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

Anonymous CVPR submission

Paper ID *****

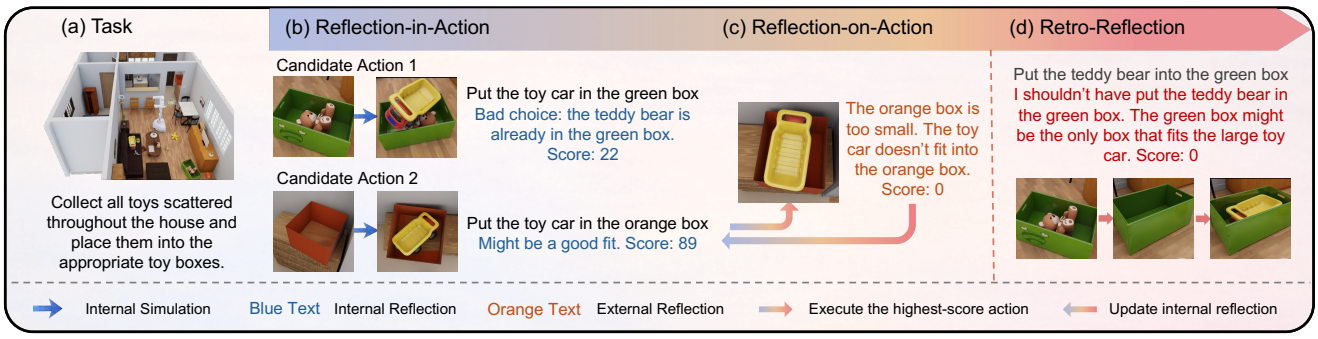

Figure 1. Conceptual overview of Reflective Test-Time Planning. The agent (a) receives a long-horizon task, (b) performs *reflection-in-action* by internally simulating and scoring candidate actions, (c) performs *reflection-on-action* by updating its beliefs and decision systems based on execution outcomes, 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 ablative studies validating the complementary roles of reflection-in-action and reflection-on-action. Qualitative analyses, including real-robot trials, highlight behavioral correction through reflection.

"Error isn't simple darkness, it sheds a light of its own." — Kathryn Schulz, *Being Wrong*

1. Introduction

Embodied LLMs [10, 15, 41] 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 [28], 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, however, capture at best a superficial version of one mode while neglecting the other. One line of work [21, 30] 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, reflections are stored only as contextual text and do not update the underlying decision process, making their effects transient and prone to repetition under distributional shift. A second line of work [5, 37] relies on internal world models to guide action selection in embodied environments. These approaches support reflection-in-action through anticipated outcomes, but typically assume fixed, pretrained dynamics models that may be wrong in ways only revealed during execution.

To operationalize both reflection modes in embodied settings, we introduce Reflective Test-Time Planning, a framework that seamlessly 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 the agent's beliefs in reality, but remains inherently local—it only evaluates consequences visible at the next timestep. Yet 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 [1], 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. 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. Qualitative analyses, including preliminary real-robot trials, show that reflection reduces repetitive failure modes in practice.

2. Related Works

Test-Time Adaptation (TTA) and Learning enables models to adjust to distribution shifts during inference without source data [20, 31, 33]. Early methods minimize prediction entropy, with Tent [33] updating batch-norm parameters online and later work adding calibrated objectives [26, 34]. Parameter-efficient tuning further boosts TTA: LoRA [11] enables low-rank weight updates with little memory [16], while bias-only tuning [3] offers alternative efficiency–accuracy trade-offs. Recent extensions operate on hidden state representations [32], supporting long-context memory. For embodied settings, continual learning frameworks [17, 23] demonstrate viability in manipulation and navigation [8, 22]. We adapt models at test time via self-supervised reflective signals extracted from the agent's own verbal assessments.

Multimodal Embodied Large Language Models couple visual perception with language understanding to enable embodied planning. Recent foundation models leverage large-scale robotic datasets for zero-shot generalization [2, 15, 41], with RT-2 transferring web knowledge and OpenVLA offering open-source support across heterogeneous embodiments. 3D spatial understanding has become central, spanning point clouds [9], 3D patches [40], and lightweight point cloud injection [19]. Further extensions incorporate multisensory interaction [10], generative world models for manipulation [37], and long-term spatial–temporal embodied memory [12]. Architectural advances explore interleaved multimodal instructions [4] and chain-of-thought reasoning [24, 36] for improved task decomposition. In contrast, we view deployment as a learning phase: instead of acting as a fixed policy, our embodied multimodal LLM reflects on its own actions and updates itself via test-time training.

