input to the action generation LLM. We implement two test-time training variants for ROA: 1) test-time training on all base weights; 2) test-time training on LoRA parameters only. Reflective Test-Time Planning significantly improves both success metrics.

placement failures, enabling precise measurement of reflective test-time training mechanisms. The environment features a multi-compartment cupboard and a set of colored geometric objects that a Franka Panda arm must place via high-level natural language commands. Success requires reasoning about object-compartment compatibility, multi-object spatial packing, and long-horizon dependencies where early placement decisions affect later possibilities. We define two metrics: *fit rate* measures objects successfully placed in any compartment, and *correct rate* measures objects placed in their designated compartments. Full environment and implementation details are in Appendix Q.

Experimental Results & Analysis. Figure 3 shows that our full method achieves 60.2% fit rate and 25.3% correct rate, substantially outperforming all baselines. Both reflection mechanisms are essential: removing RIA drops performance to 53.5% and removing ROA to 45.2% fit rate, with removing both degrading further to 44.5%. LoRA-based TTT (60.2%) performs comparably to full base-weight training (57.4%) while reducing trainable parameters, confirming that reflection-in-action for candidate selection and reflection-on-action for test-time adaptation together enable effective learning from failures during deployment. Appendix O also shows that our method enables long-horizon planning by retrospective reflection that outperforms Receding Horizon Planning while saving 5× test-time compute. Hyperparameter analyses of both ROA & RIA are provided in Appendix N.

Setting	Fit %	Correct %
Zero-shot, w/o RIA & ROA	13.1	4.5
Fine-tuned, w/o RIA & ROA	20.7	7.2
Zero-shot, full	40.6	13.4
Fine-tuned, full	44.2	16.6

Table 4: Real-robot Cupboard Fitting on a Franka Panda arm. Zero-shot denotes direct sim-to-real transfer.

Method	Preparation (%)
Reflexion	2.44
Self-Refine	4.88
ReflectVLM	0.00
PPO	0.00
DreamerV3	2.44
3DLLM-Mem	7.32
Ours	**19.5**

Table 5: Zero-shot generalization to HM3D.

Real-Robot Experiments. Table 4 reports results on a Franka Panda arm under both zero-shot and fine-tuned settings. The full model generalizes substantially better than the ablation without RIA and ROA, with the performance gap widening in the real-world setting: real-world uncertainties such as grasp imprecision and object slippage introduce the kind of unexpected failures that reflection-on-action is designed to recover from. Notably, even without any fine-tuning, the full model outperforms the zero-shot fine-tuned ablation, suggesting that active failure recovery matters more than domain adaptation when execution errors are the primary bottleneck. The qualitative example in Figure 5(b) further illustrates how retrospective reflection enables the robot to correct earlier placement decisions mid-episode. Full real-robot setup details are provided in Appendix M.

6 Conclusion

We introduce Reflective Test-Time Planning, which couples reflection-in-action for pre-action evaluation with reflection-on-action for post-execution assessments. Evaluations across two newly-designed

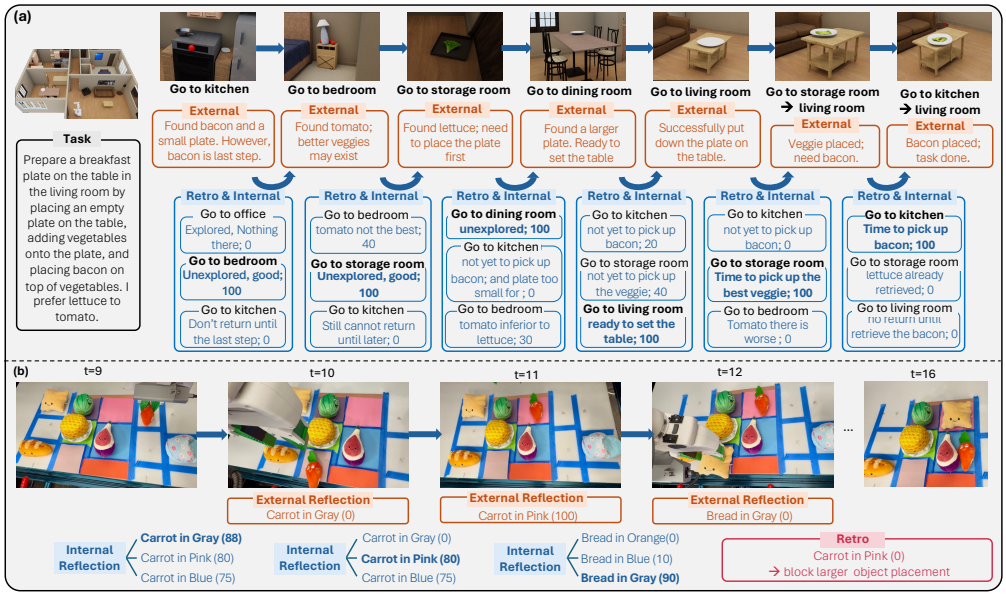

Figure 5: **Qualitative Examples.** Steps and reflections simplified for better presentations. Blue text: internal reflection. Orange text: external reflection. red text: retrospective reflection. (a) Long-Horizon Household example. We use retro & internal because the generated retro reflection is also used to train the internal model. (b) Real-robot Cupboard Fitting example. We put reflection scores inside brackets, omit detailed reflections and only present the scores for simplicity.

embodied tasks demonstrate strong gains and highlight the complementary roles of the two reflection modes. Future work may extend reflective adaptation to richer sensory modalities (*e.g.*, tactile).

Acknowledgments and Disclosure of Funding

This paper is in part funded by ONR MURI N00014-24-1-2748, ONR MURI N00014-22-1-2740, Singapore MDDI AIVP-2025-008, IITP funded by the Korean Government (MSIT) (No. RS-2024-00457882, National AI Research Lab Project), and Grant AW1134392 (Reasoning in Motion) from the TRI University 3.0 Program. This work used computational resources provided by Google through the Google Gemini Academic Program.

References

Chris Argyris and Donald Schön. *Theory in Practice: Increasing Professional Effectiveness*. Jossey-Bass, 1977. 2, 3, 4

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023. 3

Sri Harsha Dumpala, Chandramouli Sastry, and Sageev Oore. Test-time training for speech, 2023. URL https://arxiv.org/abs/2309.10930. 3

Cunxin Fan, Xiaosong Jia, Yihang Sun, Yixiao Wang, Jianglan Wei, Ziyang Gong, Xiangyu Zhao, Masayoshi Tomizuka, Xue Yang, Junchi Yan, and Mingyu Ding. Interleave-vla: Enhancing robot manipulation with interleaved image-text instructions. 2025. URL https://arxiv.org/abs/2505.02152. 3

Yunhai Feng, Jiaming Han, Zhuoran Yang, Xiangyu Yue, Sergey Levine, and Jianlan Luo. Reflective planning: Vision-language models for multi-stage long-horizon robotic manipulation, 2025. URL https://arxiv.org/abs/2502.16707. 2, 3, 36

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2024. 3

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, pages 1861–1870, 2018. 3

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023. 3

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. URL https://arxiv.org/abs/2301.04104. 3

Elvin Hajizada, Balachandran Swaminathan, and Yulia Sandamirskaya. Continual learning for autonomous robots: A prototype-based approach. 2024. URL https://arxiv.org/abs/2404.00418. 3

Yining Hong, Haoyu Zhen, Peihao Chen, Shuhong Zheng, Yilun Du, Zhenfang Chen, and Chuang Gan. 3D-LLM: Injecting the 3D world into large language models. In *Advances in Neural Information Processing Systems*, 2023. 3

Yining Hong, Zishuo Zheng, Peihao Chen, Yian Wang, Junyan Li, and Chuang Gan. MultiPLY: A multisensory object-centric embodied large language model in 3D world. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26406–26416, 2024. 2, 3

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. *arXiv preprint arXiv:2106.09685*, 2021. 3

Wenbo Hu, Yining Hong, Yanjun Wang, Leison Gao, Zibu Wei, Xingcheng Yao, Nanyun Peng, Yonatan Bitton, Idan Szpektor, and Kai-Wei Chang. 3dllm-mem: Long-term spatial-temporal memory for embodied 3d large language model, 2025. URL https://arxiv.org/abs/2505.22657. 3, 4, 35, 36

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *arXiv preprint arXiv:2207.05608*, 2022. 3

Isaac Kauvar, Chris Doyle, Linqi Zhou, and Nick Haber. Curious replay for model-based adaptation. *International Conference on Machine Learning*, 2024. 3

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Perez Foster, Pannag Raj Sanketi, Quan Vuong, et al. OpenVLA: An open-source vision-language-action model. In *8th Annual Conference on Robot Learning*, 2024. 2, 3

Yuto Kojima, Jiarui Xu, Xueyan Zou, and Xiaolong Wang. Lora-ttt: Low-rank test-time training for vision-language models. 2025. URL https://arxiv.org/abs/2502.02069. 3

Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information Fusion*, 58:52–68, 2020. 3

Chengmeng Li, Junjie Wen, Yan Peng, Yaxin Peng, Feifei Feng, and Yichen Zhu. Pointvla: Injecting the 3d world into vision-language-action models. 2025. URL https://arxiv.org/abs/2503.07511. 3

Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Wensi Ai, Benjamin Martinez, Hang Yin, Michael Lingelbach, Minjune Hwang, Ayano Hiranaka, Sujay Garlanka, Arman Aydin, Sharon Lee, Jiankai Sun, Mona Anvari, Manasi Sharma, Dhruva Bansal, Samuel Hunter, Kyu-Young Kim, Alan Lou, Caleb R Matthews, Ivan Villa-Renteria, Jerry Huayang Tang, Claire Tang, Fei Xia, Yunzhu Li, Silvio Savarese, Hyowon Gweon, C. Karen Liu, Jiajun Wu, and Li Fei-Fei. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*, 2024. 7, 31

Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025. 3

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2023. 2, 3, 36

Russell Mendonca, Emmanuel Panov, Bernadette Bucher, Jiuguang Wang, and Deepak Pathak. Continuously improving mobile manipulation with autonomous real-world RL. In *Conference on Robot Learning*, 2024. 3

Yuan Meng, Zhenshan Bing, Xiangtong Yao, Kejia Chen, Kai Huang, Yang Gao, Fuchun Sun, and Alois Knoll. Preserving and combining knowledge in robotic lifelong reinforcement learning. *Nature Machine Intelligence*, pages 1–14, 2025. 3

Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought, 2023. URL https://arxiv.org/abs/2305.15021. 3

Andrew Ng. Agentic design patterns part 2: Reflection. *DeepLearning.AI The Batch*, 2024. URL https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-2-reflection/. 3

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shuai Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting. In *International Conference on Machine Learning*, pages 16888–16905, 2022. 3

Santhosh K. Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X. Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai, 2021. URL https://arxiv.org/abs/2109.08238. 23

Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1(2):127–190, 1999. 2, 3

Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, NY, USA, 1992. 2

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017a. 2, 3

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017b. URL https://arxiv.org/abs/1707.06347. 36

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2023. 2, 3, 36

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International Conference on Machine Learning*, pages 9229–9248, 2020. 3

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): RNNs with expressive hidden states. In *International Conference on Machine Learning*, 2024. 3

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. In *International Conference on Learning Representations*, 2021. 3

Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M Rehg, Byron Boots, and Evangelos A Theodorou. Information theoretic MPC for model-based reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1714–1721, 2017. 2, 3

Hao Yang, Min Wang, Jinshen Jiang, and Yun Zhou. Towards test time adaptation via calibrated entropy minimization. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 3736–3746, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671672. URL https://doi.org/10.1145/3637528.3671672. 3

Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. Self-contrast: Better reflection through inconsistent solving perspectives. 2024. URL https://arxiv.org/abs/2401.02009. 3

Qingqing Zhao, Yao Lu, Moo Jin Kim, Zipeng Fu, Zhuoyang Zhang, Yecheng Wu, Zhaoshuo Li, Qianli Ma, Song Han, Chelsea Finn, et al. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1702–1713, 2025. 3

H. Zhen et al. Learning 4d embodied world models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 3

Haoyu Zhen, Xiaowen Qiu, Peihao Chen, Jincheng Yang, Xin Yan, Yilun Du, Yining Hong, and Chuang Gan. 3D-VLA: A 3D vision-language-action generative world model. *arXiv preprint arXiv:2403.09631*, 2024. 2, 3

Siyuan Zhou, Yilun Du, Jiaben Chen, Yandong Li, Dit-Yan Yeung, and Chuang Gan. Robodreamer: Learning compositional world models for robot imagination, 2024. URL https://arxiv.org/abs/2404.12377. 36

Chenming Zhu, Tai Wang, Wenwei Zhang, Jiangmiao Pang, and Xihui Liu. LLaVA-3D: A simple yet effective pathway to empowering LMMs with 3D-awareness. In *International Conference on Computer Vision*, 2025. 3, 7, 35

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023. 2, 3

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the three contributions: the Reflective Test-Time Planning framework, the three reflection mechanisms (reflection-in-action, reflection-on-action, retrospective reflection), and the two newly designed benchmarks. All claims are supported by experimental results in Sections 4 and 5.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations are discussed in the conclusion and broader impacts section, including the $\sim 3\times$ test-time overhead, reliance on a minimal SFT initialization, and the current restriction to high-level language actions without low-level control.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [N/A]

Justification: The paper does not include theoretical results or proofs.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: All training hyperparameters, model architectures, LoRA configurations, optimizer settings, and evaluation protocols are described in the experiment sections and Appendices P and Q. Code and data are provided in the supplemental material.

Guidelines:

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

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

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

   Answer: [Yes]

   Justification: Code and data are included in the supplemental material with instructions for reproducing all main experimental results.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Training details for both benchmarks are provided in the main paper and in Appendices P and Q, including optimizer, learning rate, batch size, LoRA configuration, and evaluation protocol. Hyperparameter selection is justified via ablation studies in Appendix N.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [No]

   Justification: Error bars are not reported. Each experiment involves full task execution across all evaluation instances on fixed evaluation sets, and results are reported as task success rates over the full set rather than across repeated runs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Compute requirements including GPU type, memory, and approximate runtime are described in Appendix C.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research involves simulation environments and a controlled real-robot setup and does not involve human subjects, sensitive data, or dual-use risks beyond those discussed in the broader impacts section.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

Justification: Broader impacts are discussed in Appendix B, covering both positive impacts (safer household robots, reduced human intervention) and negative considerations (unexpected strategy development, language model biases, reduced human oversight).

Guidelines:

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

11. **Safeguards**

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

Answer: [N/A]

Justification: The paper does not release pretrained language models or scraped datasets. The benchmark and code do not pose high misuse risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: All existing assets are cited, including BEHAVIOR-1K, OmniGibson, MuJoCo, LLaVA-3D, Qwen2.5-VL, and HM3D. Licenses are respected and referenced in the supplemental material.

Guidelines:

Contents

1 Introduction **2**

2 Related Works **3**

3 Reflective Test-Time Planning **3**

3.1 Reflection-in-Action 4

3.2 Reflection-on-Action 5

3.2.1 Multi-Scale External Reflection 5

3.2.2 Test-Time Training Dataset Construction. 6

3.2.3 Test-Time Training 6

4 Experiments on Long-Horizon Household Tasks **7**

4.1 Long-Horizon Household Task Construction 7

4.2 Experimental Results & Analysis 7

5 Experiments on the Cupboard Fitting Task **8**

6 Conclusion **9**

A Contribution Statement **21**

B Broader Impacts **21**

C Test-Time Cost Analysis **21**

C.1 Computational Comparison 21

C.2 Why the Overhead is Justified 21

C.3 Compute-Matched Experiment 21

D Efficiency Improvements via Parallelization **22**

E Generalization to Habitat-Matterport 3D Scenes: More Details **23**

E.1 Experimental Setup 23

E.2 Task Construction and Adaptation 23

E.3 Implementation Details 23

F Reliability of the External Reflection Model **23**

F.1 Human Evaluation of V_{ϕ_e} Quality 23

F.2 Human Oracle Comparison 23

G SFT Initialization Analysis **24**

G.1 Zero-SFT Baselines 24

G.2 Base Model Scale Analysis 24

H Retro-Reflection Quality Analysis **24**

I Search-Based Baseline Comparison **24**

J Generalization to Human-Written Instructions **25**

K Commercial LLM Baselines **25**

L GPT-5 Data Quality Validation **25**

M Real-Robot Experiment Details **26**

N Hyperparameter Analyses for Cupboard Fitting Task **26**

N.1 Test-Time Scaling: Number of Candidate Actions 26

N.2 Sampling Temperature Analysis 28

N.3 LoRA Configuration Analysis 28

N.4 Action Budget Analysis 29

O Single-Step Action Generation vs. Receding Horizon Planning **29**

P Experiments on Long-Horizon Household Tasks: More Details **31**

P.1 Task Generation & Validation Overview 31

P.2 Why Build Upon BEHAVIOR? 31

P.3 Task Categories 32

P.4 GPT-5 Task Generation Prompting Strategy 32

P.5 Physical Validation in BEHAVIOR. 34

P.6 Training Data Construction 34

P.7 Model Architecture and Training 35

P.8 Test-Time Training Configuration 35

P.9 Evaluation Protocol and Baselines 36

Q Cupboard Fitting: More Details **36**

Q.1 Cupboard Setup 36

Q.2 Base Model and Supervised Fine-tuning. 37

Q.3 Test-Time Training Variants. 37

R More Qualitative Examples **37**

A Contribution Statement

Yining Hong proposed the idea; implemented all codes, data, experiments and wrote the paper. Huang Huang helped set up the real-world robot. The other authors contributed through advisory support and research supervision.

B Broader Impacts

The positive impact includes more robust household robots that recover from mistakes, enabling safer deployment in unstructured environments. However, autonomous behavior updates during deployment raise important considerations: agents might develop unexpected strategies that bypass safety constraints, verbal reflections could inherit language model biases, and improved failure recovery may reduce human oversight in safety-critical applications. We believe transparency through interpretable verbal reflections and careful monitoring during initial deployments can help mitigate these risks while advancing more capable and trustworthy embodied AI systems.

C Test-Time Cost Analysis

C.1 Computational Comparison

Our full model introduces two additional sources of inference time cost: (i) candidate action sampling and internal reflection scoring for reflection-in-action (RIA), and (ii) periodic reflection-on-action(ROA)-based LoRA test-time updates.

On average across Long-Horizon Household Tasks and Cupboard Fitting Tasks, we observe a $\sim 3\times$ increase in per-step wall-clock time compared to the vanilla baseline:

$$\text{Time}_{\text{full}} \approx 3.0 \times \text{Time}_{\text{no-RIA/ROA}}.$$

Importantly, this latency is incurred at deployment and does not require additional supervised data or environment rollouts beyond normal task execution.

C.2 Why the Overhead is Justified

Despite the additional test-time latency, the reflective overhead is justified for three reasons:

Deployment-Oriented Adaptation. The extra time cost occurs at deployment rather than pretraining. This aligns with realistic embodied settings where robots adapt online rather than relying on costly retraining cycles.

