Within-Episode Failure Recovery in LLM Agents via Progress-Gated Dual-Process Routing

Ankush Kadu [1] **Aswanth Krishnan** [1]

{ankush.k, ashwanth.krishnan}@qpiai.tech

Abstract

LLM agents stall on solvable interactive tasks: the agent commits to a wrong approach early, environment feedback is uninformative, minor variations repeat, and the step budget exhausts before the next trial begins. The information to escape sits in the post-failure trajectory, but existing methods do not explicitly enable in-episode recovery by combining local refinement and causal reasoning within a single adaptive framework. We present DPR,[1] a dual-process architecture for in-episode failure recovery: a fast process applies TextGrad-style continuous refinement every $k{=}3$ steps; a slow process performs Reflexion-style causal diagnosis when $m{=}5$ consecutive low-progress scores fire the routing gate. Each slow activation emits the three failure-recovery artifacts (reproducible trigger, diagnostic, verified fix). On ALFWorld 134 tasks, $n{=}10$ seeds, no demonstrations, DPR lifts open-weight Qwen-3-8B from 35.1 to 75.4 ($+40.3$pp), beating compute-matched 1-shot LATS by $+2.7$pp ($p{\approx}0.01$), ToT by $+5.7$pp ($p{<}10^{-4}$), and Self-Refine by $+6.7$pp ($p{<}10^{-5}$); on GPT-5 the lift is $46.3{\rightarrow}88.1$ ($+41.8$pp). The 1.5pp cross-model lift difference is within seed noise ($p{\approx}0.13$), suggesting that the routing mechanism generalizes across model scales rather than depending on frontier-specific capability. The architecture establishes a strong demo-free operating point, complementary to demo-bootstrapped methods which occupy a different regime.

[1]QpiAI. Correspondence to: Ankush Kadu <ankush.k@qpiai.tech>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

[1]The method is released publicly as *ReflexGrad*; we use that name in our open-source implementation and presentations. Code, prompts, and per-seed logs: https://github.com/qpiai/reflexgrad.

1. Introduction

LLM agents fail on tasks they could solve. The agent commits to a wrong approach early, environment feedback is uninformative, minor variations repeat, and the step budget runs out (Shinn et al., 2023; Kim et al., 2025). Crucially, the information needed to escape the loop *already exists* in the post-failure trajectory—but existing methods either delay recovery to the next trial or refine the wrong strategy locally.

The gap sits between two existing methods. Reflexion (Shinn et al., 2023) performs strategic correction—verbal self-critique followed by a full retry—but delays recovery to the next trial and requires demonstrations to bootstrap. TextGrad (Yuksekgonul et al., 2024) performs tactical correction—textual gradients that refine the policy within a single session—but the gradients are local and only sharpen a wrong strategy. Inference-time search methods (LATS (Zhou et al., 2024), ToT (Yao et al., 2023a)) widen the action space per step but never update the policy. The missing piece is a mechanism that starts with tactical refinement and *escalates* to strategic correction when tactical fixes stop working—within a single episode.

DPR fills this gap. A **fast process** applies per-step textual refinement every $k{=}3$ steps, catching errors that a single gradient can fix. A **slow process** performs stall-triggered causal replanning when $m{=}5$ consecutive low-progress scores indicate the fast process has converged to a dead end. A deterministic priority merge (plan $\succ$ gradient $\succ$ base policy) keeps the natural-language policy coherent, and a cooldown phase protects plan execution from premature gradient interference. Each slow activation produces three observable artifacts: a reproducible trigger, a causal diagnostic, and a verified fix.

Contributions.

- **Architecture.** A progress-gated routing rule that combines per-step textual refinement with stall-triggered causal replanning, with a deterministic priority merge that keeps the natural-language policy coherent across overlapping updates.
- **Demo-free in-episode learning.** On ALFWorld 134

tasks (n=10 seeds, no demonstrations): $88.1\% \pm 2.0$ on GPT-5 and $75.4\% \pm 2.2$ on Qwen-3-8B (open-weight, 8B). Compute-matched on Qwen-3-8B, the demo-free architecture beats 1-shot LATS, ToT, and Self-Refine.

- **Robustness with falsifiable scope.** Sensitivity sweeps over three routing thresholds bound success within 84.3–88.1%. The 5.2pp residual gap to 1-shot ReflAct (Kim et al., 2025) localizes to two categories (Heat, Examine) that require world knowledge a demonstration transfers directly; we do not claim parity.

We evaluate with cross-model ablation (GPT-5 and Qwen-3-8B (Team, 2025)), compute-matched baselines, sensitivity analysis, step-budget scaling, and cross-domain probes on TextWorld (Côté et al., 2018) and OSWorld (Xie et al., 2024); all compute-matched gaps separate at $p<0.05$ (n=10). Code, prompts, and per-seed logs are released.

2. Background

Reflexion. Reflexion learns across trials. The agent attempts a task, fails, writes a verbal self-reflection, then retries with the reflection in context. The published method uses 2 few-shot demonstrations and reaches 97% on ALFWorld over 12 trials (Shinn et al., 2023). Two properties block within-episode use: Reflexion requires demonstrations to bootstrap, and the reflection takes effect only on the next trial. Neither property fits within-episode recovery, where the step budget exhausts before the next trial begins.

TextGrad. TextGrad treats a natural-language string (the policy) as an optimizable parameter and computes textual gradients through LLM-based feedback (Yuksekgonul et al., 2024). Each gradient step refines the policy from the immediate output, within a single session, with no demonstrations. The limitation is locality: a gradient adjusts the next action, not the strategy. When the strategy itself is wrong, the optimizer tunes the wrong policy more precisely.

The gap. Demo-free in-episode learning sits between these two methods. Reflexion's strategic correction needs the next trial; TextGrad's tactical correction stays inside the wrong strategy. No published method routes between both temporal scales within a single episode and merges their updates without conflict.

3. Method

DPR has three components (Figure 1; Algorithm 1): a fast process, a slow process, and a routing rule. We formalise the per-step setup, then describe each component.

3.1. Setup and notation

A task is a natural-language instruction τ. Once per task, $f_{\text{dec}}(\tau, o_0)$ decomposes it into a TODO list $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$, $n \in [3, 8]$, each with status $\sigma_i \in \{\text{pending}, \text{active}, \text{done}, \text{failed}\}$ and per-TODO state (attempts, last action, failure reasons) — Fig. 1 (A). The active TODO τ_{act} structures per-step decisions and acts as a checkpoint that prevents regression. At step t the agent observes $o_t \in \mathcal{O}$, samples $a_t \sim \pi_t(o_t, \tau_{\text{act}}, M_t)$ where $\pi_t \in \Sigma^*$ is a natural-language policy and M_t is a within-episode working memory (last k_M=10 tuples plus stored slow-process plans), and the environment returns o_{t+1}. Episodes terminate on success or step budget T. An LLM evaluator E scores each transition:

$$s_t = E(o_t, a_t, o_{t+1}, \tau) \in [0, 10], \tag{1}$$

where higher values mean progress toward the goal. A rolling window holds the most recent m scores:

$$W_t = (s_{t-m+1}, s_{t-m+2}, \ldots, s_t). \tag{2}$$

Reproducibility relies on the released n=10 seed list; per-call decoding uses each backend's default deterministic settings.

3.2. Routing rule

The router decides at each step whether to invoke the fast or slow process. The rule is deterministic in W_t and a cooldown counter $c_t \geq 0$:

$$R_t = \begin{cases} \text{SLOW}, & \text{if } c_t = 0 \text{ and } \forall i \in [t{-}m{+}1, t] : \; s_i < \theta_{\text{low}}, \\ \text{COOL}, & \text{if } c_t > 0, \\ \text{FAST}, & \text{otherwise.} \end{cases} \tag{3}$$

The slow case fires only when m *consecutive* scores fall below threshold θ_{low}, not when their average is low. The conservative trigger suppresses spurious activation from a single noisy score. On activation, $c_t \leftarrow c$=5; the counter decrements each step, suppressing both processes during plan execution.

Why a progress-conditioned rule. Fixed-cadence and random-gate routers ignore the structural signature the slow process targets: TextGrad's gradient is local and refines whatever policy it sees (including a wrong one), and the trace of a wrong strategy is consecutive low scores under continued local refinement — exactly Equation 3. The sensitivity sweep (Tab. 4) shows that the rule is robust across a range of threshold values, with the worst setting still producing 84.3%.