Reflection and Self-Improvement in AI Agents. Reflection mechanisms enable agents to learn from failures through self-critique and refinement. Verbal self-reflection methods (e.g., Reflexion [30]) store natural-language critiques to guide future actions, with extensions exploring self-refinement [21, 35], tool-assisted verification [6], curiosity-driven reflection [14], multi-agent systems [25], and robotics [13]. These approaches support reflection-on-action but store reflections only as text, influencing behavior indirectly without updating model parameters, and thus remain brittle under distribution shift. A complementary line enables reflection-in-action via internal predictive models that anticipate outcomes [5, 7, 37, 38]. Such world models are typically fixed despite evolving dynamics in embodied settings. Our method unifies reflection-in-action

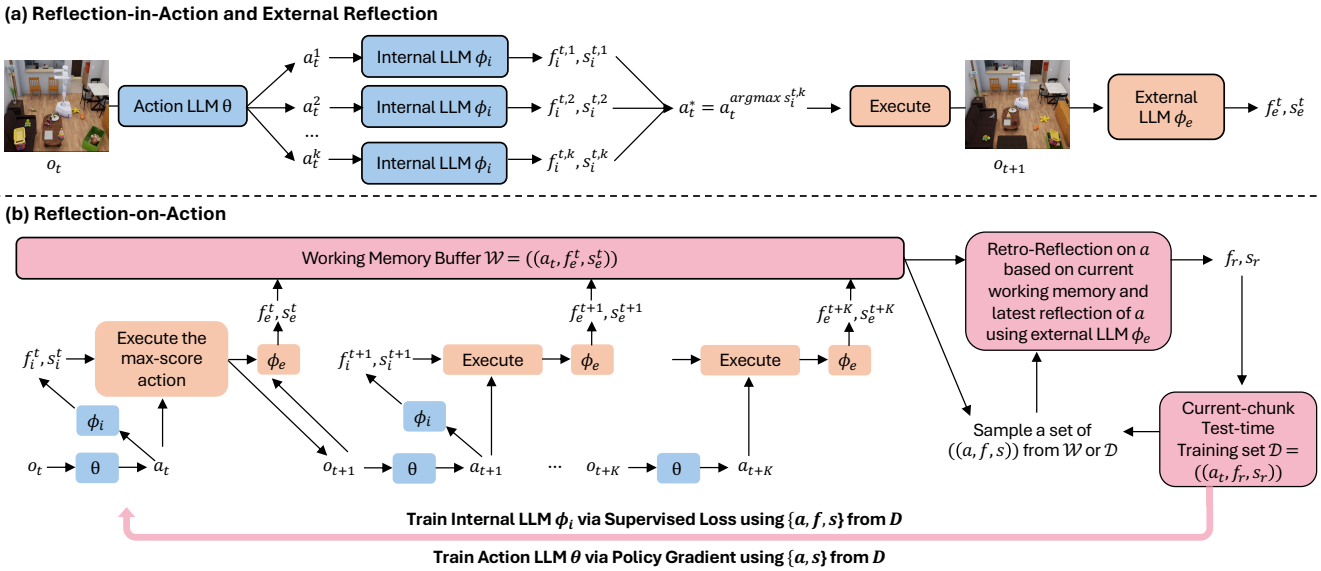

Figure 2. **Method overview.** (a) *Reflection-in-action*: multiple candidate actions are generated and scored by an internal reflection LLM prior to execution. (b) *Reflection-on-action*: iteratively invoked when working memory hits K or at key milestones. Executed actions are critiqued by an external 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.

and reflection-on-action by converting reflections into self-supervised training signals for parameter updates during deployment.

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 multimodal 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 this static inference setting 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 multimodal LLMs are first initialized with basic capabilities for reasoning in specific 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 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 an overview of our method. We also provide a detailed method breakdown in Algorithm 1.

We choose verbal reflection as the representation for all reflection types inspired by "double-loop learning" [1]: by articulating what went wrong and why, the agent abstracts generalizable lessons transferable to future decisions rather than merely 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 several 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 candidate, which guide action selection.