Reduction of Execution Waste. Vanilla agents frequently repeat failures (e.g., placing incompatible objects, revisiting rooms without intent). RIA reduces execution waste by filtering poor actions before execution, while ROA eliminates repeated failures through hindsight-driven updates. A smaller number of higher-quality actions amortizes the reflection cost.

Conversion of Time into Learning. Extra time is not spent on naive rollouts, but on improving internal models. RIA and ROA produce persistent behavioral improvements, whereas baselines spend time without updating policies or reasoning mechanisms. In short, reflective time is structurally more valuable than mere rollout time.

C.3 Compute-Matched Experiment

A natural concern is whether the performance gap is merely due to increased wall-clock time. To evaluate this, we construct a time-matched variant where the vanilla baseline receives a $3\times$ steps budget, matching the approximate inference time of our full model:

$$\text{Steps}_{\text{baseline}}^{\text{expanded}} \approx 3 \times \text{Steps}_{\text{baseline}}.$$

From Table 6, we observe that even under a tripled time budget, the baseline:

	w/o RIA w/o ROA		Ours (Full)
	Vanilla	**Same Time Budget**	
Fitting	0.00%	0.00%	44.7%
Selection	17.6%	14.7%	32.4%
Preparation	11.1%	12.7%	31.7%
Hybrid	6.45%	6.45%	25.8%
Average	8.79%	8.46%	33.65%

Table 6: Performance comparison between (1) vanilla ablation without RIA or ROA, (2) vanilla baseline with matched time budget (3× steps), and (3) our full reflective model on Long-Horizon Household Tasks.

- fails to correct early suboptimal decisions,
- frequently revisits states without strategic change,
- exhibits repeated placement/navigation failures,
- plateaus significantly below our full model on all benchmarks and does not improve over the vanilla baseline.

This result supports that reflective time is not equivalent to naive rollout time expansion: reflective updates change the decision process itself, while rollout expansion merely increases trajectory length without improving competence or hindsight reasoning.

Conclusion. These findings indicate that although our method incurs a $\sim 3\times$ test-time latency overhead, reflective computation provides unique adaptation benefits that cannot be recovered by proportional rollout-step scaling. The overhead is therefore practical and justified in embodied deployment settings.

D Efficiency Improvements via Parallelization

We explored several strategies to reduce the $\sim 3\times$ overhead without sacrificing performance. Results are shown in Table 7.

Configuration	**Success**	**Overhead**
Ours (full)	33.7%	1.00×
+Parallel Scoring + KV Cache	33.0%	0.85×
+Batched Retro-Reflection	33.5%	0.66×
+TTT Early-Stop	32.3%	0.60×
+QLoRA	32.0%	0.52×
+Multi-GPU Parallelism	31.5%	0.45×

Table 7: Cumulative efficiency improvements. Each row adds to the previous. The combined pipeline reduces overhead to $0.45\times$ with minimal performance degradation.

Parallel candidate scoring + KV cache. Candidate scoring calls are independent given the shared prefix context and can run in parallel with a shared KV cache. **Batched retro-reflection.** Retro-reflections in the working memory window can be batched in a single forward pass. **TTT early-stop.** Training halts when the loss converges rather than running for a fixed number of epochs. **QLoRA.** 4-bit quantized LoRA reduces TTT memory footprint with minimal performance degradation. **Multi-GPU parallelism.** Model parallelism, distributed LoRA updates, and asynchronous TTT updates on a separate GPU further reduce wall-clock time.

Together these techniques reduce overhead from $\sim 3\times$ to $\sim 0.45\times$ while maintaining 31.5% success rate, demonstrating that the framework is practically deployable with engineering effort.

E Generalization to Habitat-Matterport 3D Scenes: More Details

E.1 Experimental Setup

To evaluate the generalization capacity of our Reflective Test-Time Planning framework, we conduct additional experiments on the Habitat-Matterport 3D (HM3D) dataset [Ramakrishnan et al., 2021], which provides photorealistic 3D reconstructions of real-world indoor environments. Unlike BEHAVIOR-1K scenes which are primarily synthetic household environments, HM3D offers diverse real-world scenes with different spatial layouts, object distributions, and visual appearances, providing a challenging domain shift for testing our method's transferability.

We focus specifically on **Preparation tasks**, which involve sequential constraints and dependencies where actions must occur in specific orders. These tasks are particularly challenging in HM3D environments due to: (1) more complex spatial layouts with irregular room configurations, (2) diverse object appearances and placements not seen during training, (3) ambiguous spatial relationships that require active exploration to resolve, and (4) longer navigation distances between task-relevant objects.

E.2 Task Construction and Adaptation

We adapt our task generation pipeline to HM3D scenes while maintaining the same core task structure. We select 41 preparation task instances across diverse HM3D scenes, ensuring coverage of various spatial configurations and object arrangements. Each task requires the agent to: (1) navigate through multiple rooms to locate task-relevant objects, (2) retrieve objects in the correct sequential order, (3) perform placement or assembly actions with proper dependencies, and (4) handle spatial constraints specific to real-world scene layouts.

The key difference from Long-Horizon Household evaluation is the domain gap: our models are trained exclusively on BEHAVIOR-1K synthetic scenes and must generalize to HM3D's photorealistic environments at test time. This tests whether reflection-based adaptation can overcome distribution shift through deployment-time learning.

E.3 Implementation Details

We use the same model architecture (LLaVA-3D-7B) and test-time training configuration as the Long-Horizon Household experiments, with no additional fine-tuning on HM3D scenes. Point cloud observations are extracted from RGB-D sensors in the same manner, and the reflection generation prompts remain unchanged. This zero-shot transfer setup provides a rigorous test of whether reflective mechanisms learned in synthetic environments transfer to real-world scene understanding.

F Reliability of the External Reflection Model

F.1 Human Evaluation of V_{ϕ_e} Quality

Human annotators rate V_{ϕ_e} outputs on three dimensions: Factual Correctness, Causal Quality, and Usefulness as a learning signal.

Setting	Factual	Causal	Useful
In-domain (BEHAVIOR)	98	95	99
Out-of-domain (HM3D)	96	96	95

Table 8: Human evaluation of V_{ϕ_e} reflection quality (%). Quality remains high under distribution shift.

F.2 Human Oracle Comparison

To directly isolate V_{ϕ_e}'s reliability as a supervision source, we replace V_{ϕ_e}'s reflections with human-written reflections of the same execution outcomes on the fly, serving as an oracle upper bound. We

evaluate on HM3D (environment-level shift) and two held-out BEHAVIOR task types (task-level shift).

Method	HM3D	Cleanup	Rearrange
Ours w/ V_{ϕ_e}	19.5%	40.0%	26.7%
Ours w/ Human Oracle	21.9%	40.0%	33.3%

Table 9: V_{ϕ_e} vs. human oracle under distribution shift. Near-identical performance confirms V_{ϕ_e} provides supervision of equivalent quality to human judgment.

The near-identical performance confirms that V_{ϕ_e} is a reliable teacher signal under both environment-level and task-level shift. This is expected: V_{ϕ_e} serves as a post-hoc describer of directly observed outcomes grounded in a binary execution signal e_t from the simulator, not a forward predictor reasoning about unobserved futures.

G SFT Initialization Analysis

G.1 Zero-SFT Baselines

Method	Accuracy (%)
Zero-SFT, w/o RIA & ROA	0.0
Zero-SFT, w/ RIA & ROA	0.0
Zero-SFT, Few-Shot In-Context, w/o RIA & ROA	1.3
Zero-SFT, Few-Shot In-Context, w/ RIA & ROA	2.1
Ours, w/o RIA & ROA	10.4
Ours (Full)	33.7

Table 10: Zero-SFT baselines confirm that test-time reflection cannot function without minimal SFT initialization.

G.2 Base Model Scale Analysis

Table 11 analyzes the minimal level of embodied prior required for reflection to produce semantically reliable signals, across three model scales. We measure four dimensions: semantic similarity (cosine similarity between model output and SFT ground-truth, assessing whether reflections are semantically meaningful), human evaluation (annotator ratings of reflection quality), format parse rate (whether outputs follow the required action-reflection-score structure), and fit rate (full trajectory execution success with our complete reflection framework).

Without SFT, Qwen2-VL 2B shows noticeably lower semantic similarity and human evaluation scores, confirming that insufficient embodied prior leads to semantically unreliable reflections regardless of format. Qwen2.5-VL 3B and 7B without SFT already show high semantic quality but near-zero format parse rates, confirming that format is the only missing piece SFT addresses. After SFT, format is fixed across all models, but fit rate reveals the remaining story: Qwen2-VL 2B + SFT underperforms due to insufficient embodied prior; Qwen2.5-VL 3B + SFT hits the optimal fit rate; Qwen2.5-VL 7B + SFT degrades, consistent with stronger pretrained models being harder to adapt at test time.

H Retro-Reflection Quality Analysis

In Table 12, we analyze retro-reflection quality .

I Search-Based Baseline Comparison

We add Language-Guided MCTS (LG-MCTS), inspired by LLM-MCTS and MCTS-EP. LG-MCTS builds a search tree using π_θ for node expansion and V_{ϕ_i} as value estimator, with tree depth limited to 3 for tractability.