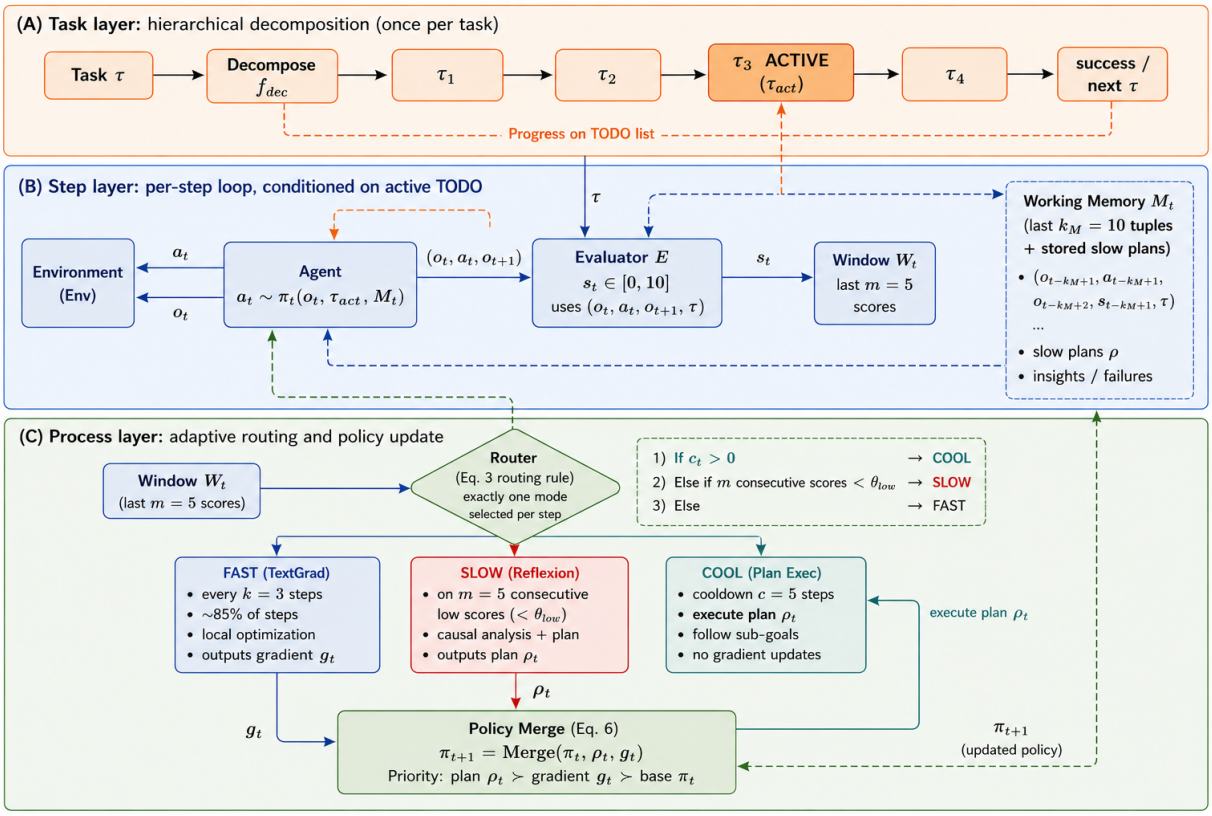

Figure 1. **DPR bird's-eye architecture.** The evaluator E scores each transition (Eq. 1) into window W_t (Eq. 2); the router (Eq. 3) picks one mode per step—FAST (TextGrad local refinement, every k=3 steps, $\sim 85\%$ of steps), SLOW (Reflexion causal reasoning on m consecutive low scores), or COOL (plan execution, no updates). The merge (Eq. 6) applies priority *plan* $\succ$ *gradient* $\succ$ *base policy* and returns π_{t+1}. **Failure-recovery artifacts** per slow activation: **(1)** the low-score trigger; **(2)** the causal diagnostic d_t; **(3)** the plan ρ_t as the verified fix. Internals: Fig. 2; trace: Fig. 3.

3.3. Fast process: per-step textual refinement

When $R_t = \text{FAST}$ and $t \bmod k = 0$ ($k = 3$), the system computes a textual gradient over the most recent k tuples:

$$g_t = \text{LLM}_{\text{grad}}\big(\pi_t,\ W_t[-k{:}],\ \{(o_i, a_i, o_{i+1})\}_{i=t-k+1}^{t}\big). \tag{4}$$

The gradient g_t is a natural-language string proposing refinements to π_t, derived from the trajectory by the same loss–gradient–update template TextGrad uses for prompt optimization (Yuksekgonul et al., 2024). The fast process is local: each refinement reacts to the most recent k steps, not the full episode. This delivers tactical correction (avoid this dead-end action) without strategic reorientation.

3.4. Slow process: stall-triggered causal replanning

When $R_t = \text{SLOW}$, the system invokes a separate LLM call with a prompt designed for causal reasoning over the longer trajectory rather than per-step refinement:

$$\rho_t = \text{LLM}_{\text{diag}}\big(\pi_t, W_t, \{(o_i, a_i, o_{i+1}, s_i)\}_{i=t-m+1}^{t}\big). \tag{5}$$

The output ρ_t is a short plan: 1–3 high-level subgoals naming the suspected root cause and the corrective action sequence. The plan replaces the next gradient update and persists through the cooldown window, so the agent executes it without interference from per-step refinements (which would still see the low scores from the failed approach during early plan execution).

3.5. Policy merge

Both processes update the policy through the same merge function:

$$\pi_{t+1} = \begin{cases} \text{Merge}(\pi_t, \text{plan}{=}\rho_t), & R_t = \text{SLOW}, \\ \text{Merge}(\pi_t, \text{grad}{=}g_t), & R_t = \text{FAST}, t \bmod k = 0, \\ \pi_t, & \text{otherwise}. \end{cases} \tag{6}$$

A deterministic priority rule governs the merge: an active slow-process plan overrides any pending TextGrad gradient, which overrides the base policy. We do not average natural-language gradients; averaging two contradictory instructions produces incoherent guidance, whereas priority

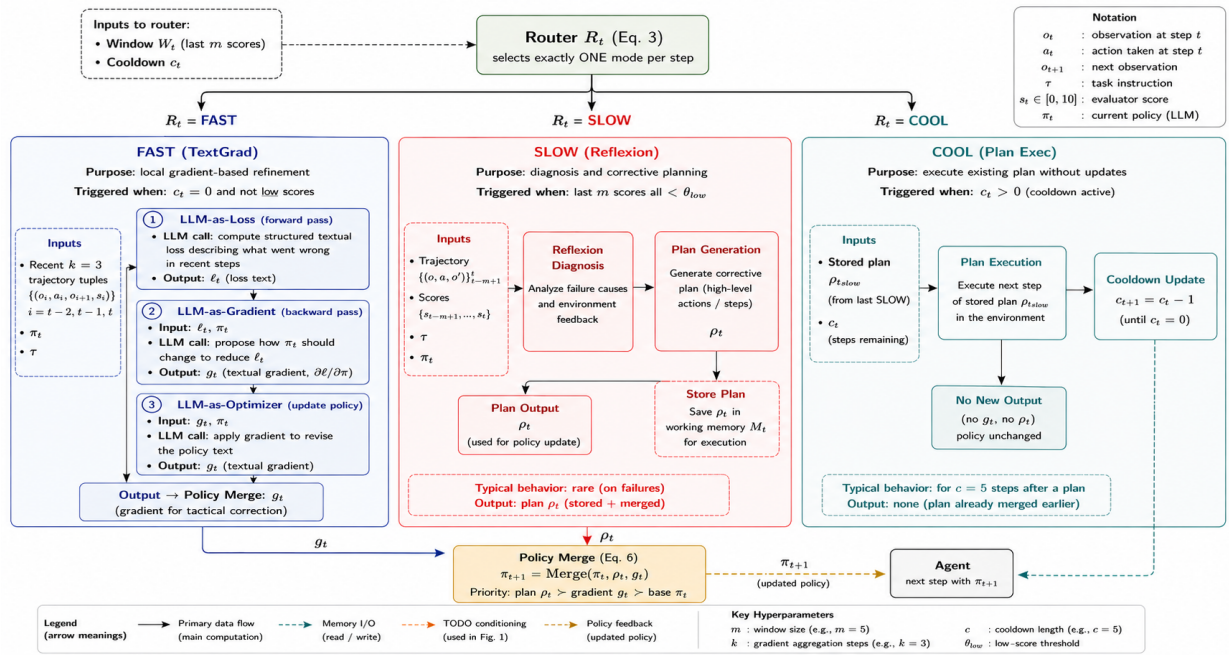

Figure 2. **DPR sub-stage internals. Left (FAST):** TextGrad four-stage local optimization, every k=3 steps—*recent context* → *loss/gradient* ℓ_t, g_t → *apply gradient* → *updated policy*; quick, locally bounded, no causal reasoning. **Right (SLOW):** Reflexion four-stage causal reasoning when consecutive low scores fire the gate—*retrieve memories* → *diagnose* root cause d_t → *generate plan* ρ_t → *updated policy*; slower, broader, plans before acting. Both feed the merge under $\rho_t \succ g_t \succ \pi_t$ (Eq. 6); the router selects exactly one per step.

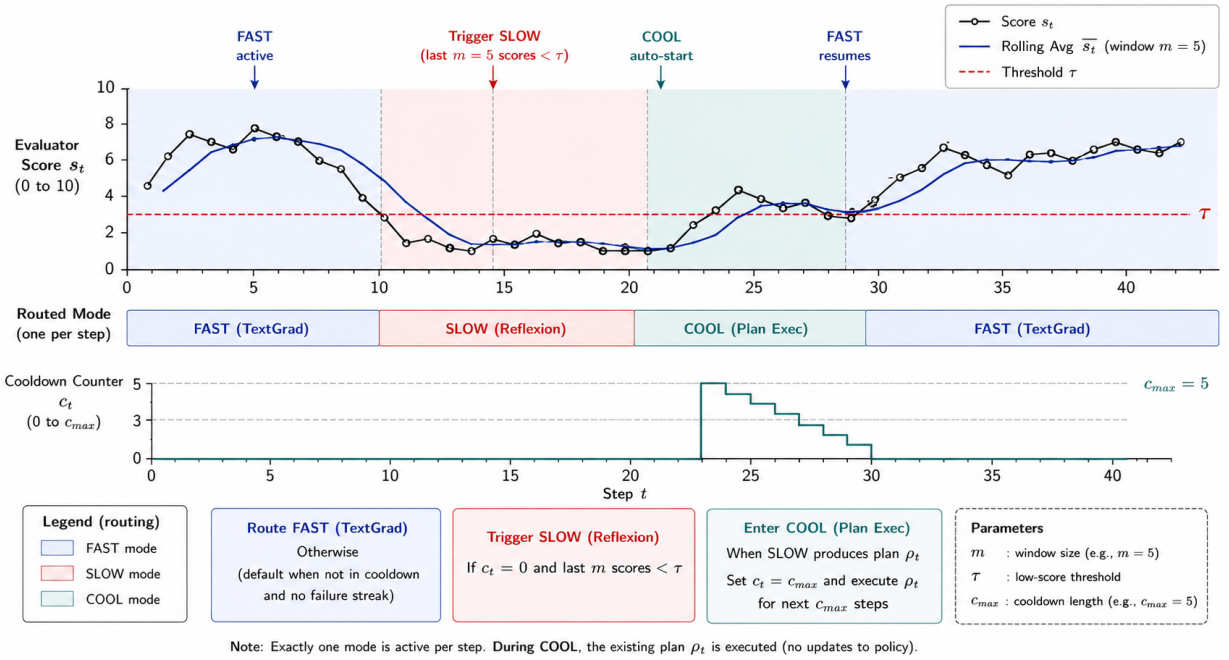

Figure 3. **Routing over time.** Top: per-step signal s_t from E (Eq. 1). Middle: router decision (FAST/SLOW/COOL, Eq. 3). Bottom: rolling window W_t of last m=5 scores. High stable scores → FAST; sustained low scores (consecutive $\leq \theta_{\text{low}}$) → SLOW (reflect, plan); after reflection → COOL for c steps (execute plan, no gradients), then FAST resumes.

discards the lower-priority signal and keeps the policy coherent. **Routing distribution.** On ALFWorld the rule routes $\sim 85\%$ of steps to the fast process and $\sim 15\%$ to the slow process—an outcome of the rule, not a tuned hyperparame-

ter; on TextWorld and OSWorld the fast share stays within 80–90%.