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, $f_e^{(0)}$ = None; working memory buffer, test-time training set, retro-buffer $\mathcal{W}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{retro}} \leftarrow \emptyset$; Temperature T; Max step $\mathcal{T}$;
for $t = 1, 2, ..., \mathcal{T}$ **do**
  // Reflection-in-Action
  Construct action prompt x_{action} from $\tau, o_t, a_{t-1}, f_e^{t-1}$
  Sample N actions: $\{a_t^k\}_{k=1}^N \sim \pi_\theta(\cdot|x_{\text{action}}; T)$
  for $k = 1, ..., N$ **do**
    Construct internal reflection prompt x_{internal}^k
    Generate 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^* in environment $\rightarrow$ observe (o_{t+1}, e_t)
  Break if task complete
  // Reflection-on-Action
  Construct external reflection prompt x_{external}
  Generate reflection 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 retro prompt x_{retro}^j from $\tau, a_j, \mathcal{W}, o_{t+1}, f^j, s^j$
      Generate revised reflection: $f_r^j, s_r^j \leftarrow V_{\phi_e}(x_{\text{retro}}^j)$
      $\mathcal{D}_{\text{retro}}[a_j] \leftarrow (a_j, f_r^j, s_r^j)$ // reflection overwrite for a_j
    end for
    $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{retro}}$
    // Regularization: Prevent Catastrophic Forgetting
    for sampled unexplored action a_l **do**
      Construct pre-action internal x_{internal}^l
      $f_i^l, s_i^l \leftarrow V_{\phi_i}(x_{\text{internal}}^l)$ // Original output
      $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{(a_l, f_i^l, s_i^l)\}$
    end for
    // Test-Time Training: Supervised Learning for Internal LLM
    for epoch $e = 1, ..., E$ **do**
      for $(a, f, s) \in \mathcal{D}_{\text{train}}$ **do**
        Construct x_{internal} from a
        Compute loss: $\ell_\phi = -\log p_{\phi_i}(f|x_{\text{internal}})$
        Update: $\phi_i \leftarrow \phi_i - \eta_\phi \nabla_{\phi_i} \ell_\phi$
      end for
    end for
    // Test-Time Training: REINFORCE for Action LLM
    for RL step $= 1, ...,$ RL_Steps **do**
      for $(a, f, s_r) \in \mathcal{D}_{\text{train}}$ **do**
        Construct x_{action} from a. Reward $r = 2s_r/100 - 1$
        Compute log probability: $\log p_\theta(a|x_{\text{action}})$
        Compute REINFORCE loss: $\ell_\theta = -r \cdot \log p_\theta(a|x_{\text{action}})$
      end for
      Accumulate gradients and update: $\theta \leftarrow \theta - \eta_\theta \nabla_\theta \ell_\theta$
    end for
    Clear working memory: $\mathcal{W} \leftarrow \emptyset$
  end if
end for

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.*, [12]). 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:

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

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. The internal reflection LLM 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 only 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:

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

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 [18] environments. Drawing inspiration from household scenarios, we define four core 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.

	Baselines						Ablations of Reflective Test-Time Planning (Ours)					
	Reflexion	Self-Refine	ReflectVLM	PPO	DreamerV3	3DLLM-Mem	w/o RIA w/o ROA	w/o RIA	w/o ROA	w/o Act. Loss	w/o Int. Loss	**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" is short for Reflection-in-action, "ROA" for reflection-on-action. Act. Loss means action model loss; Int. Loss means internal reflection model loss.

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 tasks generated by GPT-5 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 (*e.g.*, assumed task failures). Please refer to Appendix E for more details about data generation, models and experiments.

Finetuning & Evaluation Sets. Each GPT-generated and executed trajectory yields (observation, action, reflection, score) tuples extracted at corresponding timesteps, which form supervised fine-tuning pairs to initialize our three embodied LLMs with basic task competency before test-time deployment. During evaluation, the agent is given only the scene configuration and task description, with all other components initialized as None, and must autonomously generate and execute its trajectory step by step. To ensure rigorous evaluation, finetuning and evaluation sets have no overlap in task descriptions, scene configurations, or object placements. We evaluate using task success rate: the percentage of tasks where the agent completes the objective within the action budget. A task succeeds if all required objects reach target locations and constraints are satisfied.

4.2. Model Architecture & Implementation Details

We build a 3D-LLM based on LLaVA-3D [40]. During supervised fine-tuning, we train a single unified model on all three modes (action generation, internal reflection, external reflection) for cross-mode learning. We fine-tune LLaVA-3D-7B with learning rate 2×10^{-5}. At test-time, we instantiate three copies of this model as π_θ, V_{ϕ_i}, and V_{ϕ_e}, where π_θ and V_{ϕ_i} are updated via test-time training. Following Hu et al. [12], we maintain fused observations from previous-step point clouds. During test-time training, we apply LoRA (rank 4, alpha 8, dropout 0.15) targeting `q_proj` and `v_proj` layers. The internal LLM uses supervised learning (learning rate 5×10^{-3}, $E = 5$ epochs); the action LLM uses REINFORCE (learning rate 1×10^{-3}, 3 RL_Steps). We use gradient clipping (0.3 for supervised, 0.5 for RL) to prevent catastrophic forgetting. For reflection-in-action, we set $N = 4$ candidates and temperature $T = 2.0$. We adopt four categories of baselines: (1) verbal-reflection [21, 30]; (2) world-model based reflection [5]; (3) RL-based baselines: Vanilla PPO [29] and world-model based RL (DreamerV3 [7]) (4) 3D-LLM with interaction context [12].

