Vivid Memories: FINE-GRAINED TRAJECTORY MEMORY FOR ON-POLICY AGENT SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory is central to self-improving language agents, but most agent memory systems compress experience into skills, procedures, or reflections. This is useful for inference-time prompting, but it discards the step-level state–action evidence needed for training-time credit assignment. We introduce VIVID, an on-policy self-improvement method that treats memory as a *training-time supervision bank* rather than an inference-time prompt aid. VIVID stores both successful and failed rollouts as raw turn sequences, annotates each turn with an offline outcome label and a continuous Q value, retrieves a balanced mixture of state-matched successful and failed precedents during training rollouts, and uses the annotations for *reward-consistent turn-level advantage shaping* of GRPO: the direction of every update is inherited from the verified environment reward, and the judge modulates only its magnitude. Evaluation is retrieval-free: the trained policy uses no memory, retrieval, or judge. On AppWorld, ALFWorld, and Sokoban with Qwen3-1.7B, Qwen3-4B, and Llama-3.1-8B backbones, VIVID is best on every benchmark, metric, and backbone, improving Qwen3-4B average accuracy from 52.5 (GRPO) and 64.1 (Skill-SD) to 72.0 ($\geq$3 seeds, std $\approx$1.0). Controlled analyses locate the gain in the *content* of per-turn labels rather than in reweighting machinery or external-LLM spend: uninformative-label controls collapse to the GRPO floor, judge-free retrieval already matches Skill-SD with zero external LLM, budget-matched baselines do not close the gap, a future-blind (causal) judge retains most of the improvement, and letting the judge set the update sign is measurably worse than magnitude-only shaping.

1 INTRODUCTION

Large language model (LLM) agents learn from trajectories, but most agent memory systems compress experience into skills, procedures, reflections, or long-term memory entries for later retrieval (Park et al., 2023; Wang et al., 2023; Shinn et al., 2023; Zhao et al., 2024; Zhong et al., 2024; Packer et al., 2023; Fang et al., 2026; Wang et al., 2024b). This is the right design for inference-time prompting, where memory must fit into a limited context, but a poor fit for *training*, which requires credit assignment over local decisions: a failed trajectory may contain many correct setup actions, one critical error, and several recovery attempts, while a successful trajectory may still include redundant turns. Once experience is compressed into a skill or a one-line reflection, the state–action evidence needed to decide which turns to reinforce, suppress, or treat as recoveries is gone.

Standard on-policy objectives do not recover this structure: GRPO assigns a single rollout-level advantage to every token of a trajectory (Shao et al., 2024), so multi-turn credit assignment is coarse. Agent self-distillation methods add memory- or skill-conditioned supervision but still compress experience into compact skills (Wang et al., 2026), and turn-level credit methods learn a per-turn critic (Zhou et al., 2024) or exploit anchor-state structure within the current batch (Feng et al., 2026); none persists experience across iterations as a retrievable, annotated bank.

We introduce VIVID, an on-policy self-improvement method that treats memory as a *training-time supervision bank* (Figure 1). VIVID stores successful and failed trajectories as raw turn sequences, annotates each turn with an offline outcome label and continuous Q value, retrieves a balanced mixture of state-matched successful and failed precedents during rollout, and converts the rollout-level GRPO advantage into a turn-weighted advantage. We describe this objective precisely as *reward-consistent*