Model	Sem. Sim.	Human Eval	Format Parse	Fit %
Qwen2-VL 2B (no SFT)	0.59	76	0.19	4.7
Qwen2.5-VL 3B (no SFT)	0.73	89	0.27	8.0
Qwen2.5-VL 7B (no SFT)	0.75	91	0.31	11.9
Qwen2-VL 2B + SFT	—	—	0.95	48.5
Qwen2.5-VL 3B + SFT	—	—	0.99	60.2
Qwen2.5-VL 7B + SFT	—	—	0.99	52.7

Table 11: Base model scale analysis. Without SFT, 3B/7B models have sufficient semantic quality but near-zero format parse rates; SFT fixes format. Qwen2.5-VL 3B hits the optimal fit rate; 7B degrades due to over-parameterization.

Metric	BEHAVIOR	HM3D (OOD)
Revision Direction (%)	99	95
Human Eval — Factual	98	96
Human Eval — Causal	95	96
Human Eval — Useful	99	95

Table 12: Retro-reflection quality. Score revision directions align with ground-truth outcomes at $\geq$95% even out-of-domain, and human evaluators rate reflections as highly factual, causally grounded, and useful.

J Generalization to Human-Written Instructions

We evaluate generalization beyond GPT-5-generated task descriptions, including deliberately ambiguous instructions and two completely held-out task types never seen during SFT.

K Commercial LLM Baselines

Table 15 compares our framework against GPT-5, Gemini 2.5 Pro, and Qwen2.5 as zero-shot baselines on the Long-Horizon Household benchmark. All three receive only the scene configuration and task description at test time, the same partial observations as our model. Note that GPT-5's role in data generation is distinct: during initialization it has access to complete scene graphs including 3D bounding boxes, whereas as a zero-shot baseline it operates under the same partial observability constraints.

Our framework outperforms all three commercial models by a large margin despite using a 3B base model, with GPT-5 achieving only 20.7% and Gemini 2.5 Pro 19.7% average success rate against our 33.7%. This directly demonstrates that test-time learning, not model scale or general language model capability, drives the performance gains. Even the most capable frontier models cannot recover from failures within a single deployment episode without the reflection mechanisms our framework provides.

L GPT-5 Data Quality Validation

Table 16 validates the quality of GPT-5-generated initialization data along two axes: consistency with OmniGibson physics simulation (whether annotated outcomes match actual simulator dynamics)

Method	Fit	Sel.	Prep.	Hybrid	Avg.	Overhead
LG-MCTS	14.9	14.7	15.8	16.1	15.4	$\sim$5–8$\times$
Ours w/o TTT	6.4	11.8	19.0	12.9	12.5	0.5$\times$
Ours	44.7	32.4	31.7	25.8	33.7	1$\times$

Table 13: LG-MCTS is inferior to our full method at 5–8$\times$ cost. Search-based lookahead is ungrounded in physical uncertainties that can only be resolved through interaction; TTT, not search depth, is the key.

Instruction Type	Fit	Sel.	Prep.	Hybrid	Avg.
GPT-Generated (current)	44.7	32.4	31.7	25.8	33.7
Human-Written	42.5	33.6	25.7	27.4	32.3

Table 14: Performance generalizes to human-written instructions with marginal degradation.

Method	Fit	Sel.	Prep.	Hybrid	Avg.
GPT-5	19.2	23.5	17.4	22.6	20.7
Gemini 2.5 Pro	14.9	20.5	20.6	22.6	19.7
Qwen2.5	12.8	14.6	14.3	15.1	14.2
Ours	44.7	32.4	31.7	25.8	33.7

Table 15: Commercial LLM zero-shot baselines. Our framework outperforms all commercial models despite using a 3B base model, demonstrating that test-time learning, not model scale, drives the gains. GPT-5 as a zero-shot baseline receives only partial observations; its data-generation role uses complete scene graphs.

and consistency with human judgment (whether human annotators agree with GPT-5's trajectory assessments).

Raw GPT-5 scores achieve 85% simulator consistency and 80% human consistency. After executing all generated trajectories in OmniGibson and rejecting instances where annotated outcomes conflict with actual simulator dynamics, simulator consistency reaches 100% by construction and human consistency rises to 98%. This confirms that physics-simulation validation is an effective filter for ensuring initialization data quality, and that the resulting dataset provides a reliable foundation for the minimal SFT bootstrap described in Section G.

Method	Consistent w/ Sim.	Consistent w/ Human
GPT Raw Scores	85%	80%
GPT + Sim. Validation	100%	98%

Table 16: GPT-5 data quality after physics-simulation validation. Simulator filtering raises human consistency from 80% to 98%.

M Real-Robot Experiment Details

We use a Franka Panda robotic arm in a cupboard fitting scenario. A camera mounted above the workspace captures RGB images from a top-down offset viewpoint matching the simulation configuration. High-level language commands are executed via two phases: (1) the robot moves to the object's registered position and grasps it with the parallel-jaw gripper; (2) it places the object at the target compartment location. After each action, the system provides binary success/failure feedback based on whether the object remains within target compartment bounds.

N Hyperparameter Analyses for Cupboard Fitting Task

We conduct ablation studies on four key hyperparameters that govern the reflection mechanism and show the results in Figure 6.

N.1 Test-Time Scaling: Number of Candidate Actions

The reflection-in-action mechanism generates N candidate actions via sampling, scores each using the internal reflection model V_{ϕ_i}, and executes the highest-scoring action. We vary $N \in \{1, 2, 3, 4, 5, 6, 8, 9, 10\}$ to measure the impact of candidate diversity on task performance (Figure 6, top left).

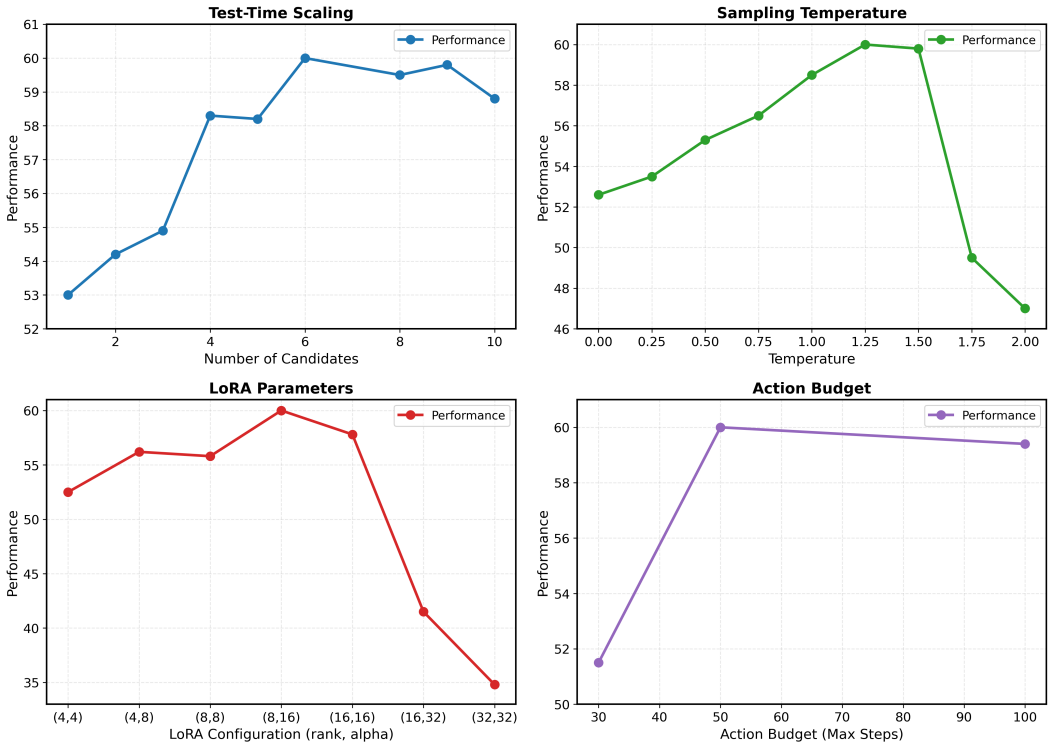

Figure 6: **Hyperparameter ablation studies on Cupboard Fitting.** *Top Left:* Performance vs. number of candidate actions. Peak performance (60.0%) occurs at N=6 candidates, demonstrating that internal reflection effectively identifies superior actions from diverse pools. Beyond N=6, performance plateaus as excessive candidates add computational cost without improving the best candidate quality. *Top Right:* Performance vs. sampling temperature. Optimal temperature range (T=1.25-1.5) balances candidate diversity with quality—temperatures below 0.5 produce overly similar candidates that limit reflection value, while temperatures above 1.75 generate incoherent actions that even accurate reflection cannot salvage. *Bottom Left:* Performance vs. LoRA configuration (rank, alpha). The optimal configuration (r=8, α=16) achieves 60.0% performance, balancing adaptation capacity with training stability. Smaller configurations like (4,4) underfit with insufficient capacity (52.5%), while larger configurations cause mode collapse during test-time training—(16,32) drops to 41.5% and (32,32) collapses to 34.8% as the model begins predicting identical outputs for all inputs, losing the ability to distinguish between different spatial configurations and task contexts. *Bottom Right:* Performance vs. action budget (maximum steps). Performance improves dramatically from 30 steps (51.5%) to 50 steps (60.0%), but slightly degrades to 59.4% at 100 steps, suggesting that excessive action budgets allow suboptimal exploration strategies that accumulate errors over longer horizons.

Dimension	Score
Task Naturalness	95
Reflection Quality	99
Score Calibration	97
Diversity	92

Table 17: Human evaluation of GPT-5-generated data quality.

Results. Performance improves monotonically from 53.0% (N=1, greedy decoding) to 60.0% (N=6), representing a 7 percentage point gain. This demonstrates that internal reflection scoring effectively identifies superior actions from diverse candidate pools. Beyond N=6, performance plateaus and slightly decreases to 58.8% at N=10, suggesting diminishing returns from excessive candidates—likely due to increased computational cost without proportional quality improvements in the candidate pool.