4.3. Experimental Results & Analysis

Table 1 shows that our model achieves substantial improvements over both prior methods and ablated versions across all task categories. The improvement is most prominent on Fitting tasks, where our approach reaches 44.7% success rate compared to 10.6% for the strongest baseline (3DLLM-Mem), 2.1% for ReflectVLM and 0% for PPO. Fitting tasks benefit most from our method because they impose tight spatial constraints that require iterative refinement through trial and error—agents must reason about 3D geometry from multiple attempts and continuously adjust their understanding based on execution feedback. The ablation studies reveal that reflection-in-action (RIA) and reflection-on-action (ROA) are mutually dependent. Removing either causes performance degradation. Sometimes removing just one component performs worse than removing both. For instance, without RIA, Preparation falls to 3.17% and Hybrid to 3.23%, while removing both components yields 11.1% and 12.9% respectively on these tasks. This is because RIA without ROA produces overconfident yet inaccurate action scores with no hindsight correction mechanism, while ROA without RIA wastes learning on poorly cho-

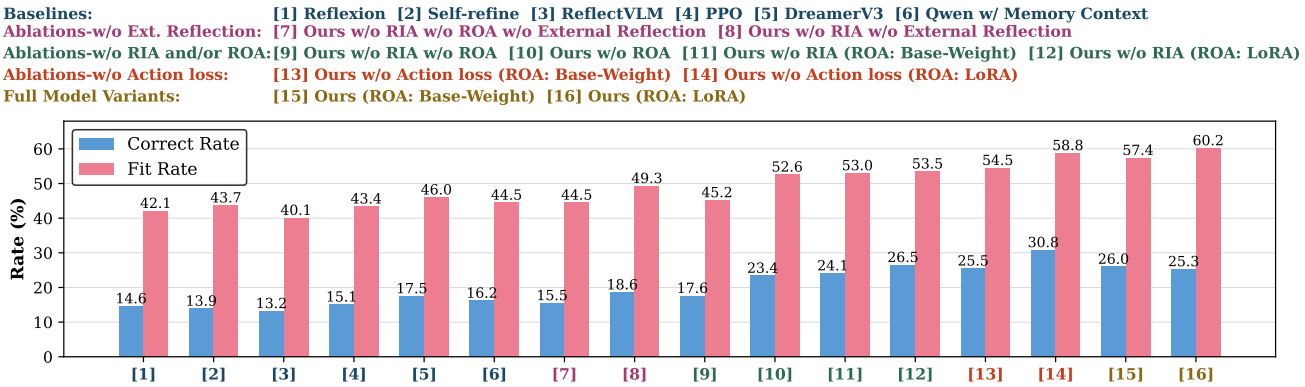

Figure 3. **Cupboard Fitting results**. Blue bars show correct-placement rate, pink bars show fit rate. RIA means Reflection-in-action; ROA means Reflection-on-action. "W/o external reflection" means that we don't use external reflection as the input to the action generation LLM. We implement two test-time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

sen actions that fail to reveal true scene affordances. When combined, they form a virtuous cycle: best-of-N selection yields higher-quality trajectories for learning, and test-time training refines the internal model, leading to better future action selection. The loss ablations similarly demonstrate mutual dependency between the action loss and internal reflection loss, which together constitute ROA. Removing either loss causes performance degradation, and sometimes removing just one loss performs worse than removing both (w/o ROA). Without the internal loss, Hybrid drops to 9.68% compared to 12.9% for w/o ROA, and Selection falls to 8.82% compared to 11.8% for w/o ROA. This demonstrates that policy-gradient updates and supervised reflection both contribute essential signals for effective adaptation. The qualitative example in Figure 5(a) demonstrates that our model supports continual learning and active perception driven by accumulated scene experience. The test-time cost analysis and compute-matched experiment in Appendix A shows that even with substantially more test-time compute, the baselines continue to repeat failures and their performance even degrades. Appendix B also shows that the reflection mechanisms enable the agents to generalize to novel embodied environments like HM3D [27].

5. Experiments on 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 placement failures, enabling precise measurement of reflective test-time training mechanisms.

The task environment consists of three key components.

Figure 4. Examples of the Cupboard Fitting Task.

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.