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

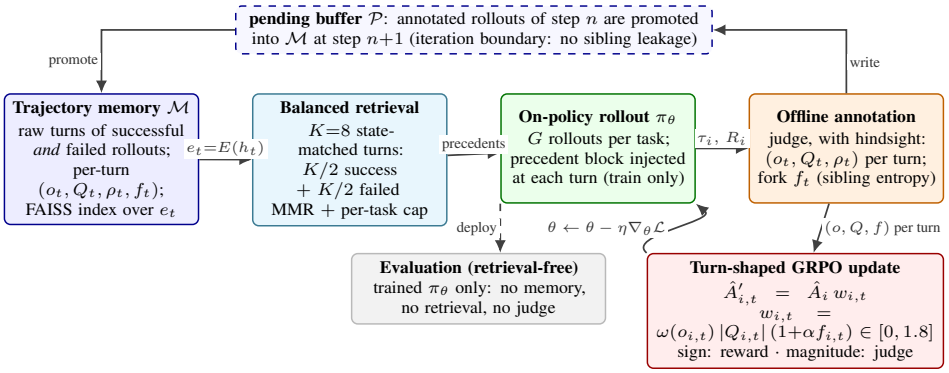

Figure 1: **VIVID training loop.** At step n, each turn of an on-policy rollout is conditioned on a balanced block of K state-matched precedents ($K/2$ successful, $K/2$ failed) from the memory store $\mathcal{M}$. Completed rollouts are annotated offline: an LLM judge assigns each turn an outcome label o_t, value Q_t, and reflection ρ_t with hindsight, and a fork score f_t is computed from sibling rollouts. The annotations shape the GRPO advantage per turn in the *reward-consistent* direction ($w_{i,t} \geq 0$), so the update sign always comes from the verified reward. Annotated rollouts become retrievable only at step $n{+}1$. Evaluation is retrieval-free: the deployed policy uses no memory, retrieval, or judge.

turn-level advantage shaping: the turn weight is non-negative, so the direction of every update is inherited from the verified environment reward, and the judge—whose per-turn agreement with humans is substantial but imperfect (Cohen's $\kappa \geq 0.6$)—controls only a bounded magnitude. The asymmetry is deliberate: VIVID has no learned per-turn value baseline that would license per-turn sign flips, and a signed variant in which the judge can flip the update direction is measurably worse (Section 5.4).

VIVID differs from compressed-memory methods in three *design inversions*: contrastive storage of both successful and failed trajectories with balanced retrieval; raw step-annotated trajectories rather than task-level skills; and step-level outcome labels rather than trajectory-level success or failure. The unit of supervision—an annotated turn inside a multi-turn trajectory, persisted as retrievable memory—is the contribution; the LLM judge is a convenience for producing it, and a judge-free configuration already matches the strongest compressed-memory baseline. Memory, retrieval, and the judge are used only during training.

On AppWorld, ALFWorld, and Sokoban (Trivedi et al., 2024; Shridhar et al., 2020; Schrader, 2018), covering API-based interaction, embodied instruction following, and symbolic planning, VIVID is best on every benchmark, metric, and backbone (Qwen3-1.7B, Qwen3-4B, Llama-3.1-8B), improving Qwen3-4B average accuracy from 52.5 (Vanilla GRPO) and 64.1 (Skill-SD, re-implemented under an identical protocol) to 72.0, with a +7–9 margin over Skill-SD on every backbone ($\geq$3 seeds, std $\approx$1.0). Controlled analyses then isolate *where* the gain comes from: a $2{\times}2$ decomposition of the channels through which judge labels reach the policy, uninformative-content controls, a future-blind causal judge, budget-matched baselines, a signed-Q variant with sign-alignment statistics, a retrieval-ratio sweep, and a cross-backbone replication of the ablations with seed-level uncertainty.

2 Related Work

Memory for language agents. Generative Agents, MemoryBank, and MemGPT store summaries, observations, or records for later retrieval (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023); Memp, Voyager, and Agent Workflow Memory compress experience into procedures, executable skills, or workflows (Fang et al., 2026; Wang et al., 2023; 2024b); Reflexion and ExpeL distill failures into verbal reflections or insights (Shinn et al., 2023; Zhao et al., 2024). These are inference-time memories consumed by prompting, and compression is what makes them fit. VIVID targets training-time memory: raw trajectories with step-level annotations, including failures, and a trained policy that carries no memory at deployment.

On-policy agent training and credit assignment. GRPO (Shao et al., 2024) and DAPO (Yu et al., 2026) assign one advantage per trajectory. Multi-turn agent RL has addressed this with a learned turn-level critic (ArCHer; Zhou et al., 2024), stability analyses (RAGEN; Wang et al., 2025), and in-batch anchor-state grouping (GiGPO; Feng et al., 2026). On-policy distillation learns from self-generated samples under a teacher signal (Agarwal et al., 2024); Skill-SD, the closest method to ours, improves multi-turn agents through skill-conditioned self-distillation with a compact skill memory (Wang et al., 2026), and SSD shows that diversity in self-generated samples helps self-distillation (Zhang et al., 2026). VIVID differs on the source of the per-turn signal, persistent cross-iteration experience with hindsight annotations retrieved at the state level rather than a learned critic or in-batch coincidences, and consumes it through bounded, reward-consistent shaping rather than replacing the advantage.

LLM feedback and process supervision. Constitutional AI and RLAIF replace human preference labels with model labels (Bai et al., 2022; Lee et al., 2023), LLM-as-a-judge evaluates responses (Zheng et al., 2023), and process reward models score intermediate reasoning steps (Lightman et al., 2024; Wang et al., 2024a). These score a response or a reasoning step of a single-turn generation. VIVID assigns credit to turns inside interactive multi-turn trajectories, persists it as a bank that also conditions future rollouts, and deliberately discards the judge's sign; Section 5.3 shows that the gain tracks the content of per-turn labels, not the presence of an external model.