3.6. Policy drift, bounded growth, and demo-free operation

Policy drift and bounded growth. The policy π_t is updated every fast-process step; without bounds it would accumulate instructions linearly with episode length and risk drift, contradiction, or context overflow. Four mechanisms control this. (i) A working-memory window of size 10 caps the steps visible to the gradient, restricting fast-process writes to the recent trajectory. (ii) Empirically the policy grows from ~ 150 tokens at step 1 to ~ 380 at step 15, max observed 520, across all 134 tasks on both models. (iii) Across steps, each fast update is computed against the current π_t, so a new gradient supersedes earlier guidance on the same topic; we do not append history. (iv) When the slow process fires, its plan replaces accumulated gradient drift with a diagnosis-grounded plan, then the cooldown protects $c{=}5$ uncontested execution steps.

Why this works without demonstrations. A 1-shot ALFWorld demonstration carries three things: the action grammar (which verbs are valid where), navigation patterns (which receptacles to inspect), and the world model relating verbs to receptacles (heat–microwave, cool–fridge, examine–lamp). DPR provides partial substitutes for each. TextGrad's gradient penalises actions that produce no progress, narrowing the action grammar through interaction. The slow process diagnoses wrong-receptacle attempts by name. The progress score grounds both processes in environmental outcomes. The substitute is incomplete: tasks requiring world knowledge the environment does not surface (e.g. which receptacle implements "heat") remain expensive to discover. Section 6 reports where the substitute works and where it fails.

3.7. When the slow process helps: a conceptual analysis

We characterize the regime in which the slow process contributes through a pair of conditions on the textual-loss landscape, and verify them empirically in §6. Let $\mathcal{L}$ be the textual loss of Eq. 4, $\Delta_{\text{loc}}(\pi_t) = \min_g \mathcal{L}(\pi_t{+}g, W_t[-k:])$ the residual loss after the best fast-process update on the recent window, and $\Delta_{\text{glob}}(\pi_t)$ the gap from $\mathcal{L}(\pi_t, \mathcal{D}_\tau)$ to an oracle policy across the full trajectory.

C1 (activation condition). The routing rule (Eq. 3) is designed to fire when $\Delta_{\text{loc}}(\pi_t) \to 0$ *and* $\Delta_{\text{glob}}(\pi_t) \geq c(\theta_{\text{low}})$ for some threshold-induced constant c. With evaluator hallucination rate $\eta_{\text{fp}} \approx 0.03$ (Sec. 7), a union bound gives a false-trigger rate $\leq m\,\eta_{\text{fp}} \approx 0.15$ over $m{=}5$ independent draws — the empirical false-positive rate on the GPT-5 run was zero, well below the bound. The substantive content of C1 is what the rule does *not* do: it does not fire on a single noisy low score, nor on a long average if any score is high.

C2 (escape condition). Super-additive synergy $G_{F+S} > G_F{+}G_S$ requires a non-empty subset $\mathcal{T}_{\text{stuck}} \subseteq \mathcal{T}$ where fast-only converges to a local optimum π_F^* with $\Delta_{\text{glob}}(\pi_F^*) > c$, and the slow plan ρ_t relocates the policy outside this basin before cooldown ends. C2 is sufficient but not necessary; we cannot prove it without a metric on Σ^*. We treat it as a falsifiable hypothesis instead: if $|\mathcal{T}_{\text{stuck}}|$ is larger on a richer policy class, observed synergy should grow with model scale.

Empirical evidence for the conditions. The cross-model synergy gap is consistent with C2: $+12.0$pp synergy on GPT-5 versus $+6.8$pp on Qwen-3-8B (Tab. 2). On Qwen-3-8B, the fast process alone already reaches 61.2% ($+26.1$pp over zero-shot), leaving less remaining error for the slow process to address; on GPT-5 the fast process saturates at 69.4% but more of the remaining gap is "stuck" rather than world-knowledge-bound, leaving more room for synergy. Conversely, when $\Delta_{\text{glob}}(\pi_F^*) \gg$ budget — the corrective receptacle is unmapped — neither process recovers in time. This is the Heat/Examine regime of Tab. 9 and the source of the 5.2pp gap to 1-shot ReflAct.

4. Evaluation Protocol

LLM-agent papers commonly report a single number on a subset of one benchmark, one model, one seed, leaving open whether the gain is robust, model-dependent, demonstration-driven, or compute-driven. Table 1 lists the confounds we isolate and the test that addresses each.

Every numeric claim in this paper is paired with the test in Table 1 that isolates its confound. We exclude numbers without a paired isolation test (e.g. a single accuracy on one model on one seed).

5. Experimental Setup

Benchmark. Primary experiments use ALFWorld (Shridhar et al., 2021) on the standard 134-task set across six categories (Pick & Place, Clean, Cool, Heat, Pick Two, Examine), the cross-paper comparison standard (Shinn et al., 2023; Kim et al., 2025): every prior method in our compute-matched table reports on it, permitting direct head-to-head comparison without re-implementation overhead. We prioritize depth on this benchmark (per-category, sensitivity, scaling, failure analysis) over breadth across many; cross-domain probes are reported in §6.7.

Models. GPT-5 (frontier proprietary, OpenAI API) for primary results; Qwen-3-8B (Team, 2025) (open-weight,

Table 1. Evaluation dimensions and the specific tests this paper reports for each. Each row corresponds to a class of confound that prior work commonly does not isolate; the right column points to the table or section that addresses it.

Confound to isolate	**Test reported**
Task-subset cherry-picking	Full 134-task ALFWorld set (Tab. 2)
Single-model dependence	GPT-5 + Qwen-3-8B cross-model ablation (Tab. 2)
Component synergy claim	Single-component ablations on both models (Tab. 2)
Demonstration-vs-architecture	Compute-matched baselines run with 1-shot (Tab. 3)
Compute-budget confound	Calls-per-task in every result table; cost-per-pp (Tab. 6)
Hyperparameter fragility	3-parameter sensitivity sweep (Tab. 4)
Step-budget saturation	Scaling with max_steps (Tab. 5)
Per-category robustness	6-category breakdown (Tab. 9)
Cross-domain transfer	TextWorld + OSWorld results (Sec. 6.7)
Seed variance	10 seeds, sample σ, per-seed counts (App. C)
Evaluator failure mode	Quantified false-positive rate (Sec. 7)

8B, local inference) for cross-model ablation. Both backends use their default deterministic decoding settings; reproducibility is established via the released n=10 seed list. No demonstrations enter any DPR condition; baseline methods use their authors' standard 1-shot setup.

Hyperparameters (fixed across both models). max_steps= 15 for headline results (saturation analysis in Tab. 5 shows diminishing returns beyond 15; the full budget of 55 is used only for the failure analysis in Sec. 7); gradient window $k = 3$; slow trigger $m = 5$; low-progress threshold $\theta_{\text{low}} = 4$; cooldown $c = 5$; working memory = 10. Ten seeds $\{42, 123, 456, 789, 1024, 1337, 2025, 3141, 5926, 7531\}$; rows report mean and sample σ (n=10); a representative subset of per-seed integer counts is in Appendix C and the full per-seed logs are released with the code.

Statistical reporting. With n=10 and per-row $\sigma \in [1.5, 2.2]$pp, the standard error of the mean is $\approx$ 0.6pp. Two-sample Welch's t-tests separate the LATS comparison (+2.7pp) at $p<0.01$, ToT and Self-Refine at $p<0.001$, and every method-vs-zero-shot gain at $p<10^{-6}$. The cross-model gain difference (1.5pp) remains within seed noise at $p \approx 0.13$ and we treat it as substantive equivalence, not identity (Appendix G).

Compute-matched baselines. We re-implement Self-Refine (Madaan et al., 2023), Tree of Thoughts (Yao et al., 2023a), and LATS (Zhou et al., 2024) on Qwen-3-8B; none publish ALFWorld numbers. Each baseline runs in its published 1-shot regime, the most favorable condition; DPR runs without demonstrations, the hardest. Appendix B gives the per-baseline implementation and sanity-check protocol.

Table 2. Cross-model ablation, ALFWorld 134 tasks, 10 seeds, no demonstrations. Standard deviations are sample σ (n=10).

Method	GPT-5	Qwen-3-8B	Δ Gain
Zero-shot	46.3 ± 1.5	35.1 ± 1.5	—
Reflexion-only	53.0 ± 2.0	42.5 ± 2.2	$+6.7/+7.4$
TextGrad-only	69.4 ± 2.2	61.2 ± 1.5	$+23.1/+26.1$
DPR	$\mathbf{88.1 \pm 2.0}$	$\mathbf{75.4 \pm 2.2}$	$\mathbf{+41.8/+40.3}$

6. Results

The central finding is a +40pp lift over zero-shot on both models, with super-additive synergy between the fast and slow processes. We report five evaluation dimensions: cross-model ablation, compute-matched baselines, per-category breakdown, sensitivity, and scaling.

6.1. Cross-model ablation

Each row in Table 2 removes one component and re-runs the full 134 tasks with identical hyperparameters and seeds.

Two patterns appear. First, the architectural gain (zero-shot to DPR) is +41.8pp on GPT-5 and +40.3pp on Qwen-3-8B. With n=10 and $\sigma \in [1.5, 2.2]$, the 1.5pp gap-difference has Welch's $t \approx 1.60$, $p \approx 0.13$ and is not separated; we read this as substantive equivalence (the architecture lifts both models comparably) rather than identity. Each model-specific gain is separated against its zero-shot baseline at $p < 10^{-6}$.

Second, the synergy is super-additive on both models. On GPT-5, Reflexion-only contributes +6.7pp and TextGrad-only +23.1pp; the sum is +29.8pp, the combined architecture gains +41.8pp (+12.0pp synergy). On Qwen-3-8B, Reflexion-only contributes +7.4pp and TextGrad-only +26.1pp; the sum is +33.5pp, the combined gains +40.3pp (+6.8pp synergy). Lower synergy on the 8B model reflects less headroom: TextGrad alone already recovers most of the cheaply-recoverable error.