Model Architecture & Implementation Details. We build upon Qwen2.5-VL-3B with unified supervised fine-tuning (batch size 128, learning rate 1×10^{-5}, weight decay 0.1) on action generation, internal reflection, and external reflection. For test-time training, we implement two variants: (1) *Base-Weight*: updates all non-visual parameters with SGD (learning rate 1×10^{-3} for action model, 5×10^{-5}

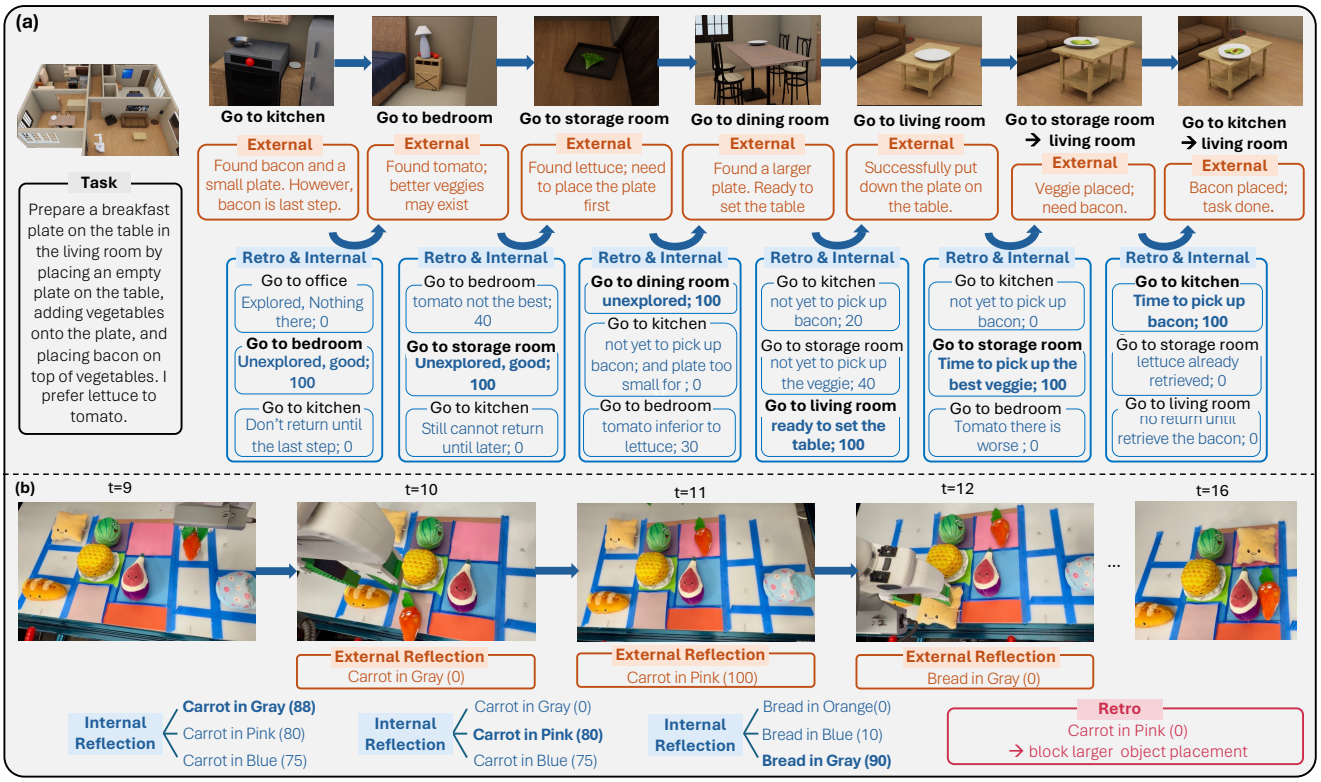

Figure 5. **Qualitative Examples.** Steps and reflections are simplified for better presentations. Blue text shows internal reflection used for candidate selection, orange text shows external reflection after execution, and red text suggests retrospective reflection. (a) Long-Horizon Household example. 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.

for internal model); (2) *LoRA*: applies LoRA (rank 8, alpha 16, dropout 0.1) targeting all linear layers except lm_head, embed_tokens, and visual encoder, using SGD with learning rate 0.01 for action model and 0.2 for internal model. Both train for 3 epochs. We limit episodes to 50 maximum steps and update the model whenever an execution failure occur. Hyperparameter selection ablations can be found in Appendix C. We use the same baselines as the Long-Horizon Household Task except that we replace 3D-LLM with Qwen for the baseline model with context memory.