3 Method

3.1 Problem Setup

VIVID stores trajectories uncompressed and annotates each turn with step-level outcome information; the memory conditions rollouts and shapes advantages, both at training time. We consider a multi-turn task x with horizon H. A rollout $\tau_i = (u_{i,0}, \ldots, u_{i,T_i-1})$ consists of turns, each containing the agent's thought, executable action, and the environment observation returned after the action. The policy π_θ generates action tokens autoregressively from the task prompt and interaction history, and a completed rollout receives a verified scalar reward $R_i = R(x, \tau_i) \in [0, 1]$ (in AppWorld, the fraction of task-specific unit tests passed). For each task we sample G rollouts and compute the group-relative advantage of GRPO (Shao et al., 2024),

$$\hat{A}_i = \frac{R_i - \frac{1}{G}\sum_{j=1}^{G} R_j}{\mathrm{std}(R_1, \ldots, R_G) + \epsilon}, \tag{1}$$

which is initially shared by all action tokens of τ_i; VIVID converts it into a turn-specific shaped advantage.

3.2 Uncompressed Trajectory Memory

VIVID maintains a memory store $\mathcal{M}$ whose items are complete trajectories with turn annotations,

$$m = \big(x, \tau, R, \mathrm{outcome}_{\mathrm{final}}, \{a_t\}_{t=0}^{T-1}\big), \qquad a_t = \big(h_t, e_t, o_t, Q_t, \rho_t, f_t, \mathcal{A}_t^{\mathrm{alt}}\big), \tag{2}$$

where h_t is the pre-action history, e_t its embedding, o_t a categorical outcome label, $Q_t \in [-1, 1]$ a continuous turn value, ρ_t a short diagnostic reflection, f_t a fork score, and $\mathcal{A}_t^{\mathrm{alt}}$ the set of distinct sibling actions at the same turn index. The raw turn sequence is stored verbatim, preserving the exact state, action, and feedback that produced a success or failure; the same action can be useful in one state and harmful in another, which a compressed skill cannot express.

State embedding, index, and iteration boundary. The state at turn t is the text rendering $h_t = \mathrm{render}(u_0, \ldots, u_{t-1})$, encoded with a frozen `all-MiniLM-L6-v2` encoder (Wang et al., 2020) into an ℓ_2-normalized 384-dimensional vector stored in a FAISS inner-product index (Douze et al., 2025; Johnson et al., 2019), so that search equals cosine similarity; freezing the encoder decouples retrieval from policy optimization. At training step n, VIVID first promotes items from a pending buffer $\mathcal{P}$ into $\mathcal{M}$, then collects rollouts with the current memory, and only afterwards writes the newly annotated trajectories into $\mathcal{P}$: rollouts at step n can retrieve trajectories from steps $< n$ but never their own siblings, which prevents circular supervision and follows Skill-SD's iteration-boundary discipline (Wang et al., 2026).

3.3 Offline Turn Annotation

After a rollout completes, an LLM judge annotates each turn given the task prompt, the full trajectory, the final reward, and the terminal outcome, emitting (o_t, Q_t, ρ_t) with

$$o_t \in \mathcal{O} = \{\text{PROG}, \text{NEUT}, \text{REC}, \text{ERR}, \text{RED}\}, \tag{3}$$

where, throughout the paper, PROG = PROGRESS, REC = RECOVERY, ERR = CRITICAL_ERROR, NEUT = NEUTRAL, and RED = REDUNDANT. Q_t is calibrated so that positive values indicate useful actions, negative values harmful actions, and values near zero neutral or uncertain turns (prompt in Appendix F). The judge is intentionally *offline*: it sees the future trajectory and final reward when labeling earlier turns, which provides hindsight credit assignment (a locally reasonable turn that later leads to a dead end should receive a low value); it never generates actions and is used only at training time. We treat hindsight as an explicit, ablated design axis: Section 5.3 reports a *causal* judge that sees only the prefix up to the labeled turn.