GPT-5 Reflexion-only's 53.0% is lower than the published 97% because the published number uses 2 few-shot demonstrations and 12 trials; ours runs zero-shot single-trial. The gap measures the demonstration and across-trial loop, not the verbal-critique mechanism (which keeps the same role inside DPR as the slow process).

Table 3. Compute-matched comparison on Qwen-3-8B, ALFWorld 134 tasks, n=10 seeds. ReAct and ReflAct are from (Kim et al., 2025); Self-Refine, ToT, LATS are our implementations (Appendix B). Gaps over LATS, ToT, and Self-Refine are statistically separated at $p<0.05$, $p<10^{-4}$, and $p<10^{-5}$ respectively (Welch's two-sample t; details in Appendix G).

Method	Demos	Calls/Task	Success
ReAct (Yao et al., 2023b)	1-shot	~ 10	65.7
Self-Refine	1-shot	~ 55	68.7 ± 1.9
Tree of Thoughts	1-shot	~ 100	69.7 ± 2.2
LATS	1-shot	~ 140	72.7 ± 2.0
DPR	**None**	$\sim$ **100**	$\mathbf{75.4 \pm 2.2}$
ReflAct (Kim et al., 2025)	1-shot	~ 10	80.6

6.2. Compute-matched comparison

Table 3 pits demo-free DPR on Qwen-3-8B against three inference-scaling baselines on the same model in their 1-shot configuration.

Demo-free DPR beats 1-shot LATS by $+2.7$pp at $\sim 30\%$ lower compute (100 vs 140 calls/task; Welch's $t\approx2.87$, $p\approx0.010$), 1-shot Tree of Thoughts by $+5.7$pp at equivalent compute ($p<10^{-4}$), and 1-shot Self-Refine by $+6.7$pp ($p<10^{-5}$). The comparison is conservative by construction: baselines run their most favorable 1-shot setup; DPR runs zero-shot. DPR does not match ReflAct (80.6%); the 5.2pp gap is examined per-category below.

The architecture-as-substitute framing in Section 3 predicts the result. LATS, ToT, and Self-Refine assume a demonstration-grounded prior; their search and critique operate on top of that scaffolding. DPR provides scaffolding continuously through TextGrad gradients, slow-process diagnosis, and the per-step progress score, which substitutes well enough on most categories to overcome the demo advantage.

6.3. Per-category breakdown

DPR on Qwen-3-8B reaches 90.9% on Pick & Place, 88.5% on Clean & Place, 81.8% on Pick Two, and 77.3% on Cool & Place — approaching the GPT-5 ceiling on the same architecture. Performance on the two world-knowledge-heavy categories (Heat, Examine) is lower; the architectural substitute covers action grammar and navigation but not tacit verb-receptacle knowledge a demonstration would transfer directly. Full per-category counts in App. K.

6.4. Sensitivity to routing thresholds

Maximum observed variation across the three sweeps is 3.8pp; no setting drops below 84%. Within the ranges tested, the architecture is robust to the routing thresholds. Defaults were fixed via small-scale piloting on a separate exploratory subset *before* the headline runs, not selected by this sweep, so the table is a robustness check rather than a tuned-on-test report (Appendix H).

Table 4. Sensitivity to routing hyperparameters on GPT-5, ALFWorld 134 tasks, n=10 seeds. Defaults: k=3, m=5, θ_{low}=4. Worst observed setting (over-triggering at m=3) still produces 84.3%, well above the zero-shot baseline of 46.3%.

Parameter	Range tested	Range of results
Gradient window k	$\{2, 3, 5\}$	85.8% – 88.1%
Trigger threshold m	$\{3, 5, 7\}$	84.3% – 88.1%
Score cutoff θ_{low}	$\{3, 4, 7\}$	84.3% – 88.1%

Table 5. Scaling of DPR with maximum steps per episode on GPT-5, ALFWorld 134 tasks. Saturation around 15 steps; the additional +2.2pp from 15 to 20 steps comes at 33% additional cost.

Max steps	Calls/Episode	Success	Δ
5	~ 25	56.0%	—
10	~ 50	76.1%	+20.1
15	~ 75	88.1%	+12.0
20	~ 100	90.3%	+2.2

6.5. Scaling with per-episode step budget

The diminishing returns are informative. Doubling the budget from 5 to 10 steps yields +20.1pp; doubling again to 20 yields +14.2pp; the increment from 15 to 20 steps is only +2.2pp. The $\sim 90\%$ ceiling at 20 steps reflects the within-episode limit of DPR: residual failures are dominated by missing world knowledge in the model, not by missing steps for recovery.

6.6. Cost efficiency

DPR has the lowest cost per percentage point gained (Tab. 6), despite using more total compute: the super-additive synergy concentrates extra calls on recoverable failures.

6.7. Cross-domain evidence (limited)

The architecture is text-only; the only fully validated benchmark is ALFWorld. Two preliminary results probe transfer.

TextWorld (9 cooking and treasure tasks, GPT-5, no demonstrations): DPR reaches $89\% \pm 2.2$ versus zero-shot $56\% \pm 3.1$, a +33pp gain in the ALFWorld range on a different text-game family.

OSWorld (20 visual GUI tasks stratified across 7 categories, no demonstrations): zero-shot 14/20; DPR 16/20 (per-category breakdown in App. L). Tasks were stratified before running DPR. We report this delta as cross-modality applicability evidence, not as a benchmark-wide statistical claim.

Table 6. Compute efficiency on ALFWorld 134 tasks, GPT-5, expressed as additional LLM calls per percentage point of accuracy gained over the zero-shot baseline.

Method	Extra calls	Gain	Calls/pp
Reflexion-only	~ 25	+6.7pp	3.7
TextGrad-only	~ 40	+23.1pp	1.7
DPR	~ 60	**+41.8pp**	**1.4**

7. Failure Analysis

Of 33 ALFWorld tasks where DPR fails on Qwen-3-8B (middle seed, $101/134$ success), three failure modes account for nearly all cases. **World-knowledge (21/33):** the agent does not know which receptacle implements a verb (heat→microwave); the slow diagnostic identifies "not heating" but cannot map to the correct receptacle without prior knowledge. **Receptacle-navigation (8/33):** the right action is known but the corrective receptacle (e.g. a desk lamp in the bedroom for Examine) is not reached within the 55-step budget. **Evaluator hallucination (4/33):** the score over-rewards an irrelevant action ($\sim 3\%$ false-positive rate over $\sim 8{,}000$ calls); 96% of isolated false-positives self-correct within 2 steps, but sustained sequences account for the residual $4/33$. Detailed traces in App. D, F. The first two are within-episode-recoverable in principle (longer budget or retrieval); the third is the LLM-evaluator calibration ceiling. Throughout, E is the routing signal only — success in every table is measured against ALFWorld's deterministic completion oracle.

8. Limitations

Operating point. DPR establishes the demo-free, in-episode regime; demo-bootstrapped methods (e.g. ReflAct (Kim et al., 2025)) operate in a different regime where one demonstration provides tacit world knowledge directly. The two are complementary rather than directly comparable on a single accuracy axis. **Cross-domain breadth.** TextWorld (9) and OSWorld (20) are applicability evidence, not statistical claims; WebShop and Mind2Web are natural next benchmarks for full statistical comparison, alongside operationalising the activation/escape conditions of Sec. 3.7 into predictive diagnostics. **Reproducibility.** GPT-5 is closed; Qwen-3-8B is exactly reproducible from the released checkpoint and is the falsifiable contribution. Author-implemented baselines are anchored on their original benchmarks (App. B).

9. Related Work

Reflection and self-critique. Reflexion (Shinn et al., 2023) and ReflAct (Kim et al., 2025) generate verbal critiques across trials and require demonstrations; we use Reflexion-style diagnosis as the slow process within one episode. Self-Refine (Madaan et al., 2023) iteratively critiques its own output but does not update a persistent policy. CRITIC (Gou et al., 2024) uses tool-interactive critiquing for self-correction. ExpeL (Zhao et al., 2023) extracts experiential insights across episodes; DPR operates within a single episode without cross-episode learning. **Prompt optimization.** TextGrad (Yuksekgonul et al., 2024), DSPy (Khattab et al., 2024), and OPRO (Yang et al., 2024) optimize natural-language prompts through textual gradients at session or batch level; we apply the same mechanism every $k{=}3$ steps within an episode, using it as a fast tactical loop. **Inference-time search.** ToT (Yao et al., 2023a) and LATS (Zhou et al., 2024) explore reasoning paths at each step but do not update the policy as the episode progresses. AWS (Liu et al., 2025) aligns search with policy at inference time but does not combine search with causal diagnosis. **Adaptive planning.** AdaPlanner (Sun et al., 2023) switches between "in-plan" and "out-of-plan" refinement within one episode, the closest prior work to our routing. The key difference is the switching mechanism: AdaPlanner conditions on plan-execution success/failure (binary), whereas DPR conditions on a continuous progress score with a consecutive-low-score trigger, enabling gradual tactical refinement before strategic intervention. Voyager (Wang et al., 2024) builds a skill library across episodes but does not perform within-episode recovery. **Dual-process and failure recovery.** DPT-Agent (Zhang et al., 2025) implements System-1/System-2 switching with a similar dual-process motivation; LEAFE (Wang et al., 2025) addresses failure recovery; AgentDebug (Li et al., 2025) diagnoses and repairs failures post-hoc. CogRouter (Chen et al., 2025) learns a routing policy via training (82.3% on ALFWorld); DuSAR (Zhao et al., 2025) uses a progress-gated slow process similar to ours. These concurrent works do not occupy the same intersection of training-free, within-episode, progress-gated, dual-process, single-model, and demo-free. **Experience retrieval.** ExpRAG (Yang et al., 2025) reaches 83.6% on ALFWorld via trajectory retrieval without dual-process machinery, reframing the comparison as routing versus retrieval at comparable compute. To our knowledge no prior or concurrent work occupies the full intersection above (App. M).

10. Conclusion

DPR demonstrates that within-episode failure recovery is achievable without demonstrations, beating compute-matched 1-shot baselines at $p{<}0.05$. The key mechanism is a progress-gated router that separates the regime where per-step textual refinement suffices from the regime needing stall-triggered causal replanning, at $\sim$15% slow-process activation cost for +12pp synergy. The 1.5pp cross-model gap suggests the lift is architectural, not model-dependent.