Experimental Results & Analysis. Figure 3 shows that our full method with both reflection-in-action (RIA) and reflection-on-action (ROA) using LoRA-based test-time training achieves 60.2% fit rate and 25.3% correct rate, substantially outperforming LLM-based reflection baselines, RL-based baselines and memory context baselines. Ablation studies reveal that both reflection mechanisms are essential: removing RIA drops performance to 53.5% fit rate, while removing ROA reduces it to 45.2% fit rate. Removing both mechanisms along with external reflection further degrades performance to 44.5% fit rate. We also find that LoRA-based test-time training (60.2% fit rate) performs comparably to full base-weight training (57.4% fit rate) while being more parameter-efficient. These results confirm that our unified framework of reflection-in-action for candidate selection and reflection-on-action for test-time adaptation enables effective learning from failures during deployment. In Figure 5(b), we show a qualitative result where we generalize our model in the real-robot setting. In these trials, reflective adaptation enables the robot to recover from execution failures, avoid repeated placement errors, and correct earlier decisions through retrospective reflection, illustrating that the learned behaviors transfer to the physical world with satisfying generalization abilities. Appendix D also shows that our method enables long-horizon planning by retrospective reflection that outperforms Receding Horizon Planning while saving 5x test-time compute.

6. Conclusion

This paper introduces 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 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).

References

[1] Chris Argyris. Double loop learning in organizations. *Harvard Business Review*, 1977. 2, 3

[2] 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. 2

[3] Sri Harsha Dumpala, Chandramouli Sastry, and Sageev Oore. Test-time training for speech, 2023. 2

[4] 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. 2

[5] 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. 2, 6, 9

[6] 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. 2

[7] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. 2, 6

[8] Elvin Hajizada, Balachandran Swaminathan, and Yulia Sandamirskaya. Continual learning for autonomous robots: A prototype-based approach. 2024. 2

[9] 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. 2

[10] 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. 1, 2

[11] 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. 2

[12] 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. 2, 4, 6, 9

[13] 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. 2

[14] Isaac Kauvar, Chris Doyle, Linqi Zhou, and Nick Haber. Curious replay for model-based adaptation. *International Conference on Machine Learning*, 2024. 2

[15] 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. 1, 2

[16] Yuto Kojima, Jiarui Xu, Xueyan Zou, and Xiaolong Wang. Lora-ttt: Low-rank test-time training for vision-language models. 2025. 2

[17] 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. 2

[18] 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. 5, 6

[19] Chengmeng Li, Junjie Wen, Yan Peng, Yaxin Peng, Feifei Feng, and Yichen Zhu. Pointvla: Injecting the 3d world into vision-language-action models. 2025. 2

[20] 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. 2

[21] 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. 1, 2, 6, 9

[22] 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. 2

[23] 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. 2

[24] 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. 2

[25] Andrew Ng. Agentic design patterns part 2: Reflection. *DeepLearning.AI The Batch*, 2024. 2

[26] 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. 2

[27] 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. 7, 1

[28] Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books, New York, NY, USA, 1992. 1

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. 6, 9

[30] 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. 1, 2, 6, 9

[31] 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. 2

[32] 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. 2

[33] 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. 2

[34] 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*, page 3736–3746, New York, NY, USA, 2024. Association for Computing Machinery. 2

[35] Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. Self-contrast: Better reflection through inconsistent solving perspectives. 2024. 2

[36] 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. 2

[37] 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

[38] H. Zhen et al. Learning 4d embodied world models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2025. 2

[39] Siyuan Zhou, Yilun Du, Jiaben Chen, Yandong Li, Dit-Yan Yeung, and Chuang Gan. Robodreamer: Learning compositional world models for robot imagination, 2024. 9

[40] 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. 2, 6, 8

[41] 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. 1, 2

A. Test-Time Cost Analysis

A.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.

A.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.

A.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 2, 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 2. 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.

B. Generalization to Habitat-Matterport 3D Scenes

B.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 [27], 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

	Baselines						Ablations of Reflective Test-Time Planning (Ours)					
	Reflexion	Self-Refine	ReflectVLM	PPO	DreamerV3	3DLLM-Mem	w/o RIA w/o ROA	w/o ROA	w/o RIA	w/o Act. Loss	w/o Int. Loss	**Ours**
Preparation	2.44%	4.88%	0%	0%	2.44%	7.32%	0%	7.32%	2.44%	14.6%	9.76%	**19.5%**

Table 3. Generalization results on Habitat-Matterport 3D (HM3D) environment for Preparation tasks (41 test cases). The significant performance drop compared to BEHAVIOR Long-Horizon Household Tasks (31.7% → 14.6%) demonstrates the substantial domain gap. However, our reflective framework maintains relative advantages over baselines, with several methods (ReflectVLM, PPO, w/o RIA w/o ROA, w/o RIA) achieving 0% success rate. "RIA" is short for Reflection-in-action, "ROA" is short for reflection-on-action. Act. Loss means action model loss. Int. Loss means internal reflection model loss.

active exploration to resolve, and (4) longer navigation distances between task-relevant objects.