Fork score. VIVID also computes a programmatic fork score from sibling rollouts: for each task and turn index, the normalized entropy of the empirical distribution of *distinct* sibling actions across the G rollouts,

$$f_t = H\big(\{a_{i,t}\}_{i=1}^{G}\big) / \log G \in [0, 1]. \tag{4}$$

Actions are canonicalized at the schema level before counting—API/tool name plus canonicalized argument structure (argument keys and value types or identifiers, with formatting and ordering normalized), not raw token strings—so that semantically identical calls that differ only in surface form are grouped. High f_t marks states where the policy explored several distinct actions, which are precisely the states where credit assignment is most valuable (Zhang et al., 2026).

3.4 Balanced Turn-Level Retrieval

Before every agent turn during rollout collection, VIVID computes $e_t = E(h_t)$ and retrieves K annotated turns with a balanced retriever under three constraints. First, retrieval targets an equal split between turns from successful trajectories and turns from failed or partially successful ones, exposing the policy to both behaviors to imitate and behaviors to avoid (Section 5.1 sweeps this ratio). Second, it applies maximal marginal relevance (Carbonell & Goldstein, 1998): for a candidate c and the already selected set $\mathcal{S}$,

$$\text{MMR}(c) = \lambda_{\text{M}} \,\text{sim}(c, e_t) - (1 - \lambda_{\text{M}}) \max_{c' \in \mathcal{S}} \text{sim}(c, c'). \tag{5}$$

Third, it caps the number of precedents from any single source task. Each precedent contains the local history around the stored turn, the action taken, the judge outcome label and reflection, and any alternative sibling actions; failed precedents are marked as patterns to avoid. The block is inserted into the prompt for the current turn only and refreshed at the next turn (Appendix F). Because the PPO datum stores the exact prompt tokens used during sampling, the policy is trained on actions generated under the same retrieved context. Unlike Skill-SD, where compact skills condition a teacher and the student learns through reverse-KL distillation (Wang et al., 2026), retrieval here shapes the on-policy data distribution and turn-level annotations shape the policy gradient, both at training time only.

3.5 Reward-Consistent Turn-Level Advantage Shaping

The central training signal is a turn-specific weight that shapes the GRPO advantage. For trajectory i and turn t,

$$w_{i,t} = \omega(o_{i,t})\,|Q_{i,t}|\,(1 + \alpha f_{i,t}), \qquad \hat{A}'_{i,t} = \hat{A}_i\, w_{i,t}, \tag{6}$$

where $|Q_{i,t}|$ (clipped to $[0, 1]$) acts as a confidence gate, $\alpha = 0.2$, and

$$\omega(\text{PROG}) = 1.0, \quad \omega(\text{ERR}) = 1.5, \quad \omega(\text{REC}) = 1.3, \quad \omega(\text{NEUT}) = 0.5, \quad \omega(\text{RED}) = 0.3, \tag{7}$$

so that $w_{i,t} \in [0, 1.8]$. These values were fixed a priori and never tuned on an evaluation split (Appendix C.5 gives a post-hoc sensitivity analysis); Equation (6) is the single definition of the weight used throughout the paper and the code. We optimize the clipped surrogate (Schulman et al., 2017)

$$\mathcal{L}_{\text{VIVID}}(\theta) = -\frac{1}{N} \sum_{i,k} \min\Big(r_{i,k}\,\hat{A}'_{i,t(k)},\ \text{clip}(r_{i,k}, 1 - \epsilon_l, 1 + \epsilon_h)\,\hat{A}'_{i,t(k)}\Big), \tag{8}$$

where $r_{i,k} = \pi_\theta(y_{i,k} \mid x, y_{i,<k}) / \mathrm{sg}[\pi_{\theta_{\text{old}}}(y_{i,k} \mid x, y_{i,<k})]$ is the token importance ratio, k indexes action tokens, $t(k)$ maps a token to its turn, N is the number of action tokens, and $\epsilon_l = 0.2$, $\epsilon_h = 0.28$ follow the Clip-Higher motivation of DAPO (Yu et al., 2026). The objective is a drop-in modification of GRPO: critical errors and recoveries receive stronger updates, uncertain or neutral turns weaker updates, and high-fork states a mild additional emphasis.

Why magnitude, not sign. Every factor in $w_{i,t}$ is non-negative, so $\mathrm{sign}(\hat{A}'_{i,t}) = \mathrm{sign}(\hat{A}_i)$: the update direction is inherited from the verified reward and the judge modulates only its magnitude, by a bounded multiplier. The judge's signed $Q_{i,t}$ could flip the per-turn direction; we discard its sign on purpose. VIVID has no learned per-turn value baseline (the GAE-style construction (Schulman et al., 2015) that would license sound per-turn sign flips), so sign and magnitude errors are not symmetric risks: a sign error points the update *away* from the verified reward, whereas a magnitude error rescales a correct-direction update by at most $1.8\times$. Section 5.4 confirms this empirically: a signed-Q variant is significantly worse, and the adverse configuration (a CRITICAL_ERROR turn in a positive-advantage rollout) occurs on only 4.0% of positive-rollout turns. The remaining case, useful PROGRESS or RECOVERY turns inside failed rollouts, is handled by balanced retrieval, which surfaces them as *positive* precedents: good local decisions are reinforced and bad ones suppressed by two coordinated mechanisms, neither of which cedes the gradient sign to a noisy annotator.