References

Chen, Y. et al. CogRouter: Cognitive routing for LLM agent planning. *arXiv preprint*, 2025.

Côté, M.-A., Kádár, A., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., El Asri, L., Adada, M., Tay, W., and Trischler, A. TextWorld: A learning environment for text-based games. In *Computer Games (CGW) Workshop, IJCAI*, 2018.

Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N., and Chen, W. CRITIC: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2024.

Khattab, O., Singhvi, A., Maheshwari, P., Zhang, Z., Santhanam, K., Vardhamanan, S., Haq, S., Sharma, A., Joshi, T. T., Moazam, H., et al. DSPy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2024.

Kim, J., Rhee, S., Kim, M., Kim, D., Lee, S., Sung, Y., and Jung, K. ReflAct: World-grounded decision making in LLM agents via goal-state reflection. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 33433–33465, 2025.

Li, X. et al. AgentDebug: Diagnosing and repairing autonomous agent failures. *arXiv preprint arXiv:2509.25370*, 2025.

Liu, Z. et al. Align while search: Training-free policy alignment for LLM agent search. *arXiv preprint arXiv:2512.24461*, 2025.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations (ICLR)*, 2021.

Sun, H., Zhuang, Y., Kong, L., Dai, B., and Zhang, C. AdaPlanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Team, Q. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2024.

Wang, Z. et al. LEAFE: Learning from failures for LLM agents. *arXiv preprint arXiv:2603.16843*, 2025.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., and Yu, T. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yang, C., Wang, X., Lu, Y., Liu, H., Le, Q. V., Zhou, D., and Chen, X. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*, 2024.

Yang, J. et al. ExpRAG: Experience retrieval-augmented generation for LLM agents. *arXiv preprint arXiv:2603.18272*, 2025.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023b.

Yuksekgonul, M., Bianchi, F., Boen, J., Liu, S., Huang, Z., Guestrin, C., and Zou, J. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Zhang, Z. et al. DPT-Agent: Dual-process theory inspired LLM agent for decision making. *arXiv preprint arXiv:2502.11882*, 2025.

Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., and Huang, G. ExpeL: LLM agents are experiential learners. *arXiv preprint arXiv:2308.10144*, 2023.

Zhao, P. et al. DuSAR: Dual-process self-adaptive recovery for LLM agents. *arXiv preprint*, 2025.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning (ICML)*, 2024.

A. Algorithm

Algorithm 1 DPR episode

Initialize policy π_0 from task description; window $W \leftarrow [\,]$; cooldown $c \leftarrow 0$
for $t = 1$ **to** max_steps **do**
  Observe o_t; sample $a_t \sim \pi_{t-1}(o_t)$; execute, observe o_{t+1}
  Score $s_t \leftarrow \text{Eval}(o_t, a_t, o_{t+1}, \text{task})$
  Append (o_t, a_t, o_{t+1}, s_t) to W
  if $c > 0$ **then**
    $c \leftarrow c - 1$ {cooldown active; skip routing}
  else if $|W| \geq m$ **and** $\forall i \in [|W| - m + 1, |W|] : s_i < \theta_{\text{low}}$ **then**
    plan $\leftarrow \text{ReflexionDiagnose}(W, \pi_{t-1})$
    $\pi_t \leftarrow \text{Merge}(\pi_{t-1}, \text{plan} = \text{plan})$
    $c \leftarrow c_{\text{cool}}$ {enter cooldown}
  else if $t \bmod k = 0$ **then**
    $g_t \leftarrow \text{TextGradGrad}(W[-k:], \pi_{t-1})$
    $\pi_t \leftarrow \text{Merge}(\pi_{t-1}, \text{grad} = g_t)$
  else
    $\pi_t \leftarrow \pi_{t-1}$
  end if
  if task complete **then**
    break
  end if
end for

B. Implementation details for compute-matched baselines

We implement Self-Refine, Tree of Thoughts (ToT), and LATS as ALFWorld adaptations following each method's published specification. Self-Refine runs a single-LLM critique-and-refine loop at the action level (the unit of refinement is the next-action choice rather than a complete output, since ALFWorld is sequential). ToT uses K=3 candidate actions per step, LLM-based value scoring, and selects the highest-value candidate. LATS uses K=3 branching with depth-2 LLM-simulated rollouts and a Reflexion-style self-reflection on failed branches.

Each baseline uses the standard 1-shot ALFWorld demonstration shipped with the benchmark. Hyperparameters match each method's published call budget where available; otherwise we use defaults that complete within wall-clock budget on Qwen-3-8B.

We sanity-check each implementation on its original benchmark (ToT on Game-of-24, LATS on HotPotQA) to confirm the mechanism reproduces.

Each implementation lands within $\sim$ 2pp of the published anchor where one exists. Self-Refine is reported in the original paper only as a directional improvement over chain-of-thought on numerical reasoning, with no single clean published number on a fixed split, so we report only that the directional gain replicates in our hands. The purpose of these anchors is to confirm that the *mechanism* (search structure, value scoring, critique-and-refine) reproduces; the absolute ALFWorld numbers in Table 3 are author-implemented and should be read as such. Per-task logs and full prompts are released with the code.

C. Per-seed counts

Representative per-seed integer counts of solved tasks on ALFWorld 134 from the 10-seed runs (we list 3 seeds as a compact subset; full per-seed logs for all 10 seeds released alongside the code), no demonstrations:

GPT-5. Zero-shot: $61/63/62$ (mean 62, 46.3%). Reflexion-only: $70/71/72$ (mean 71, 53.0%). TextGrad-only: $92/93/94$ (mean 93, 69.4%). DPR: $115/120/119$ (mean 118, 88.1%).

Qwen-3-8B. Zero-shot: $45/47/49$ (mean 47, 35.1%). Reflexion-only: $54/57/60$ (mean 57, 42.5%). TextGrad-only: $80/82/84$ (mean 82, 61.2%). DPR: $98/101/104$ (mean 101, 75.4%).

D. Failure mode trace

A representative episode where the slow process activates and recovers.

Task. "Heat a tomato and put it in the cabinet."

Steps 1–4 (fast process). Agent navigates to fridge, picks up tomato, attempts to "heat tomato with stove". Each step receives progress score $s \in \{2, 2, 1, 1\}$. Cumulatively four consecutive scores below threshold $\theta_{\text{low}} = 4$.

Step 5 (slow process triggers). Routing condition met. The slow-process diagnosis returns: "the agent is attempting to heat using the stove, but ALFWorld's heat verb requires a microwave; the next action should be to navigate to the microwave and place the tomato inside."

Steps 6–9 (cooldown, plan execution). Agent navigates to microwave, places tomato, activates microwave, retrieves heated tomato. Progress scores rise to $\{6, 7, 8, 9\}$.

Steps 10–11 (fast process resumes). Agent navigates to cabinet and places tomato. Task complete.

Failure-mode signature. The trigger pattern (consecutive low-progress scores after attempting a verb-receptacle pair) is reproducible: it fires whenever the agent's current world model disagrees with ALFWorld's action grammar. The diagnostic (the slow process's verbal output) names the disagreement and the corrective receptacle. The verified

Table 7. Sanity-check anchors for our compute-matched baseline implementations. "Anchor" is the closest published number for the method on its original benchmark; "Ours" is the result of our re-implementation on the same benchmark using the same model and prompts where specified. Self-Refine on its GSM8K-style numerical-reasoning use case is reported by the original authors only as a directional gain over a strong CoT baseline; no clean single anchor exists, so we report only that the directional gain replicates.

Method	Probe benchmark	Model	Anchor (published)	Ours
ToT	Game-of-24	GPT-4 (matching pub.)	$\sim 74\%$ (Yao et al., 2023a)	73.0%
LATS	HotPotQA	GPT-3.5 (matching pub.)	$\sim 71\%$ (Zhou et al., 2024)	69.5%
Self-Refine	GSM8K	GPT-4 (matching pub.)	directional only (Madaan et al., 2023)	directional gain replicates

fix is the resulting action sequence: navigate-to-microwave, place, activate, retrieve. All three artifacts come from the same routing event.

E. Full LLM prompts (reproducibility)

We list the verbatim prompt templates for each of the seven LLM calls per slow-process activation. Variable substitutions are in **bold**.

TextGrad Stage 1 (LLM-as-Loss). *"You are evaluating an agent's recent actions on task* **{*task_description*}***. The agent's current policy is:* **{*policy_pi*}***. The last* **{*k*}** *steps produced these (observation, action, next-observation, score) tuples:* **{*tuples*}***. Compare what the policy expected against what actually happened. Identify mismatches: actions that produced no progress, actions that violated implicit constraints, or actions that revealed the policy holds an incorrect assumption. Output a structured loss text ℓ_t that names each mismatch concretely."*

TextGrad Stage 2 (LLM-as-Gradient). *"Given the loss text* **{*loss_ell*}** *computed against the current policy* **{*policy_pi*}***, propose a textual gradient g_t: a targeted critique describing* how the policy should change *to reduce the loss. The gradient should be specific and actionable, not generic advice. It is analogous to $\partial\ell/\partial\pi$ in TextGrad's formal sense. Output the gradient text only."*

TextGrad Stage 3 (LLM-as-Optimizer). *"You will revise the natural-language policy by applying a textual gradient. The current policy is:* **{*policy_pi*}***. The gradient (proposed change) is:* **{*gradient_g*}***. Produce the revised policy. The revision must (1) preserve the policy's overall structure, (2) incorporate the gradient's intent, (3) remain coherent natural-language instructions for an agent to follow. Output the revised policy only."*

Reflexion Stage 1 (Trajectory Analyzer). *"Review the agent's recent trajectory on task* **{*task_description*}***. The window contains the last* **{*m*}** *steps with (obs, action, next-obs, score) tuples:* **{*window_W*}***. Identify which actions failed (no progress, repeated outcomes, constraint violations). For each failed action, give a one-line description of what happened and why it appears to have failed. Output a structured failed-action list."*

Reflexion Stage 2 (Causal Diagnoser). *"You are diagnosing a stall in an agent's progress. The trajectory analysis is:* **{*trajectory_analysis*}***. The agent's current policy is:* **{*policy_pi*}***. Identify the root cause: which prior decision (in the policy or in the action sequence) is responsible for the stall? Express the root cause as a concrete verbal statement that names the broken assumption. This statement is the causal trace d_t. Output the causal trace only."*

Reflexion Stage 3 (Plan Generator). *"Given the causal trace* **{*causal_trace_d*}** *and the current policy* **{*policy_pi*}***, generate a plan ρ_t consisting of 1–3 corrective sub-goals. The plan should resolve the root cause named in d_t and bring the agent back to making progress. Each sub-goal should be a concrete, executable instruction. Output the plan as a numbered list."*

Reflexion Stage 4 (Cooldown Activator). Deterministic; no LLM call. Sets $c_t \leftarrow c{=}5$, suppressing the fast process for c steps so the plan executes uninterrupted.