B.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.

B.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.

B.4. Results and Analysis

Table 3 presents the generalization results on HM3D Preparation tasks. Our method demonstrates robust generalization to photorealistic real-world environments, achieving 19.5% success rate despite being trained exclusively on synthetic BEHAVIOR-1K scenes. While this represents a 12.2 percentage point performance drop compared to Long-Horizon Household Tasks (31.7% → 19.5%), our reflective framework maintains over 60% of its original performance and substantially outperforms all baselines, demonstrating the effectiveness of reflection-based adaptation under domain shift. The strongest baseline, 3DLLM-Mem, achieves only 7.32%, while several methods completely collapse to 0% (ReflectVLM, PPO, w/o RIA w/o ROA). This stark contrast highlights that our reflection mechanisms—particularly the combination of reflection-in-action and reflection-on-action—provide critical robustness when facing distribution shift from synthetic to photorealistic environments. The fact that our method maintains substantial relative advantages over baselines suggests that the core reflective mechanisms—generating diverse candidates, scoring them through internal simulation, learning from execution feedback, and updating both action and reflection models—transfer effectively across domain boundaries and provide a robust foundation for embodied agents generalizing to novel environments.

C. 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.

C.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).

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 ac-

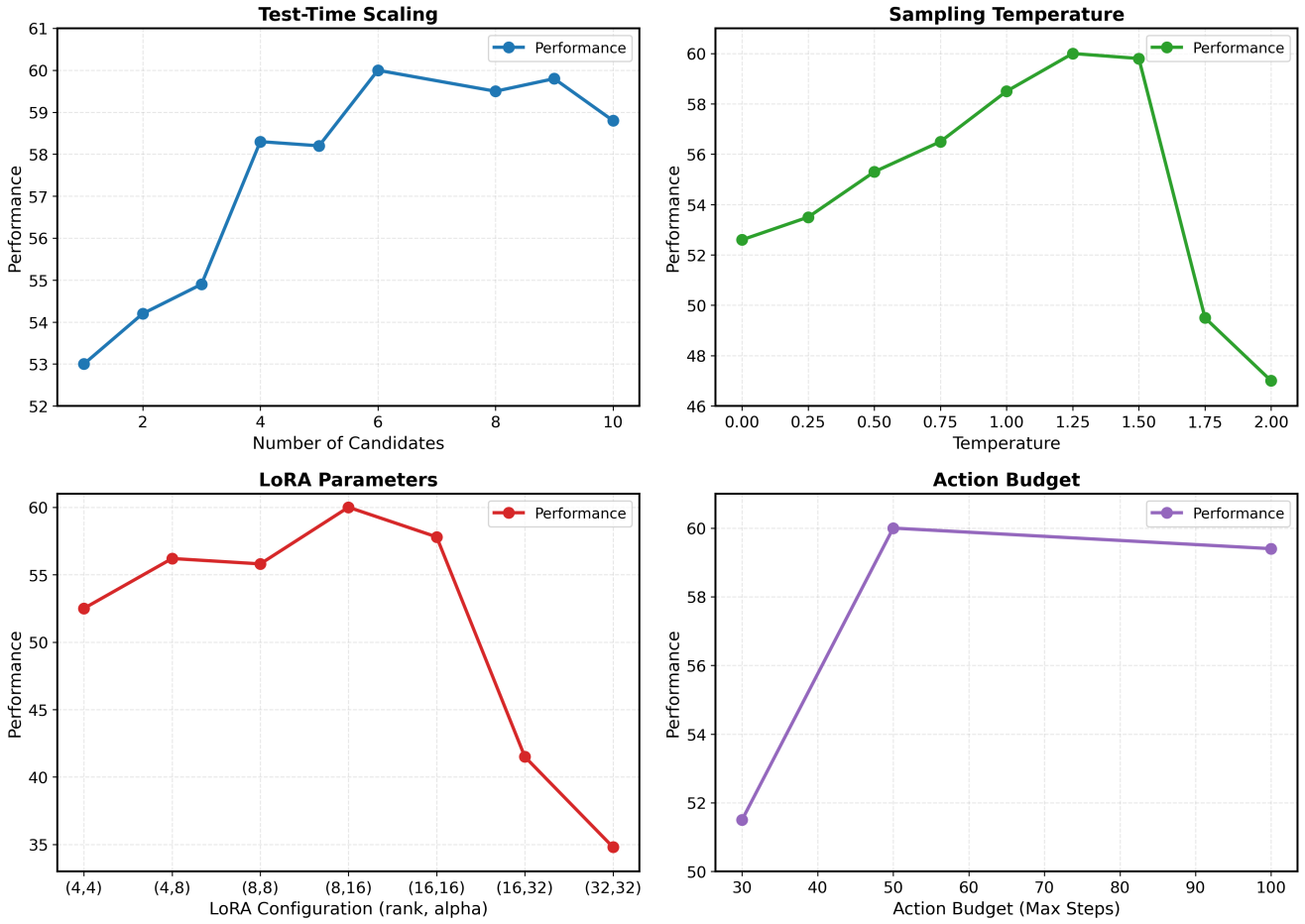

Figure 6. **Hyperparameter ablation studies on Cupboard Fitting.** *Top Left:* Performance vs. number of candidate actions. Peak performance (60.0%) occurs at N=6 candidates, demonstrating that internal reflection effectively identifies superior actions from diverse pools. Beyond N=6, performance plateaus 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.

tions (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.

C.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.

C.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.

C.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.

D. 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.

Table 4. 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×

Results. Table 4 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.

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 ad-

ditional 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.

E. Experiments on Long-Horizon Household Tasks: More Details

In this section, we provide comprehensive implementation details for our Long-Horizon Household Tasks, including task generation, validation, data construction, model architecture, training procedures, and evaluation protocols.

E.1. Why Build Upon BEHAVIOR?

BEHAVIOR-1K [18] 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.

E.2. 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

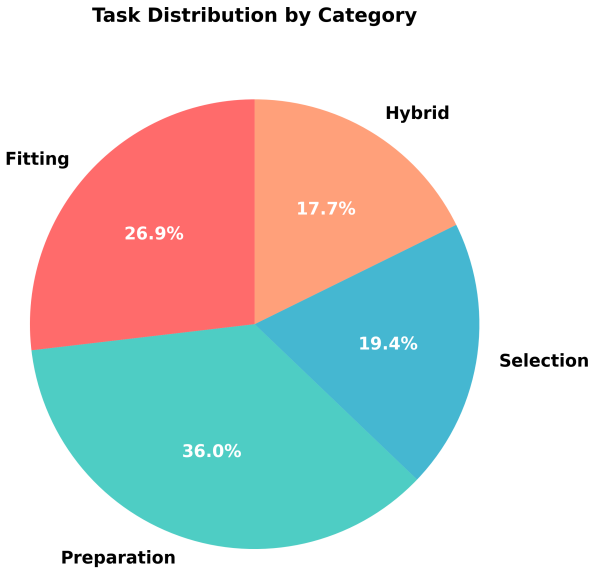