Training loop and evaluation protocol. Algorithm 1 (Appendix B) summarizes one iteration: promote the pending buffer, roll out G trajectories per task with balanced retrieval at each turn, annotate offline, compute fork scores, shape advantages, update θ, and write annotated rollouts into the pending buffer. **Evaluation is retrieval-free.** At test time VIVID injects no precedents, queries no memory, and calls no judge; every reported number, for VIVID and every baseline, is produced by the trained policy alone under the same decoding protocol, so the comparison with Vanilla GRPO is strictly fair at test time and store growth cannot affect a deployed agent.

4 Experiments

4.1 Experimental Setup

Benchmarks and models. We evaluate on AppWorld (Trivedi et al., 2024) (multi-step operations over structured app states through APIs), ALFWorld (Shridhar et al., 2020) (text-based embodied instruction following), and Sokoban (Schrader, 2018) (long-horizon planning with irreversible moves); splits and curricula follow Skill-SD (Appendix A). Backbones are Qwen3-1.7B, Qwen3-4B (Yang et al., 2025), and Llama-3.1-8B (Grattafiori et al., 2024); Qwen3-4B is the main backbone for analyses and the design-inversion ablations are replicated on Llama-3.1-8B.

Baselines and protocol parity. **Base Model** is the instruction-tuned model without agent training; **Vanilla OPD** is on-policy distillation (Agarwal et al., 2024) without the turn-weighted memory mechanism; **Vanilla GRPO** uses the standard group-relative objective; **Skill-Aug GRPO** augments GRPO with retrieved skill-like information but no step-level annotations; **Skill-SD** (Wang et al., 2026) is the strongest compressed-memory baseline and itself uses an external LLM teacher for skill extraction. Skill-SD is *re-implemented and re-run by us*, not copied from its paper. Every method is trained and evaluated under the identical protocol of Table 5 (same backbone and LoRA configuration, rollout budget $G = 4$, 200 optimization steps, optimizer and clipping, splits and Sokoban curriculum, retrieval-free evaluation), so no method receives more gradient steps, rollouts, context, or evaluation budget than another; VIVID's hyperparameters were fixed a priori. Where VIVID spends more on external supervision (the judge), Section 5.3 adds budget-matched baselines.

Metrics and uncertainty. **Acc** is pass@1 task success and **Comp** a completion score with partial credit for progress toward the goal; we also report averages over the three benchmarks. All numbers are means over $\geq$3 training seeds; the per-cell seed standard deviation is $\approx$1.0 pass@1 throughout Tables 1–4, so a two-sample difference has SE $\approx$1.4, and we describe effects in units of this SE (σ; conservative unpaired uncertainty, reported at the table level because it is homogeneous across cells). **Evaluation is retrieval-free** for every method: greedy decoding, the same horizon, no precedents, memory, or judge, so Table 1 compares training procedures, not inference-time augmentations.