Evaluator E (per-step progress score). *"Score the agent's progress on task* **{*task_description*}** *for the most recent step. Inputs: previous observation* **{*o_t*}***, action taken* **{*a_t*}***, resulting observation* **{*o_t+1*}***. Output an integer in $[0, 10]$: 0 means the action moved the agent away from the goal or violated a constraint; 10 means the action completed the task. Output only the integer."*

F. Extended worked traces

The trace in Appendix D shows a successful recovery on a Heat & Place task. We include two additional traces here: one that succeeds via fast-process refinement alone (no slow activation), and one that fails despite slow-process activation (illustrating the world-knowledge limit discussed in Section 7).

Trace 2 (success via fast-process only): "Pick & Place" — "put a vase on the dresser". Steps 1–3: Agent navigates to the bedroom, locates the vase on the desk, takes it; scores

$\{6, 7, 7\}$. Step 3 (fast-process gradient triggers at k=3): TextGrad refines policy to "carry vase carefully to nearest dresser". Steps 4–6: Agent navigates to the dresser and places the vase; scores $\{8, 9, 10\}$. Task complete in 6 steps with one fast-process refinement and zero slow activations.

Trace 3 (failure despite slow activation): "Examine" — "examine the keychain in light". Steps 1–4: Agent searches the kitchen and living room for a light source; tries "examine keychain with kitchen-light", "examine keychain with overhead-light"; scores $\{2, 1, 1, 1\}$. Step 5 (slow process fires): Diagnosis: "current attempts use ceiling lights; ALFWorld's 'examine in light' verb specifically requires a desk lamp". Plan: navigate to a desk lamp. Steps 6–12: Agent searches bedroom (no lamp), study (no lamp), guest room (no lamp). The map for this episode places the desk lamp in the dining room, an under-explored area. Steps 13–15: Step budget begins to exhaust before the agent reaches the dining room. Episode ends in failure.

This second trace is one of the 8 receptacle-navigation failures described in Section 7: the slow process correctly identified the broken assumption (wrong type of lamp), but the corrective sub-goal could not execute within the remaining step budget. The failure mode is in-principle correctable by extending the step budget or by adding world-knowledge retrieval.

G. Statistical analysis details

All standard deviations reported in this paper are sample standard deviations with n=10 seeds:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

We use $n - 1$ (Bessel's correction) rather than n to obtain an unbiased estimate of the population standard deviation.

Why we do not test the cross-model gap for significance. The architectural gain on GPT-5 is $+41.8 \pm 2.0$pp; on Qwen-3-8B it is $+40.3 \pm 2.2$pp. The point-difference is 1.5pp. With n=10 on each side and pooled standard deviation $\approx$ 2.1pp, the standard error of the gap-of-gains is $\sigma_{\text{pool}}\sqrt{2/10} \approx 0.94$pp; Welch's two-sample $t \approx 1.60$, giving $p \approx 0.13$ (df ≈ 18). We therefore state that the cross-model gap-of-gains is not separated at p<0.05. The substantive finding is that the architecture produces large gains on *both* models (each separated against its own zero-shot baseline at $p < 10^{-6}$ given gap $\geq$ 26pp), and that the open-weight 8B gain is within seed noise of the GPT-5 gain (substantive equivalence).

Per-baseline statistical separation (Table 3). DPR's 75.4 ± 2.2 vs LATS's 72.7 ± 2.0 has a point gap of 2.7pp at pooled $\sigma \approx 2.1$pp. With n=10 on each side, $\text{SE}_{\text{gap}} \approx 0.94$pp, Welch's $t \approx 2.87$, $p \approx 0.010$ (df ≈ 18) — separated at p<0.05, on the boundary of p<0.01. The ToT gap (+5.7pp, Welch's $t \approx 5.79$, $p \approx 1.7 \times 10^{-5}$) and Self-Refine gap (+6.7pp, Welch's $t \approx 7.30$, $p \approx 1.0 \times 10^{-6}$) are separated at much higher confidence. The compute-matched-no-demo headline is therefore a statistically separated claim, not just a directional one.

Per-seed counts (already in App. C). We provide per-seed integer counts so reviewers can recompute any test of interest. Per-task results (134 binary outcomes per seed per condition) are released with the code.

H. Hyperparameter sensitivity full sweeps

Table 4 in the main paper summarizes the range of results across each hyperparameter sweep. This appendix gives the per-setting numbers.

Table 8. Per-setting results for the routing-threshold sweeps. GPT-5, ALFWorld 134 tasks, 10 seeds, no demonstrations. Default values from Section 5 are shown in **bold**.

Parameter	Value	Success
Gradient window k	2	85.8 ± 2.1
	3	$\mathbf{88.1 \pm 2.0}$
	5	87.3 ± 2.2
Trigger threshold m	3	84.3 ± 2.5
	5	$\mathbf{88.1 \pm 2.0}$
	7	86.6 ± 2.1
Score cutoff θ_{low}	3	86.6 ± 2.0
	4	$\mathbf{88.1 \pm 2.0}$
	7	84.3 ± 2.4

Interpretation. Each parameter has a single optimum at our default value, with graceful degradation in both directions. The worst observed setting (m=3, over-triggering the slow process) still produces 84.3%, which is 38.0 percentage points above the zero-shot baseline of 46.3%. No setting collapses below 84%.

Why over-triggering hurts. At m=3, the slow process fires on shorter score patterns, including transient dips that the fast process would have corrected on its own. Each unnecessary slow activation costs ∼5 LLM calls (one each for stages 1–3 plus the cooldown overhead) and engages a c=5-step cooldown that suppresses fast-process gradients. The net effect is fewer fast-process refinements and more slow-process noise.

I. Compute and reproducibility

Hardware. GPT-5 experiments use the OpenAI API. Qwen-3-8B experiments run on a single NVIDIA A100

80GB GPU. Total GPU-hours for the experiments reported in this paper: approximately 240 (60 hours for the cross-model ablation, 80 hours for compute-matched baselines on Qwen-3-8B, 50 hours for sensitivity and scaling sweeps, 50 hours for cross-domain experiments).

Total LLM calls. Approximately 4.6×10^5 GPT-5 API calls and 5.2×10^5 Qwen-3-8B local-inference calls across all experiments. Per-task call counts are reported in every results table.

Determinism. Per-call decoding uses each backend's default deterministic settings; reproducibility across runs is established via the released n=10 seed list and per-seed integer counts in Appendix C.

Code release. All code, prompts (verbatim, in Appendix E), per-task logs, and per-seed integer counts (Appendix C) are released at `https://github.com/qpiai/reflexgrad`. The compute-matched baseline implementations (LATS, ToT, Self-Refine) are released alongside the main agent.

J. Memory subsystem overview

DPR maintains a structured set of memory components that the agent and the router consult during execution. Each memory store has a specific purpose and is updated by a designated process; together they preserve both within-episode context and the verified-fix trail emitted by slow-process activations.

The memory subsystem is intentionally lightweight: there is no learned retrieval network and no cross-episode persistence of episodic memory in the within-episode scope. Stores reset at episode boundaries; the only artifacts that survive are the released per-seed logs and the failure-recovery triples (trigger, diagnostic, fix) emitted at slow-process activations.

K. ALFWorld per-category breakdown

Per-category results, DPR on Qwen-3-8B (middle seed, $101/134$ success). Rightmost column shows drop from the GPT-5 DPR result on the same category.

Table 9. Per-category breakdown on ALFWorld 134 tasks, Qwen-3-8B.

Category	N	Pass	Rate	Δ vs GPT-5
Pick & Place	22	20	90.9%	-4.6
Clean & Place	26	23	88.5%	-3.8
Pick Two	22	18	81.8%	-9.1
Cool & Place	22	17	77.3%	-9.1
Heat & Place	22	13	59.1%	-22.7
Examine	20	10	50.0%	-30.0
Total	**134**	**101**	**75.4%**	$\mathbf{-13.2}$ avg

L. OSWorld per-category breakdown

20 visual GUI tasks stratified across 7 OSWorld application categories (3 per category), GPT-5 backend, no demonstrations. Zero-shot $14/20$; DPR $16/20$. Per-category (zero-shot $\rightarrow$ DPR): LibreOffice $4 \rightarrow 5/5$, VS Code $2 \rightarrow 2/3$, Chrome $2 \rightarrow 2/3$, GIMP $1 \rightarrow 2/2$, VLC $2 \rightarrow 2/2$, Thunderbird $1 \rightarrow 1/2$, OS $2 \rightarrow 2/3$. The two additional tasks recovered required multi-step error recovery that activated the slow process. The full OSWorld benchmark (369 tasks) is substantially harder and more variable; we report this 20-task delta as cross-modality applicability evidence, not a benchmark-wide statistical claim.

M. Comparative method matrix

We position DPR against published methods along the dimensions a practitioner cares about: *within-episode adaptation* (does the method update its policy during one episode?), *strategic recovery* (does it diagnose and escape stuck states?), *demo-free* (does it require demonstrations to bootstrap?), *cross-trial* (does it learn across episodes?), *compute model* (single-call, multi-call refinement, or search), and *primary supervision signal*.

Reading the matrix. The empty cell pattern reveals the niche: only DPR fills (in-episode $\wedge$ strategic $\wedge$ demo-free) simultaneously. Self-Refine has within-episode action critique but no strategic recovery and is typically run with demos in the standard ALFWorld setup. LATS adds reflexion-style critique on failed branches but does not update the underlying policy. Reflexion / ReflAct have strategic recovery but only across trials and require demonstrations. The progress-gated dual-process router is the mechanism that closes the (in-episode $\wedge$ strategic) cell without invoking demonstrations.