Figure 7. Distribution of task categories in the dataset.

7.

E.3. 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 1.2m × 0.8m × 0.75m 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 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.

E.4. 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.

E.5. 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:

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

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:

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

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.

E.6. Model Architecture and Training

Base Architecture. We build our 3D vision-language-action model on LLaVA-3D [40], 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.

E.7. Evaluation Protocol and Baselines

Train-Test Split. We ensure zero overlap between fine-tuning 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* [30] 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* [21] 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 [5] using learned value functions for action scoring, with a separate value head trained on our training data. *3DLLM-Mem* [12] 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* [29] and *DreamerV3* [39] 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.

F. Cupboard Fitting: More Details

F.1. 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.

F.2. 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.

G. More Qualitative Examples

In Figure 8, we show more qualitative examples. We can see that the model does get improved over reflection mech-

Figure 8. Steps and reflections are simplified for better presentations. Blue text shows internal reflection used for candidate selection, orange text shows external reflection after execution. We put reflection scores inside brackets. Red text shows retrospective reflection and model updates.

anisms.

H. Physical Experiment Setup

We validate our framework on a physical Franka Panda robotic arm in a cupboard fitting scenario. The workspace consists of a multi-compartment cupboard with compartments of varying sizes, along with different shape objects that must be placed into designated compartments. A camera mounted above the workspace captures RGB images from a top-down offset viewpoint, matching the 'top_down_offset' camera configuration used in simulation training. Before each task, we record the initial positions of objects and compartment locations in the robot base frame through calibration. Given high-level action commands from the model (e.g., "pick up the red cube and put it in the blue compartment"), the robot executes a two-phase manipulation: (1) moves to the object's registered initial position and grasps it using the parallel-jaw gripper, (2) places the object at the target compartment location computed from the registered compartment position. After each action, the system captures a post-execution image from the same camera viewpoint and provides binary success/failure feedback based on whether the object remains within the target compartment bounds, which is used to generate external reflections following the same protocol as simulation.