Backbone	Method	AppWorld		ALFWorld		Sokoban		Average	
		Acc	Comp	Acc	Comp	Acc	Comp	Acc	Comp
Qwen3-1.7B	Base Model	5.5	28.0	6.0	25.0	8.0	22.0	6.5	25.0
	Vanilla OPD	16.0	45.0	18.0	38.0	14.0	28.0	16.0	37.0
	Vanilla GRPO	36.0	62.0	38.0	60.0	38.0	58.0	37.3	60.0
	Skill-Aug GRPO	28.0	60.0	25.0	50.0	12.0	25.0	21.7	45.0
	Skill-SD	50.0	72.0	50.0	70.0	47.0	62.0	49.0	68.0
	VIVID (ours)	**58.0**	**77.0**	**57.0**	**75.0**	**54.5**	**67.5**	**56.5**	**73.2**
Qwen3-4B	Base Model	8.8	39.1	10.0	35.0	12.5	32.0	10.4	35.4
	Vanilla OPD	22.8	59.7	25.0	50.0	21.9	37.5	23.2	49.1
	Vanilla GRPO	50.9	76.3	55.0	75.0	51.6	68.8	52.5	73.4
	Skill-Aug GRPO	42.1	76.1	40.0	65.0	20.3	37.5	34.1	59.5
	Skill-SD	64.9	84.9	65.0	82.0	62.5	71.1	64.1	79.3
	VIVID (ours)	**74.0**	**90.0**	**72.0**	**87.0**	**70.0**	**77.0**	**72.0**	**84.7**
Llama-3.1-8B	Base Model	12.0	42.0	12.0	36.0	14.0	33.0	12.7	37.0
	Vanilla OPD	28.0	60.0	28.0	50.0	24.0	38.0	26.7	49.3
	Vanilla GRPO	55.0	78.0	58.0	76.0	54.0	70.0	55.7	74.7
	Skill-Aug GRPO	45.0	72.0	42.0	65.0	22.0	38.0	36.3	58.3
	Skill-SD	68.0	85.0	68.0	83.0	65.0	73.0	67.0	80.3
	VIVID (ours)	**76.0**	**91.0**	**75.0**	**88.0**	**73.0**	**79.0**	**74.7**	**86.0**

Table 1: **Main results** across three backbones. Acc is pass@1 and Comp is completion score; all methods are evaluated retrieval-free under the identical protocol of Table 5, and Skill-SD is our re-implementation run under the same budget. Entries are means over $\geq$3 seeds with seed std $\approx$1.0 (two-sample SE $\approx$1.4), so the +7–9 average-accuracy margins over Skill-SD are ≈ 5–6σ on every backbone.

4.2 Main Results

Table 1 reports the main results (visualized in Appendix C.1). VIVID is best for every benchmark, metric, and backbone, so the gains transfer across API-based interaction, text-world control, and symbolic planning. On Qwen3-1.7B it improves average accuracy from 6.5 to 56.5 and completion from 25.0 to 73.2, outperforming Skill-SD by 7.5 and 5.2 points; on Qwen3-4B it reaches 72.0 and 84.7, +19.5 accuracy over Vanilla GRPO and +7.9 over Skill-SD; on Llama-3.1-8B it reaches 74.7 and 86.0, +7.7 and +5.7 over Skill-SD. With seed std $\approx$1.0, every VIVID–Skill-SD margin exceeds 5σ. Vanilla GRPO gives helpful setup actions, critical errors, recoveries, and redundant turns the same trajectory-level signal; VIVID adds turn-level shaping and precedent-conditioned rollouts, and neither uses memory at test time. Skill-Aug GRPO collapses on Sokoban, where injected skills cannot express local state–action constraints; Skill-SD, although much stronger and also equipped with an external LLM teacher, trails VIVID by a stable +7–9 points at every scale, which points to the structure of the supervision signal rather than model capacity. VIVID also improves steadily throughout training as the store grows and keeps a clear gap over both baselines (Appendix C.3), and it degrades more gracefully under reduced task repetition and out-of-domain evaluation (Appendix C.4).

5 Analyses

5.1 Are Failed Trajectories Useful, and How Many?

All analyses use Qwen3-4B unless stated otherwise, under the protocol and uncertainty conventions of Section 4.1.

Table 2 (left) varies the outcome composition of retrieved memory. Balanced retrieval is best on every benchmark: 72.0 average pass@1 against 66.7 for success-only, 64.7 for failure-only, and 63.7 for unbalanced Top-K. Failure-only beats Top-K, so failed trajectories are supervision rather than noise; success-only beats failure-only, so positive precedents remain necessary; combining both is

Retrieval pool	AppWorld	ALFWorld	Sokoban	Avg
Top-K (no balance)	66.0	64.0	61.0	63.7
Failure-only	67.0	65.0	62.0	64.7
Success-only	69.0	67.0	64.0	66.7
Balanced 50/50 (VIVID)	**74.0**	**72.0**	**70.0**	**72.0**

Success/failure quota (Avg pass@1)				
0/100	25/75	**50/50**	75/25	100/0
64.7	68.0	**72.0**	71.0	66.7

Table 2: **Memory outcome composition** (Qwen3-4B, pass@1). *Left:* retrieval restricted to one polarity or left unbalanced. *Right:* sweep of the success/failure quota with $K{=}8$, MMR, and the per-task cap fixed (0/100 and 100/0 are the failure-only and success-only rows). Balance is the empirical peak and the interior is flat.