Analysis. The optimal operating point at N=6 balances exploration breadth with computational efficiency. At N=1, the model lacks opportunities to reconsider suboptimal greedy choices. At N=2-5, expanding the candidate pool allows the internal reflection model to compare alternatives and avoid locally optimal but globally poor actions (e.g., placing a small object in a large compartment early). At N$\geq$6, the candidate pool may include too many low-quality options that add noise without improving the best candidate's quality, and the internal reflection model's scoring may become less reliable across excessively diverse samples.

N.2 Sampling Temperature Analysis

Temperature T controls the sharpness of the probability distribution during action generation: low temperatures ($T \to 0$) produce near-greedy sampling, while high temperatures ($T \to \infty$) approach uniform sampling. We evaluate $T \in \{0.0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0\}$ (Figure 6, top right).

Results. Performance exhibits a clear inverted-U relationship with temperature. At T=0.0 (deterministic greedy decoding), performance is 52.6%. Then it increases steadily through moderate temperatures, reaching a peak of 60.0% at T=1.25, and maintains near-peak performance of 59.8% at T=1.5. However, performance drops sharply at higher temperatures: 49.5% at T=1.75 and 47.0% at T=2.0.

Analysis. The optimal temperature range T=1.25-1.5 balances two competing factors: (1) *Diversity for reflection*: Sufficient randomness generates meaningfully different candidates for the internal reflection model to compare, enabling it to identify strategic failures (e.g., blocking future placements) that greedy or near-greedy decoding would miss; (2) *Quality preservation*: Excessive randomness (T$\geq$1.5) samples from the low-probability tail of the distribution, producing incoherent or physically infeasible actions that even accurate internal reflection cannot salvage. The sharp drop at T$\geq$1.75 indicates that overly stochastic sampling overwhelms the reflection mechanism's corrective capacity—no amount of scoring can recover from fundamentally poor candidate pools.

N.3 LoRA Configuration Analysis

For parameter-efficient test-time training, we apply Low-Rank Adaptation (LoRA) with varying rank r and scaling factor α. We evaluate configurations $(r, \alpha) \in \{(4,4), (4,8), (8,8), (8,16), (16,16), (16,32), (32,32)\}$ to study the trade-off between adaptation capacity and training stability (Figure 6, bottom left).

Results. Performance exhibits a sharp peak at the intermediate configuration. Small configurations show limited performance: (4,4) achieves 52.5% and (4,8) reaches 56.2%, indicating insufficient adaptation capacity. Performance peaks at (r=8, α=16) with 60.0%, our optimal configuration. However, larger configurations show dramatic performance degradation: (8,8) maintains 55.8%, but (16,16) drops to 57.8%, (16,32) plummets to 41.5%, and (32,32) collapses catastrophically to 34.8%—below even the no-adaptation baseline of 53.0%.

Analysis. The LoRA rank controls the expressiveness of the adapter matrices, but excessive rank causes mode collapse during aggressive test-time training. At (r=4, α=4) and (r=4, α=8), the adapter

capacity is too limited to capture nuanced spatial reasoning required for effective reflection—the model cannot adequately learn from retrospective feedback about blocking placements or strategic failures. The optimal configuration (r=8, α=16) provides sufficient capacity to update internal reflection scoring and action selection policies based on task-specific feedback while maintaining stable optimization under high learning rates (0.2 for internal reflection, 0.01 for action model).

Larger configurations fail catastrophically due to mode collapse. At (r=16, α=32) and beyond, the increased parameter space combined with aggressive learning rates and limited training data (10-15 retrospective examples per test-time training iteration) causes the model to collapse to predicting identical outputs for all inputs. Rather than learning task-specific spatial reasoning, the overparameterized adapters converge to degenerate solutions that ignore input variations. At (32,32), the model effectively stops distinguishing between different object configurations, compartment sizes, or task contexts—producing the same stereotyped action regardless of the actual scene state. This mode collapse is particularly severe because test-time training lacks the regularization and diverse data that prevent collapse in offline training.

N.4 Action Budget Analysis

We evaluate the impact of maximum action budget on task performance by varying the step limit from 30 to 100 steps (Figure 6, bottom right).

Results. Performance shows a sharp initial gain followed by slight degradation. At 30 steps, performance is only 51.5%, indicating insufficient budget to complete multi-object placement tasks. Performance peaks at 50 steps with 60.0%, our optimal setting. However, extending the budget to 100 steps results in slight performance degradation to 59.4%, a 0.6 percentage point drop.

Analysis. The low performance at 30 steps reflects task incompletion—agents frequently run out of steps before placing all objects, especially when early mistakes require exploration of alternative strategies. The 50-step budget provides sufficient runway for the agent to explore the environment, execute placements, recover from initial failures through reflection, and retry with corrected strategies learned via test-time training.

The counterintuitive degradation at 100 steps reveals a failure mode of excessive budgets: when given too many steps, agents exhibit suboptimal exploration patterns that accumulate errors over longer horizons. With a generous budget, the model may attempt more exploratory actions rather than committing to placements, leading to inefficient trajectories. Additionally, longer episodes provide more opportunities for compounding errors—a single poor placement early in a 100-step episode has more downstream consequences than in a tightly constrained 50-step episode where the agent must act decisively. This suggests that moderate action budgets not only improve computational efficiency but also serve as a useful inductive bias that encourages focused, goal-directed behavior in embodied agents.

O Single-Step Action Generation vs. Receding Horizon Planning

A key design decision in our framework is single-step action generation: the agent generates and executes one action at a time, rather than planning action sequences through receding horizon control. We validate this choice through ablation experiments on the Cupboard Fitting benchmark.

Experimental Setup. We compare two variants of our full method: (1) *Single-Step (Ours)*: generates one action, executes it, observes outcome, performs test-time training, then generates the next action; (2) *Receding Horizon*: generates a complete action sequence (5-10 actions), executes only the first action, observes outcome, performs test-time training, then replans a new sequence from the updated state. Both variants use identical model architecture, training procedures, and test-time training mechanisms, differing only in planning granularity. The receding horizon approach requires approximately 5× more computation per step due to generating full sequences at each decision point.

Results. Table 18 shows that receding horizon planning achieves only 57.8% fit rate compared to 60.0% for single-step action generation—a comparable performance but at the cost of 4x more computation per step. This demonstrates that planning full action sequences at each decision point not only increases computational cost but actually harms performance in our framework.

Table 18: Single-step action generation vs. receding horizon planning on Cupboard Fitting. Despite using 5× more computation to plan sequences at each step, receding horizon shows degraded performance, demonstrating that single-step action generation is more effective for our reflective learning framework.

Method	Fit	Correct	Compute
Ours (Receding Horizon)	57.8%	25.8%	5.0×
Ours (Single-Step)	60.2%	25.3%	1.0×

Why Receding Horizon is Incompatible with Test-Time Training. The performance gap reveals fundamental incompatibilities between sequence planning and reflective test-time training:

(1) *Wasted computation on unpredictable futures*: Receding horizon generates 5-action sequences at each step but executes only the first action, discarding 80% of the computation. Critically, in our error-driven tasks, the outcomes of actions are inherently unpredictable before execution—whether an object fits in a compartment, whether placement will be stable, or whether grasping succeeds depends on precise physical interactions that cannot be reliably simulated. The model imagines 4 future actions based on assumed execution outcomes, but these assumptions are frequently wrong. When the first action fails or succeeds differently than predicted, the entire planned sequence becomes invalid. This wasted computation could instead generate more candidates for reflection-in-action (increasing N from 3 to 6) or perform additional test-time training epochs, both of which operate on actual execution outcomes rather than unreliable predictions.

(2) *Learning from imagination conflicts with learning from reality*: Test-time training fundamentally relies on learning from actual execution feedback—the model updates its understanding of spatial constraints only after physically attempting placements and observing real outcomes (does the object fit? does it block other spaces?). However, generating 5-action sequences forces the model to predict these outcomes before they occur: "If I place object A here, then I can place object B there, then object C..." These predictions are made with the model's current (pre-update) understanding, which is precisely what test-time training aims to improve. The model must simultaneously optimize two conflicting objectives: (a) accurately predicting hypothetical future states for sequence generation, and (b) updating its beliefs based on actual execution outcomes that contradict those predictions. This creates optimization interference where gradients from test-time training (learned from reality) fight against the inductive bias from sequence generation (learned from imagination).

Our single-action approach eliminates this conflict by committing only to decisions that can be made based on current observations, executing the action to obtain ground truth feedback, updating the model with real outcomes, then making the next decision with improved understanding. This aligns perfectly with the test-time training paradigm: learn from actual experience, not imagined futures.

Retrospective Reflection Provides Implicit Long-Horizon Planning. A potential concern is that single-step action generation lacks the foresight that explicit sequence planning provides. However, retrospective reflection addresses this through a different mechanism. While receding horizon achieves long-horizon reasoning by explicitly generating future action sequences, our approach achieves it through learned anticipation: retrospective reflection re-evaluates past actions with hindsight about their long-term consequences, creating training signals that teach the internal reflection model to anticipate multi-step outcomes before execution.

For example, when placing objects in a cupboard, an action that initially appears successful may be retrospectively downgraded when the agent discovers that this placement blocks the only space for a larger object. These retrospective scores train the internal reflection model to predict such long-horizon consequences at decision time—effectively distilling multi-step lookahead into single-step action evaluation. This learned implicit planning is more sample-efficient than explicit sequence generation: rather than exploring all possible future sequences at every step (most of which will be discarded), the agent learns which single actions lead to favorable long-term outcomes through accumulated experience.