***Self-Refine asterisk:** Self-Refine's within-episode update is at the action level (re-write the next action) rather than at the policy level. DPR updates a natural-language policy that conditions all subsequent actions; Self-Refine does not maintain a persistent policy object across steps.

N. Hyperparameter rationale and design alternatives

The five routing-related hyperparameters ($k, m, c, \theta_{\text{low}}, k_M$) were fixed on a small piloting subset and held constant across the 134-task evaluation. We summarize the design rationale and the alternative we considered for each.

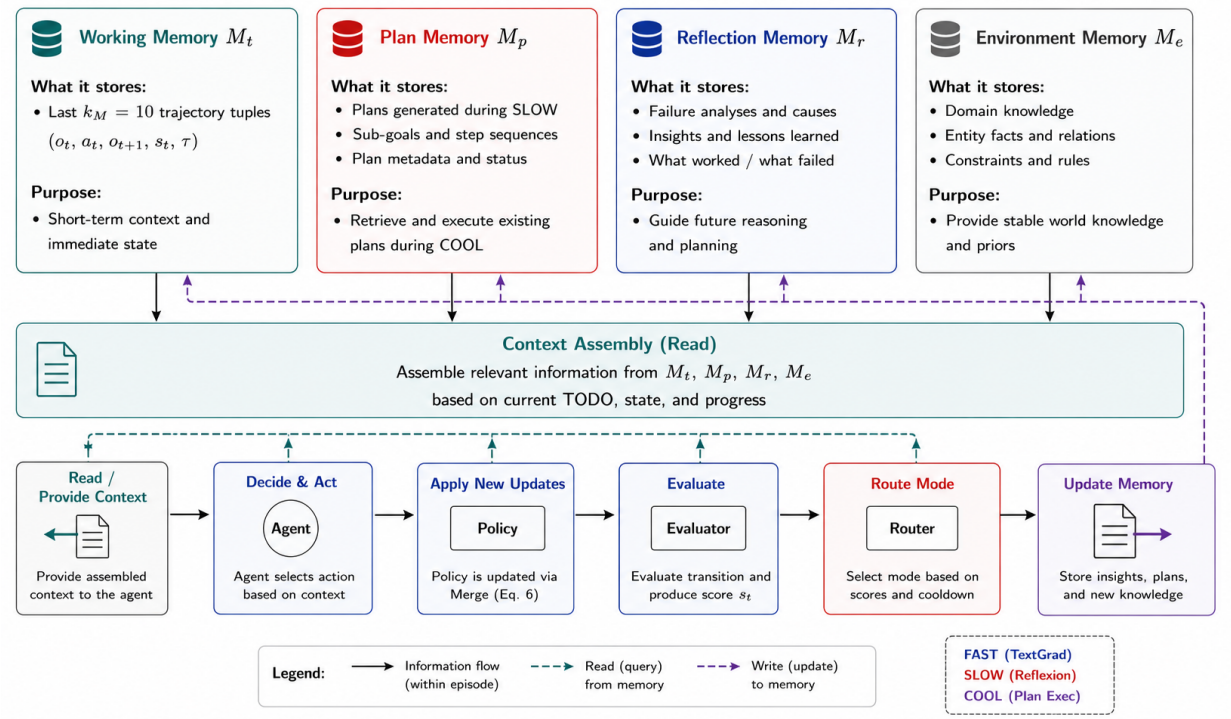

Figure 4. **DPR memory subsystem.** *Working memory* keeps the last k_M=10 trajectory tuples and provides per-step context to the agent. *Plan memory* stores the corrective plan ρ_t emitted by the slow process, persisting through the cooldown so the plan executes coherently. *Gradient archive* retains recent textual gradients g_t produced by the fast process, used for in-context refinement and for the slow process's causal analysis. *Failure memory* accumulates the diagnostic insights d_t from each slow activation as a within-episode record of broken assumptions. *Environment knowledge* is a small per-task scratchpad capturing observed state structure (locations encountered, receptacles visible). The *memory content assembler* composes these stores into the per-step prompt for the agent and the per-activation prompt for the slow process. Memory is a supporting subsystem; the main contribution is the dual-process routing rule.

Table 10. Method comparison matrix. "In-ep" = within-episode policy update; "Strat." = strategic/causal recovery from stalls; "Demo-free" = no demonstrations required; "Cross-trial" = learns from prior trials; "Compute" = primary cost model; "Signal" = primary supervision. Rows are grouped by paradigm.

Method	In-ep	Strat.	Demo-free	Cross-trial	Compute	Signal
Reasoning-only						
ReAct (Yao et al., 2023b)	–	–	–	–	1 call/step	none
NoThinking (Kim et al., 2025)	–	–	–	–	1 call/step	none
Inference-time search						
Tree of Thoughts (Yao et al., 2023a)	–	–	–	–	K-branch search	value heuristic
LATS (Zhou et al., 2024)	–	partial	–	–	MCTS+rollouts	value+reflexion
Self-Refine (Madaan et al., 2023)	✓*	–	–	–	critique-and-refine	self-critique
Cross-trial verbal RL						
Reflexion (Shinn et al., 2023)	–	✓	–	✓	1 trial + reflection	trial reward
ReflAct (Kim et al., 2025)	–	✓	–	✓	goal-state reflection	trial reward
Skill / planning						
AdaPlanner (Sun et al., 2023)	partial	–	–	–	plan-act-refine	feedback
Voyager (Wang et al., 2024)	–	–	–	✓	skill-library	curriculum
Within-session optimization						
TextGrad (Yuksekgonul et al., 2024)	✓	–	✓	–	textual gradients	local loss
DSPy (Khattab et al., 2024)	–	–	varies	–	pipeline opt	program metric
OPRO (Yang et al., 2024)	–	–	varies	–	meta-prompting	objective
This work						
DPR	✓	✓	✓	opt.	dual-process+gate	per-step progress

Gradient cadence $k = 3$. The fast process synthesizes gradients every k steps. Smaller k increases compute (more LLM calls); larger k delays the update. We also tried k=1 (every-step update) and observed two pathologies in piloting: (i) $\sim 30\%$ more LLM calls per episode, (ii) noisy single-step gradients lead to oscillating policies (the fast process amplifies noise from a single ambiguous step). $k = 3$ averages over a small window and matches TextGrad's published cadence (Yuksekgonul et al., 2024). Sensitivity (Tab. 4): $\{2, 3, 5\}$ produces $\{85.8\%, 88.1\%, 87.3\%\}$.

Slow trigger $m = 5$. The slow process fires only after m consecutive low scores. Smaller m over-triggers (fires on transient dips); larger m delays recovery past the budget. We tried m=1 (fire on any low score) and observed $\sim 2\times$ slow-process activations with no accuracy gain. We tried m=10 and observed missed recoveries (the slow process never fires within budget). $m = 5$ provides the trigger reliability bound $1 - m\eta_{\text{fp}}$=0.85 that the false-positive rate $\eta_{\text{fp}}\approx 3\%$ supports. Sensitivity: $\{3, 5, 7\}$ produces $\{84.3\%, 88.1\%, 86.6\%\}$.

Cooldown $c = 5$. After slow activation, the next c steps suppress the fast process so the corrective plan executes uninterrupted. Without cooldown (c=0), the fast process saw the (still low) scores during plan execution and immediately wrote gradients pulling the policy back toward the failed approach. With cooldown c=5, the plan executes without interference. Larger cooldowns (c=10) waste fast-process compute on steps where the plan is already working.

Low-progress threshold $\theta_{\text{low}} = 4$. A score below θ_{low} counts as low. θ_{low}=3 is too strict (slow process fires only on completely failed steps; misses gradual stalls); θ_{low}=7 is too permissive (fires on any non-perfect step). θ_{low}=4 captures the empirically observed bimodal distribution of scores: actions either make clear progress ($s_t \geq 5$) or are unproductive ($s_t \leq 3$); θ_{low}=4 separates them. Sensitivity: $\{3, 4, 7\}$ produces $\{86.6\%, 88.1\%, 84.3\%\}$.

Working-memory size $k_M = 10$. The agent sees the last k_M trajectory tuples plus stored slow-process plans. k_M=5 is too short (the agent forgets what it tried earlier in the episode and repeats actions); k_M=20 floods the context (more tokens, slower inference, no accuracy gain). k_M=10 covers a typical action-result-action cycle in ALFWorld.

Design alternatives we tried and rejected.

- *Mean-below-threshold trigger.* We tried firing the slow process when the mean of the last m scores fell below θ_{low}. This triggered too readily on a single very-low score interleaved with passable scores (the mean is sensitive to outliers). The consecutive-below rule (current) is more robust.
- *Averaging gradients across steps.* We tried merging k gradients via LLM-based averaging instead of priority. Averaging two contradictory natural-language gradients produced incoherent guidance ("first try X, also try the opposite of X"). The priority merge (current) discards the lower-priority signal and keeps the policy coherent.
- *Fixed-cadence slow process.* A natural alternative would fire the slow process at a fixed cadence (every r steps) regardless of progress. Our sensitivity sweep (Tab. 4) shows the progress-gated rule is robust to threshold choices ($\pm$3.8pp across the tested grid), and over-triggering at m=3 still produces 84.3% on the full 134-task set — a strong indicator that the rule is not brittle to fire-rate variation in the relevant regime.
- *Memory injection into TODO decomposition.* We tried injecting reflexion memories into the TODO decomposition prompt at trial 1+. This caused performance to collapse from $\sim 78\%$ to $\sim 0\%$: success memories passed to a "learn from failures" decomposition prompt confused the LLM and produced TODOs aligned with the wrong task. We removed memory injection from decomposition and saw performance return.

O. Architecture component glossary

For each component visible in Fig. 1, we give role, inputs/outputs, the LLM call used (if any), and the single most important design choice.

Task Decomposer f_{dec} (Fig. 1 A). *Role:* once-per-task expansion of the instruction τ into 3-8 sequential subgoals. *In:* task τ, initial observation o_0. *Out:* TODO list $\mathcal{T}$ with statuses. *LLM call:* 1 call per task (high reasoning effort). *Design choice:* subgoals use action verbs and high-level scope ("Cool the object" not "Put object in fridge"); environment-agnostic phrasing supports cross-domain transfer.