		Qwen3-4B				Llama-3.1-8B	
Variant		Avg Acc	Avg Comp	ΔAcc	σ	Avg Acc	ΔAcc
VIVID (full)		**72.0**	**84.7**	–	–	**74.7**	–
w/o failed memory (success-only storage)	(I1)	66.7	80.5	-5.3	3.8	69.7	-5.0
w/o raw trajectory (compressed summaries)	(I2)	65.0	79.5	-7.0	5.0	68.2	-6.5
w/o step labels (trajectory-level outcome only)	(I3)	65.5	79.8	-6.5	4.6	68.8	-5.9
w/o balanced retrieval		65.3	79.6	-6.7	4.8	–	–
w/o retrieval (weighting only)		67.5	81.5	-4.5	3.2	–	–
w/o weighting (retrieval only)		68.8	82.6	-3.2	2.3	–	–
w/o $\lvert Q\rvert$ confidence gate		69.5	83.0	-2.5	1.8	–	–
w/o fork bonus		70.8	84.0	-1.2	0.9	–	–
ref: Skill-SD (re-run)		64.1	79.3	-7.9	5.6	67.0	-7.7

Table 3: **Component ablation** averaged over the three benchmarks. Δ is the drop in average accuracy relative to the full model and σ its size in two-sample SE units ($\approx$1.4); the design inversions are replicated on Llama-3.1-8B. Effects below $\approx 2\sigma$ are consistent in sign but not individually significant.

best, because successes provide actions to imitate and failures local patterns to avoid. Because a fixed 50/50 quota could merely inherit a task's success rate, we sweep it (right). The curve is concave and peaks at balance; both extremes and the natural distribution trail 50/50 by 5–8 points, so forcing balance beats any skewed or natural ratio, and the interior is flat (68–72 across 25/75–75/25), so the exact ratio needs no per-task tuning. The mild asymmetry (75/25 $>$ 25/75) matches success-only beating failure-only: positive precedents are slightly more valuable, but removing either polarity hurts, which is exactly the contrastive design.

5.2 Component Ablation and Cross-Backbone Replication

Removing any of the three design inversions hurts substantially and significantly ($>3.5\sigma$; Table 3): success-only storage drops average accuracy to 66.7, compressed summaries in place of raw trajectories to 65.0, and trajectory-level labels in place of step labels to 65.5, so failed trajectories carry contrastive supervision, step-level state–action evidence must be preserved, and trajectory-level rewards are too coarse. The same inversions replicated on Llama-3.1-8B cost 5.0–6.5 points in the same rank order, and every ablated variant still beats Skill-SD on both backbones, so the conclusions are not backbone-specific. Retrieval and weighting are complementary and individually effective: from the GRPO floor of 52.5, weighting alone reaches 67.5 (+15.0) and retrieval alone 68.8 (+16.3), within $\approx 1\sigma$ of each other, so we do not claim that one dominates; together they reach 72.0, and the two axes share credit by construction, since retrieval also reshapes the training distribution through the stored precedent-conditioned prompts. The weighting sub-components are smaller: the $\lvert Q\rvert$ gate is worth 2.5 points (1.8σ, marginal) and the fork bonus 1.2 (within noise); we report both as consistent in direction but not individually significant.

5.3 Where Does the Gain Come From? Channels, Content, Hindsight, and Budget

The full method injects judge-annotated precedents at rollout time *and* shapes gradients with the same annotations, and it spends an external-LLM budget that Vanilla GRPO does not; Table 4 separates these factors.

	Configuration	AppWorld	ALFWorld	Sokoban	Avg
(A)	VIVID full: tags displayed + gradient weighting	**74.0**	**72.0**	**70.0**	**72.0**
	Retrieval-only: tags displayed, no weighting	–	–	–	68.8
	Weighting-only: gradient weighting, no retrieval	–	–	–	67.5
	Judge-free retrieval: no tags, no weighting, balance on R (no LLM)	64.5	65.5	63.4	64.5
(B)	Random labels (same judge calls, content scrambled)	55.1	53.2	56.5	54.9
	Structured-uninformative w (reward-only heuristic)	53.7	54.2	55.1	54.3
	Magnitude-matched random w	52.3	53.4	54.2	53.3
(C)	Causal (future-blind) judge labels (no labels: 65.5, Table 3)	–	–	–	68.2
(D)	Skill-SD, external-LLM budget matched to VIVID	–	–	–	65.2
	Vanilla GRPO + matched external-LLM budget	–	–	–	55.7
(E)	Signed-Q weight (judge can flip the update direction)	69.6	70.2	66.5	68.8
	Additive centered judge term (instead of $\times\lvert Q\rvert$)	70.0	71.0	70.5	70.5
	ref: Skill-SD (LLM teacher, as re-run)	64.9	65.0	62.5	64.1
	ref: Vanilla GRPO	50.9	55.0	51.6	52.5