Computational Efficiency Analysis. The 5× computational cost of receding horizon stems from generating full 5-action sequences at each decision step, only to execute the first action and discard the rest. This is particularly inefficient in our test-time training setting, where: (1) Sequence generation requires forward passes through the language model for all actions in the sequence; (2) The additional

compute does not improve learning quality—test-time training still operates on single executed actions, so planning discarded sequences provides no learning benefit; (3) The saved computation in single-step action generation can be reallocated to improvements that actually help: generating more candidates (N=6 vs. N=3), performing more test-time training epochs, or using larger working memory windows for retrospective reflection.

Our results demonstrate that effective long-horizon reasoning in embodied agents need not come from explicit sequence planning. Instead, combining single-step action generation with retrospective reflection achieves superior performance at 5× lower computational cost by learning to anticipate long-term consequences rather than exhaustively simulating future possibilities.

P Experiments on Long-Horizon Household Tasks: More Details

P.1 Task Generation & Validation Overview

Task Generation & Execution. We employ GPT-5 to generate task specifications through a carefully structured prompting procedure, adapting existing task templates that emphasize long-horizon reasoning and failure recovery from the original BEHAVIOR-1K benchmark. Each generated task instance includes: (1) a task description, (2) 3-7 relevant rooms, (3) new objects with placement specifications, and (4) a complete trajectory with interleaved thoughts, actions, and reflections scores. Since we include scene graphs in the prompts to GPT-5, which provides object properties such as bounding boxes, GPT-5 can deduce potential failures (e.g., size mismatches) during generation. We further execute the generated tasks in BEHAVIOR simulators to ensure consistency with the actual scene dynamics and provide ground-truth data for finetuning embodied LLMs. We initialize the environment by loading the BEHAVIOR-1K scenes, placing new objects at designated locations, and positioning the robot at a default starting pose. At each step, the agent executes the given actions. After every interaction, the system captures RGB-D observations, converted to point clouds which serve as the inputs to our Embodied LLMs. The simulator then performs physics-based verification and provides execution results that could be used for prompting external reflections. We also perform task verification checks by comparing the task execution results with GPT-5's generated task specifications. Please refer to Appendix P for details about data generation, models and experiments.

P.2 Why Build Upon BEHAVIOR?

BEHAVIOR-1K Li et al. [2024] provides an excellent foundation for embodied AI research with several key strengths: (1) **Photorealistic environments**: BEHAVIOR-1K features high-fidelity household scenes with realistic object models, physics simulation, and diverse room layouts across 1,000+ scenes; (2) **Rich object diversity**: The benchmark includes hundreds of object categories with varied sizes, shapes, and physical properties, enabling complex manipulation tasks; (3) **Standardized infrastructure**: BEHAVIOR-1K provides well-maintained simulation infrastructure, observation APIs, and action spaces that facilitate reproducible research.

However, BEHAVIOR-1K's original task design does not systematically stress two critical capabilities for reflective learning: (1) **Learning from failures**: Most BEHAVIOR-1K tasks are designed to be solvable with correct initial planning, without requiring agents to recover from or learn from execution failures. Tasks rarely include scenarios where early actions create downstream failures that only become apparent after multiple steps (e.g., placing a small object first that later blocks the only space for a larger object). (2) **Long-term dependencies**: The original benchmark emphasizes task completion but does not specifically design tasks around sequential dependencies where action order critically determines success, or where consequences of early actions remain hidden until much later in the episode.

To evaluate our reflective test-time training framework—which specifically learns from execution failures through retrospective reflection—we adapt BEHAVIOR-1K environments to create tasks that systematically incorporate these failure modes. We retain BEHAVIOR-1K's photorealistic scenes and simulation infrastructure while introducing task specifications that stress failure recovery, long-horizon credit assignment, and dependency reasoning. This allows us to leverage the strengths of BEHAVIOR-1K (realism, diversity, standardization) while evaluating capabilities (learning from failures, retrospective reasoning) that the original benchmark was not designed to measure.

P.3 Task Categories

We develop a systematic pipeline to generate Long-Horizon Household Tasks that stress error-driven adaptation in embodied agents. Our tasks are designed around four core failure modes common in real life: spatial reasoning errors, object selection mistakes, sequential dependency violations, and non-local planning failures.

Task Categories and Failure Modes. We define four task categories, each targeting specific failure patterns:

Fitting Tasks require agents to pack or place objects into constrained containers or surfaces. These tasks stress geometric reasoning and capacity constraints. Common failure modes include: (1) attempting to place oversized objects in small compartments, (2) blocking access to larger storage spaces with premature small-object placements, and (3) failing to recognize occlusion after placement. For example, placing a toy car in a box already containing a teddy bear may succeed physically but block future placements of larger items.

Selection Tasks require agents to compare and retrieve items based on preferences or constraints. Failure modes include: (1) selecting suboptimal items when better alternatives exist in unexplored rooms, (2) committing to choices before exploring all options, and (3) failing to revise decisions when new information becomes available. A typical scenario involves retrieving vegetables for a meal where lettuce (preferred) is in one room and tomato (less preferred) is in another—agents that explore insufficiently may select the inferior option.

Preparation Tasks involve sequential constraints and dependencies where actions must occur in specific orders. Common failures include: (1) attempting steps out of sequence (e.g., adding toppings before placing the base plate), (2) violating prerequisite conditions (e.g., trying to cook without retrieving ingredients first), and (3) missing intermediate setup steps. These tasks require agents to maintain action dependencies across multiple rooms and objects.

Hybrid Tasks combine multiple failure modes within a single episode, requiring agents to simultaneously reason about spatial constraints, object preferences, and sequential dependencies across long horizons.

The distribution of task categories can be found in Figure 7.

P.4 GPT-5 Task Generation Prompting Strategy

We employ a carefully structured prompting approach to generate high-quality, physically plausible long-horizon tasks. Our prompting strategy consists of three key components: (1) comprehensive scene context provision, (2) explicit failure mode specification, and (3) structured output formatting with reflection annotations.

Scene Context Provision. Each task generation prompt includes the complete scene graph from BEHAVIOR-1K, containing: room layouts with spatial relationships, existing objects in each room with 3D bounding boxes and affordance properties, navigable connections between rooms, and furniture placement. This rich context allows GPT-5 to reason about physical feasibility when proposing object placements and action sequences. For example, knowing that a table has dimensions $1.2\text{m} \times 0.8\text{m} \times 0.75\text{m}$ allows the model to avoid proposing placements of oversized objects.

Failure Mode Specification. We explicitly instruct GPT-5 to incorporate specific failure scenarios into the generated trajectories. For Fitting tasks, we request scenarios where early placements block later optimal choices. For Selection tasks, we specify that preferred items should be in rooms requiring more exploration. For Preparation tasks, we request action sequences with complex dependencies where naive sequential execution fails. For each task category, we provide 2-3 concrete examples of desired failure patterns in few-shot demonstrations.

Structured Output with Reflection Annotations. The prompt requires GPT-5 to generate not just action sequences, but complete trajectories with: (1) Each action wrapped in angle brackets (e.g., `<GO TO kitchen>`); (2) Internal reflections before each action with format `INTERNAL REFLECTION: [reasoning] | SCORE: [0-100]`; (3) Execution result annotations (`execution: success` or `execution: fail`); (4) External reflections after each executed action with format `PREVIOUS ACTION TO REFLECT ON: <action> | EXTERNAL REFLECTION: [assessment] | SCORE: [0-100]`; (5) Retrospective reflections at room

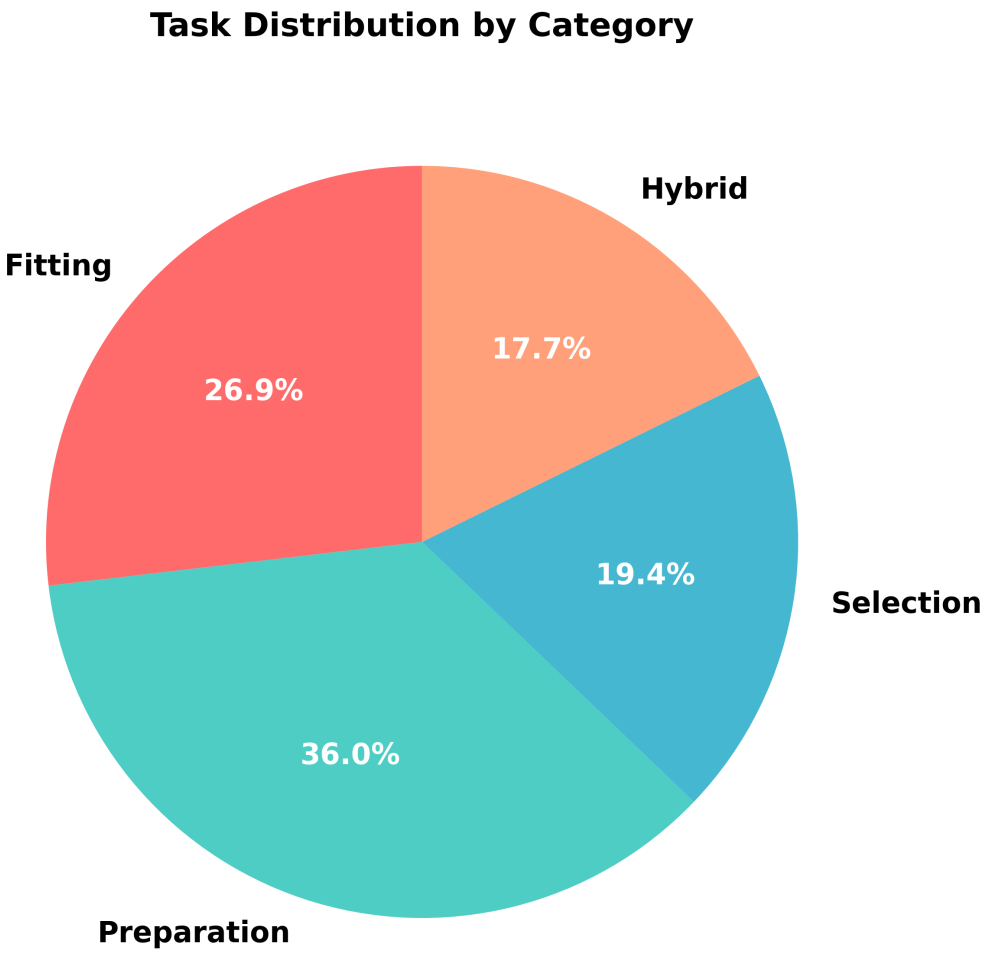

Figure 7: Distribution of task categories in the dataset.

exits with format `PREVIOUS ACTION TO REFLECT ON (retro): <action> | EXTERNAL REFLECTION: [hindsight assessment] | SCORE: [0-100]`.