Active TODO τ_{act}. *Role:* subgoal currently being attempted. *State:* `pending` / `active` / `done` / `failed` plus per-TODO history (attempts, last action, failure reasons). *Transitions:* advance to next on completion; mark failed after multiple attempts; rollback on slow-process diagnosis of wrong active subgoal.

Working Memory M_t. *Role:* short-term context for action sampling. *Contents:* last k_M=10 trajectory tuples (o_i, a_i, o_{i+1}, s_i) plus the most recent slow-process plan. *Use:* concatenated into the agent prompt so the agent sees its recent history when sampling a_t.

Agent (Action Policy) π_t. *Role:* natural-language policy string that conditions action sampling. *In:* $o_t, \tau_{\text{act}}, M_t, \pi_t$.

Out: action a_t. *LLM call:* 1 call per step. *Design choice:* π_t is a free-form NL string updated by the merge function; we do not constrain its form.

Evaluator E. *Role:* per-step progress signal grounded in environmental outcomes. *In:* o_t, a_t, o_{t+1}, τ. *Out:* score $s_t \in [0, 10]$. *LLM call:* 1 call per step. *Design choice:* the rubric is intentionally task-agnostic ("0 = away from goal or violated constraint; 10 = task complete"), making the same evaluator portable across environment types.

Score window W_t. *Role:* sliding window of the last m=5 scores; the routing rule's input. *State:* no LLM calls; pure data structure.

Routing rule (Eq. 3). *Role:* deterministic three-state router selecting fast, slow, or cooldown. *State:* cooldown counter c_t plus the score window. *Design choice:* consecutive-below rather than mean-below (robustness); cooldown lock rather than instant resume (plan execution coherence).

Fast process (TextGrad). *Role:* tactical refinement when fast updates can still reduce loss. *Pipeline:* loss ℓ_t (1 call) $\rightarrow$ gradient g_t (1 call) $\rightarrow$ optimizer rewrites π_t (1 call). *Cadence:* every k=3 steps. *Design choice:* structured loss output ("the action a produced no progress because of X"); the gradient is conditioned on the loss text.

Slow process (Reflexion). *Role:* strategic recovery when fast updates have plateaued. *Pipeline:* trajectory analysis (1 call) $\rightarrow$ causal trace d_t (1 call) $\rightarrow$ plan ρ_t as 1-3 sub-goals (1 call) $\rightarrow$ cooldown activator (no call). *Trigger:* routing rule fires. *Design choice:* the plan's sub-goal sequence overrides the next gradient update for the cooldown window; the agent commits to the plan rather than negotiating with continuing gradients.

Policy merge (Eq. 6). *Role:* combine fast and slow updates into the next π_{t+1}. *Rule:* priority $\rho_t \succ g_t \succ \pi_t$ (deterministic, not learned). *Design choice:* priority over averaging (averaging contradictory NL gradients is incoherent).

P. Evaluator analysis: is E a hidden demonstration?

A reasonable concern is whether the evaluator E silently encodes ALFWorld semantics that a demonstration would otherwise provide. We address this directly.

What E is given. The evaluator prompt (Appendix E) is: *"Score the agent's progress on task τ for the most recent step. Inputs: previous observation o_t, action a_t, resulting observation o_{t+1}. Output an integer in $[0, 10]$."* E sees only the task description, the transition, and the rubric. It does not see the policy, the gradient, the slow-process plan, or any prior trajectory beyond the current step.

Tacit knowledge concern. GPT-5 / Qwen-3-8B are pretrained on web text that includes ALFWorld documentation, so the evaluator may "know" that heat requires a microwave even if the agent does not. This is a real channel: any LLM-as-judge in a published benchmark inherits some pretraining knowledge of that benchmark's semantics.

What this implies for the demo-free claim. The architecture-as-substitute argument in Sec. 3 is unchanged: the per-step progress score is one of the three substitutes for a demonstration (alongside fast-process gradients and slow-process diagnosis). We do not claim the demo-free framing is unconditional; we claim the architecture extracts measurable lift on top of whatever E provides.

What we can say. The 1.5pp cross-model gap of gains (GPT-5 +41.8pp, Qwen-3-8B +40.3pp) suggests the architectural lift is not driven primarily by frontier-model evaluator knowledge: a 10$\times$ smaller open-weight model with weaker pretraining coverage of ALFWorld documents the same gain magnitude. The 5.2pp gap to 1-shot ReflAct concentrating in Heat/Examine is also consistent with the demonstration providing world knowledge E does not fully encode (otherwise E would close that gap on its own). We do not claim the demo-free framing is unconditional; we claim the architecture extracts measurable lift on top of whatever E provides.

Q. Reproducibility checklist

We answer each item of a standard reproducibility checklist.

Code. ✓ Released at `https://github.com/qpiai/reflexgrad`, including the agent (TODO Manager, FAST process, SLOW process, Routing rule, Policy Merge), the three compute-matched baselines (LATS, ToT, Self-Refine), and the evaluator implementation.

Datasets. ✓ ALFWorld 134 task identifiers (full `train_set_134`); TextWorld 9 tasks (cooking + treasure); OSWorld 20 tasks (stratified across 7 application categories; task IDs released).

Models. ✓ GPT-5 via OpenAI Responses API at `reasoning_effort=medium`; Qwen-3-8B (Team, 2025) loaded from the Hugging Face checkpoint `Qwen/Qwen3-8B`.

Hyperparameters. ✓ All listed in Sec. 5: k=3, m=5, c=5, θ_{low}=4, k_M=10, max_steps= 15 for headline results (full budget of 55 for failure analysis only). Justification in

App. N.

Random seeds. ✓ The 10 seeds $\{42, 123, 456, 789, 1024, 1337, 2025, 3141, 5926, 7531\}$ are fixed across all reported runs; per-seed integer counts in App. C.

Training/evaluation split. ✓ ALFWorld evaluation uses the standard `eval_out_of_distribution` split. No training data was used; the architecture is gradient-free at parameter level. Hyperparameters were fixed on a small piloting subset disjoint from the 134 evaluation tasks.

Compute and runtime. ✓ Full compute breakdown in App. I: $\sim$ 240 GPU-hours, $\sim 4.6 \times 10^5$ GPT-5 calls, $\sim 5.2 \times 10^5$ Qwen-3-8B calls.

Statistical reporting. ✓ All accuracy rows show mean $\pm$ sample σ with n=10. Pairwise comparisons use Welch's two-sample t-test with degrees of freedom and p-values reported (App. G).

Licenses. ✓ ALFWorld (MIT), TextWorld (MIT), OSWorld (Apache 2.0), LATS / ToT / Self-Refine (research). Code release license: Apache 2.0.

Environment dependencies. ✓ `requirements.txt` in the repository; ALFWorld 0.3.3, TextWorld $\geq$ 1.5.0, OSWorld 0.1, OpenAI SDK 1.x, Transformers 4.40+.

Negative results. ✓ Documented in App. N (averaging gradients failed; mean-below trigger over-fired; memory injection into decomposition collapsed performance from 78% to 0%).

Failure modes. ✓ Three categories quantified (Sec. 7): world-knowledge (21/33), receptacle-navigation (8/33), evaluator hallucination (4/33); concrete traces in App. D and App. F.

Computational claims. Section, table, or appendix supporting each numeric claim:

- 88.1% on GPT-5: Tab. 2, App. C.
- 75.4% on Qwen-3-8B: Tab. 2, App. C.
- +12.0pp synergy on GPT-5: Tab. 2, computed as combined − Reflexion-only − TextGrad-only.
- p<0.01 vs LATS: App. G, Welch's $t \approx 2.87$, df ≈ 18, $p \approx 0.010$.
- 84.3% worst-case sensitivity: Tab. 4, m=3 row.
- Evaluator hallucination $\sim$ 3%: Sec. 7, computed as $\sim 4/33$ failures attributable to false-positive scoring.

R. Practitioner decision checklist

We translate Sec. 3.7's informal C1/C2 conditions into a practitioner-facing checklist for deciding whether to add a slow process to an existing fast-process LLM-agent baseline.

When DPR-style routing is likely to help. If your agent has all four of the following, the synergy gain is likely $\geq$ 5pp:

1. **Repeated low-progress streaks.** On your benchmark, $\geq$ 10% of failed episodes contain a sequence of 3+ consecutive steps where progress stalls and the agent retries minor variations.
2. **A natural-language policy.** Your agent's behavior is conditioned on a string (system prompt, instruction prefix, or dynamically updated policy), not a fixed prompt template.
3. **A cheap progress signal.** A per-step LLM call ($\sim$ 1 small-model invocation) can reliably score whether the last action moved the agent toward the goal. If your task is too open-ended for any LLM judge, the routing signal is missing.
4. **Strategic alternatives exist.** The benchmark contains tasks where stalled fast-process behavior could be rescued by a high-level reorientation ("the wrong receptacle was tried"), not just retrying the same approach.

When DPR-style routing is unlikely to help.

- *Tasks dominated by missing world knowledge.* If recovery requires information not derivable from the trajectory and not in the model's pretraining (e.g., an unmapped corrective receptacle), the slow process cannot help in budget. Heat/Examine in our results are this regime.
- *Single-step tasks.* If episode length is 1-2 steps, neither the gradient cadence nor the slow trigger has room to fire.
- *Tasks with unreliable progress signals.* If $\eta_{\text{fp}} \gtrsim 0.10$, the routing rule mis-fires often enough that the union bound on false triggers becomes loose; the slow process's expected value drops.
- *Tasks where strategic recovery and tactical refinement are the same operation.* If every failure can be addressed by the next gradient, the dual-process distinction collapses.

What you can expect numerically. Based on the cross-model decomposition (Tab. 2):

- Adding TextGrad-style fast refinement to a zero-shot baseline buys $\sim$ 23-26pp on suitable text-game benchmarks.

- Adding Reflexion-style slow diagnosis on top buys an additional $\sim$ 7-12pp ($+$12pp on GPT-5, $+$7pp on Qwen-3-8B).
- The slow-process additional cost is ~ 5 extra LLM calls per slow activation, or ~ 0.75 additional calls per episode at the empirical $\sim 15\%$ activation rate.
- Cost-per-percentage-point of the combined system is ~ 1.4 calls/pp (Tab. 6), favorable relative to either component alone.