Actions marked with `INTERNAL REFLECTION: ... | SCORE: [low score]` followed by the note "score low. don't execute" represent strategically poor choices that should be avoided through reflection-in-action. These actions may be physically feasible but lead to suboptimal task outcomes (e.g., picking up an inferior item when better alternatives exist). Yet they provide valuable data to train the internal reflection LLM (these actions are not used for training action LLM though).

Few-Shot Demonstrations. We provide 2-3 complete task examples for each task category, demonstrating the expected output format, reflection structure, and failure patterns. These examples show diverse scenarios: a Fitting task where placing small objects first blocks large object storage, a Selection task where exploring only nearby rooms leads to selecting inferior items, and a Preparation task where violating sequential dependencies causes failure.

Prompt Iteration and Refinement. We iteratively refined the prompt through multiple rounds of generation and validation. Early versions produced tasks with: (1) Physically implausible object placements (e.g., large furniture items on small shelves); (2) Insufficient failure diversity (most tasks had similar error patterns); (3) Inconsistent reflection scores (high scores for objectively poor actions). We addressed these through: (1) Adding explicit bounding box information and size reasoning requirements; (2) Providing diverse few-shot examples spanning different failure modes;

(3) Including calibration guidelines for score assignment (e.g., "score 0-30 for actions that fail or lead to dead ends; 70-100 for optimal strategic choices").

Reflection Score Calibration. To ensure consistent score semantics across generated tasks, we provide GPT-5 with explicit calibration guidelines: Scores 0-20 indicate actions that fail physically or lead to immediate dead ends; scores 21-40 indicate poor strategic choices that succeed physically but create future problems; scores 41-60 indicate suboptimal but acceptable actions; scores 61-80 indicate good strategic choices; scores 81-100 indicate optimal or near-optimal actions. This calibration ensures that scores are meaningful training signals for the reflection models rather than arbitrary numbers.

P.5 Physical Validation in BEHAVIOR.

Raw GPT-5 outputs may contain physically implausible scenarios or inconsistent spatial reasoning. We validate each generated task through execution in BEHAVIOR OmniGibson physics simulation:

Object Placement Validation: We verify all new objects can be physically placed at specified locations using sampling-based kinematics (`sample_kinematics`). Objects that fail placement (due to size mismatches, collision constraints, or stability issues) trigger task rejection. We use uniform scaling for all objects to avoid non-orthogonal transform errors.

Trajectory Execution Verification: We execute the ground-truth trajectory action-by-action, verifying that: (1) Navigation actions (`GO TO`) successfully place the robot in target rooms; (2) Manipulation actions (`PICK UP`, `PUT DOWN`) complete as annotated; (3) Expected failures actually fail (confirming physical constraints match annotations); (4) Action sequences respect object affordances and spatial constraints. Tasks where any expected-success action fails, or any expected-failure action succeeds, are rejected as inconsistent. This ensures our evaluation measures genuine agent learning rather than dataset annotation errors.

Observation Generation: During validation, we generate 3D point cloud observations for each room after every interaction step (navigation, pickup, placement). We capture observations from three external camera viewpoints positioned around the robot at fixed relative poses. For each viewpoint, we: (1) Capture RGB-D images at 560×560 resolution; (2) Convert depth to point clouds using camera intrinsics; (3) Transform point clouds from camera frame to robot base frame; (4) Store point clouds (stacked across viewpoints) as `.npy` files; (5) Save corresponding RGB images as `.png` files. Each room observation is stored in a unique directory named {`room_name`}_{`step_idx`}, allowing models to access the most recent observation per room at any point during inference.

P.6 Training Data Construction

From validated trajectories, we extract training data for three model components. All training examples include point cloud observations stored as file paths, which are loaded and processed by the 3D vision encoder during training.

Action Training Data. For each step t in a validated trajectory, we create training examples of the form:

$$\begin{aligned}\text{Input: } &\{\text{Task}, \text{All Rooms}, \text{Explored Rooms},\\ &\text{Current Room}, \text{Observations},\\ &\text{Previous Action}, \text{Previous External Reflection}\}\\ \text{Output: } &\text{Action}_t\end{aligned}$$

Observations consist of point cloud paths for each explored room (most recent per room).

Internal Reflection Training Data. For each action (including non-executed low-score actions), we create examples:

$$\begin{aligned}\text{Input: } &\{\text{Task}, \text{All Rooms}, \text{Explored Rooms},\\ &\text{Current Room}, \text{Observations},\\ &\text{Previous Action}, \text{Previous External Reflection}\}\\ &\text{Potential Action}_t\}\\ \text{Output: } &\{\text{Internal Reflection}_t, \text{Score}_t\}\end{aligned}$$

This includes actions marked "don't execute"—the model must learn to score these actions low during internal reflection to prevent execution.

External Reflection Training Data. After each executed action, we create examples:

$$\begin{aligned}\text{Input: } \{&\text{Task, All Rooms, Explored Rooms,}\\ &\text{Current Room, Observations (before and after),}\\ &\text{Previous Action, Previous External Reflection}\}\\ &\text{Executed Action}_t, \text{Execution Result}_t\}\\ \text{Output: } \{&\text{External Reflection}_t, \text{Score}_t\}\end{aligned}$$

Execution results indicate `success` or `fail`, and observations include both pre-action and post-action point clouds to enable change detection.

Retrospective Reflection Training Data. For retrospective reflection, we identify room transitions (marked by `EXIT` actions) and collect all actions taken in that room. For each historical action a_j, we create:

$$\begin{aligned}\text{Input: } \{&\text{Task, Context, Current Observations,}\\ &\text{Action}_j, \text{Room Action History,}\\ &\text{Last Reflection}_j\}\\ \text{Output: } \{&\text{Retro Reflection}_j, \text{Updated Score}_j\}\end{aligned}$$

The prompt includes all actions and their external reflections from the current room window, allowing the model to re-evaluate a_j with hindsight about downstream consequences.

P.7 Model Architecture and Training

Base Architecture. We build our 3D vision-language-action model on LLaVA-3D Zhu et al. [2025], which processes point clouds through a 3D encoder and fuses them with language instructions via a multimodal projector. The architecture consists of: (1) A 3D point cloud encoder that extracts spatial features; (2) A vision-language projector that aligns 3D features with language embeddings; (3) A Llama-based language model backbone (7B parameters) for reasoning and generation.

Unified Multi-task Pretraining. Rather than training three separate models, we first train a single unified model on all three reflection modes (action generation, internal reflection, external reflection) using task-specific prompts to distinguish modes. This enables cross-mode knowledge transfer: the model learns shared representations for spatial reasoning and object affordances that benefit all three capabilities. We fine-tune LLaVA-3D-7B with learning rate 2×10^{-5}, batch size 8 per GPU, gradient accumulation over 4 steps, for 3 epochs using AdamW optimizer with weight decay 0.01. Following prior work Hu et al. [2025], we maintain fused point cloud observations from previous steps to provide the model with spatial memory across the episode.

Test-Time Model Instantiation. At deployment, we instantiate three separate copies of the finetuned unified model: π_θ (action generation model), V_{ϕ_i} (internal reflection model, updated via test-time training), and V_{ϕ_e} (external reflection model, frozen). This architecture allows selective adaptation: only V_{ϕ_i} and π_θ update their parameters during test-time deployment, while V_{ϕ_e} remains fixed to provide stable assessment signals.

P.8 Test-Time Training Configuration

LoRA Configuration. For memory-efficient test-time training, we apply LoRA with rank $r = 4$, alpha $\alpha = 8$, dropout rate 0.15, targeting `q_proj` and `v_proj` modules.

Internal Reflection Model Training. We train V_{ϕ_i} via supervised learning to predict retrospective reflections with learning rate 5×10^{-5}, 3 epochs, gradient clipping at 0.3, and negative log-likelihood loss over reflection text.

Action Model Training via REINFORCE. We train π_θ via policy gradient using retrospective scores as rewards: SGD optimizer with learning rate 1×10^{-3}, 3 epochs, gradient clipping at 0.5, and reward transformation $r = 2(s_r/100) - 1$ mapping [0,100] scores to [-1,1] rewards.

Regularization Strategy. To prevent catastrophic forgetting during test-time training, we include regularization examples in $\mathcal{D}_{\text{train}}$: we sample unexplored actions from the action space and use the current internal reflection model's outputs as targets, with a 50% retrospective examples and 50% regularization examples mixing ratio. This anchors the model to its pretrained knowledge for regions of the state space not covered by recent experience.

P.9 Evaluation Protocol and Baselines

Train-Test Split. We ensure zero overlap between finetuning and evaluation: (1) No shared task descriptions—evaluation tasks have completely different natural language instructions; (2) No shared scenes—different scene instances from BEHAVIOR-1K; (3) No shared object placements—all object configurations are unique; (4) No shared trajectories—action sequences and reflection patterns differ. This tests generalization to novel tasks rather than memorization.

Success Criteria. A task succeeds if and only if: (1) All required objects reach target locations; (2) All spatial constraints are satisfied (e.g., inside, on top of); (3) All preference constraints are met (e.g., selecting preferred items); (4) All sequential dependencies are respected; (5) Task completion occurs within the action budget (30 steps).

Deployment Procedure. For each test task, we: (1) Initialize agent in the first room with task description; (2) Execute action generation + internal reflection + external reflection loop; (3) Trigger retrospective reflection at room exits or after $K = 5$ steps; (4) Perform test-time training when retrospective reflections accumulate; (5) Continue until task completion or action budget exhaustion. The agent receives no human feedback during deployment—all learning signals come from self-generated reflections.

Baseline Implementations. We implement the following baselines: *Reflexion* Shinn et al. [2023] maintains a text buffer of past reflections and includes them in prompts for future actions, generating verbal critiques after each step without parameter updates. *Self-Refine* Madaan et al. [2023] iteratively improves actions through self-critique and revision cycles, allowing up to 3 refinement iterations per action before execution. We revise the above two baselines to incorporate multimodal inputs. *ReflectVLM* adapts the reflection mechanism from Feng et al. [2025] using learned value functions for action scoring, with a separate value head trained on our training data. *3DLLM-Mem* Hu et al. [2025] maintains fused point cloud observations from all previous steps and previous-step execution results as context, providing spatial memory without explicit reflection or test-time training. *PPO* Schulman et al. [2017b] and *DreamerV3* Zhou et al. [2024] are trained as reinforcement learning baselines: PPO uses on-policy policy gradient with clipped surrogate objective and GAE for advantage estimation, while DreamerV3 learns a world model from observations and trains a policy in the learned latent space. Both RL baselines are trained for the same total number of environment interactions as our supervised fine-tuning phase to ensure fair compute comparison. All baselines use the same LLaVA-3D-7B backbone with identical finetuning procedures where applicable for fair comparison.

Q Cupboard Fitting: More Details

Q.1 Cupboard Setup

The task environment consists of three key components. First, a multi-compartment cupboard structure with 6-8 compartments of varying sizes and colors, each defined by precise 3D bounding boxes. Second, a set of 6-10 colored geometric objects that must be placed into compartments. Third, a Franka Panda robotic arm controlled through high-level natural language commands such as "pick up the red apple" or "put the blue object in the green compartment." The agent's objective is to place all objects into the cupboard such that each fits completely within compartment boundaries, while multiple objects may share compartments when space permits. Each task has only one valid solution where all objects fit within their designated compartments. Success requires reasoning about object-compartment compatibility, multi-object spatial packing, and long-horizon dependencies where early placement decisions affect later possibilities. After each action, the environment provides execution status through forward simulation, and updated visual observations. We define two evaluation metrics: correct rate measures the percentage of objects placed in their correct target compartments, while fit rate measures the percentage of objects successfully placed in any compartment.

Q.2 Base Model and Supervised Fine-tuning.

We build upon Qwen2.5-VL-3B as our base vision-language model for the Cupboard Fitting benchmark. We employ unified multi-task supervised fine-tuning on action generation, internal reflection, and external reflection tasks using: global batch size 128, learning rate 1×10^{-5}, weight decay 0.1, trained for 3 epochs using AdamW optimizer. Task-specific prompts distinguish between the three reflection modes, enabling cross-task knowledge transfer.

Q.3 Test-Time Training Variants.

We implement two test-time training variants to study the trade-off between adaptation capacity and computational efficiency:

Base-Weight Test-Time Training for Reflection-on-Action: This variant updates all non-visual parameters during deployment. We freeze visual encoder parameters (identified by `visual` or `vision` in parameter names) but update all language model parameters. For the action model trained via REINFORCE, we use SGD optimizer with learning rate 1×10^{-3}, weight decay 1×10^{-4}, zero momentum, training for 3 epochs. For the internal reflection model trained via supervised learning, we use SGD optimizer with learning rate 5×10^{-5}, weight decay 1×10^{-4}, zero momentum, training for 3 epochs. This approach provides maximum adaptation capacity but requires updating millions of parameters.

LoRA Test-Time Training for Reflection-on-Action: For memory-efficient adaptation, we apply Low-Rank Adaptation with rank $r = 8$, alpha $\alpha = 16$, dropout rate 0.1, targeting all linear layers except `lm_head`, `embed_tokens`, and visual encoder components. The LoRA adapters are applied to both internal reflection and action models using the PEFT library. For the action model, we use SGD with learning rate 1×10^{-2}, weight decay 1×10^{-4}, zero momentum, trained for 3 epochs. For the internal reflection model, we use SGD with learning rate 0.2, weight decay 1×10^{-4}, zero momentum, trained for 3 epochs. The higher learning rates compensate for the reduced parameter count—LoRA updates only the low-rank adapter parameters, reducing trainable parameters by over 95% while maintaining comparable performance.

R More Qualitative Examples

In Figure 8, we show additional qualitative examples demonstrating how the model improves over the course of a deployment episode through reflection mechanisms.

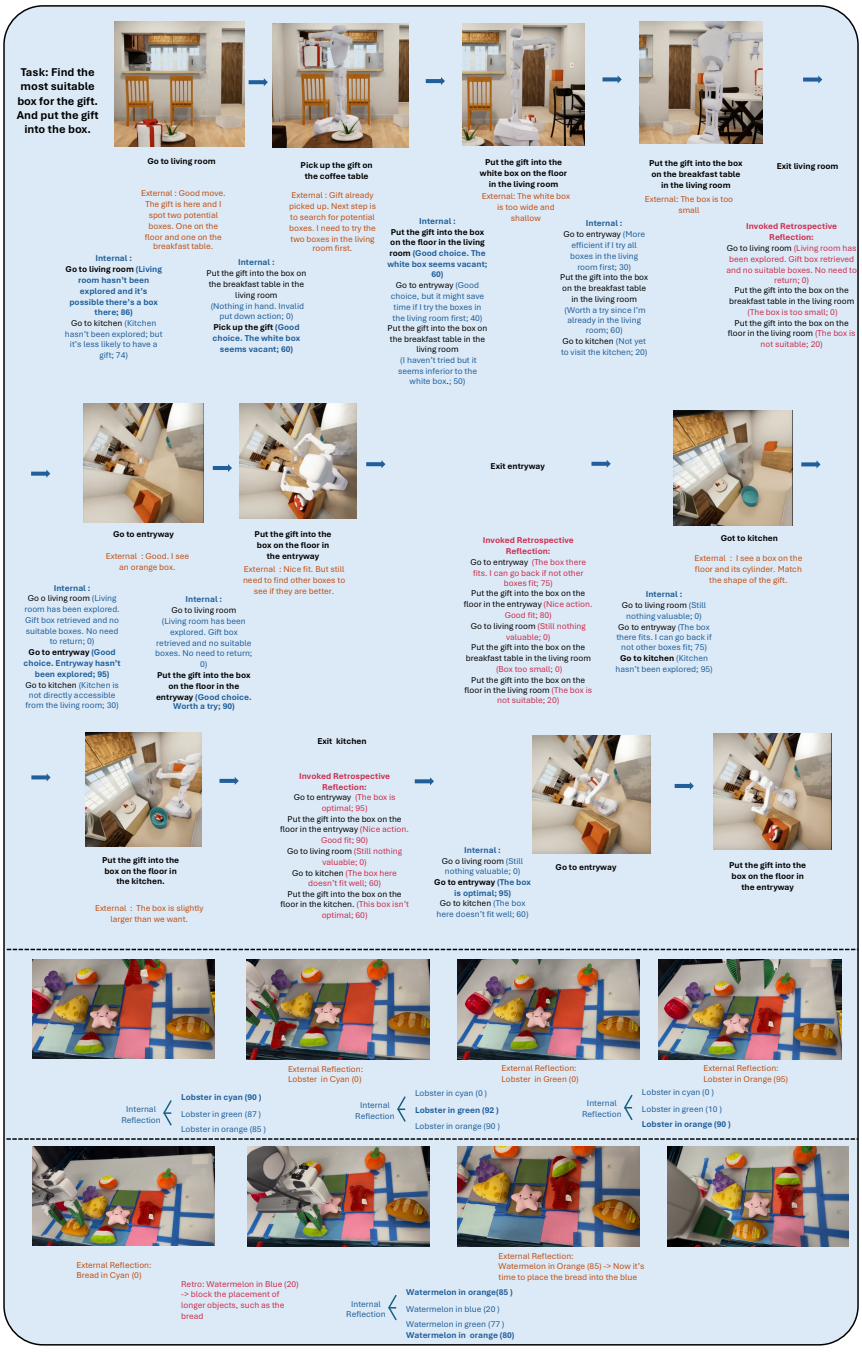

Figure 8: Additional qualitative examples. Blue text: internal reflection for candidate selection. Orange text: external reflection after execution. Red text: retrospective reflection and model updates. Reflection scores shown in brackets. Steps and reflections simplified for presentation.