Table 4: **Sources of supervision** (Qwen3-4B, pass@1). (A) Judge labels reach the policy as outcome tags displayed in retrieved precedents and as gradient weights; the 2×2 strips each channel. (B) Controls that keep the reweighting machinery and the external-LLM calls but destroy per-turn label content. (C) A causal judge that sees only the prefix up to the labeled turn. (D) Baselines given an external-LLM budget matched to VIVID's judge (≈$20 per run). (E) Sign choice (Section 5.4). Dashes: per-benchmark breakdown not logged.

(A) Two channels, de-confounded. Judge labels reach the policy as outcome tags displayed inside retrieved precedents ("This led to: CRITICAL_ERROR. Avoid this pattern.") and as the gradient weights of Equation (6). Stripping the gradient channel gives 68.8; stripping the display channel as well, so that retrieval balances on the verified reward alone with no judge call, gives 64.5. Displayed tags are thus worth 4.3 points and gradient weighting 3.2; both channels contribute, and the I3 ablation is decomposed rather than confounded. Judge-free retrieval already matches Skill-SD (64.5 vs. 64.1) with *zero* external LLM, giving a clean ladder: GRPO 52.5 → raw balanced memory 64.5 → +judge tags 68.8 → +weighting 72.0.

(B) Content, not machinery or spend. If the judge merely re-encoded the final reward, labels with the same reward correlation but scrambled per-turn placement would match it. They do not: random labels with identical judge calls reach 54.9, a structured-uninformative weighting with matched marginals (equivalently, a reward-only heuristic that upweights late turns of failed trajectories without any judge) reaches 54.3, and magnitude-matched random weights reach 53.3. All collapse to the GRPO floor, 9–11 points below Skill-SD, while the real labels reach 72.0: the gain comes from per-turn label *content*.

(C) Hindsight vs. raw memory. A causal judge that labels turn t from the prefix up to t, with no future turns or final reward, reaches 68.2: causal labels add +2.7 over raw memory without labels (65.5) and hindsight a further +3.8. Both contributions are separable, and even the future-blind judge beats Skill-SD, so VIVID outperforms the strongest compressed-memory baseline when denied all oracle knowledge. Consistently, the labels are not a deterministic function of the outcome: $H(\text{label} \mid \text{outcome}) \approx 0.80$ on every benchmark and failed trajectories still contain PROGRESS and RECOVERY turns (Appendices C.6, C.7), and per-class Cohen's $\kappa \geq 0.6$ against 100 human-labeled turns, whose annotators also saw the outcome, certifies per-turn placement rather than outcome copying (Appendix C.8).

(D) Matched external-LLM budget. External-LLM use is not unique to VIVID: Skill-SD's skill extraction uses an LLM teacher. A budget matched to VIVID's judge lifts Skill-SD only from 64.1 to 65.2 and Vanilla GRPO from 52.5 to 55.7, both far below 72.0, so equalizing supervision spend does not close the gap and the advantage is structural.

Cost accounting. Rollout (≈23 min per step) and optimization (3 min) are shared by all methods; VIVID adds judging (5 min), memory construction (1 min), and retrieval (1.5 min), a 29% per-step overhead that comes almost entirely from the judge (≈$0.5 per step, ≈$20 per run with GPT-4.1-mini), while memory construction and retrieval are under 11% of a step and the precedent block is

fixed at $K = 8$, so prompt size does not grow with the store (Figure 5, Appendix C.3; Skill-SD's teacher calls are counted on the same footing). The overhead is not a pure tax: VIVID gives the best pass@1 for any wall-clock budget above roughly 20 hours and reaches Skill-SD's final 65% in 56 hours versus 95, a 41% reduction in time-to-target; and the judge is optional in a precise sense, since removing it still matches Skill-SD (64.5) and cheaper local judges trade off gracefully (Appendix C.9).

5.4 Sign Versus Magnitude: Should the Judge Set the Update Direction?

A natural objection to Equation (6) is that a non-negative weight changes the magnitude of an update but never its direction: a CRITICAL_ERROR turn inside a positive-advantage rollout is still reinforced, with the largest weight, and a useful RECOVERY turn inside a negative-advantage rollout is still suppressed. We tested the alternative directly (Table 4E). A signed-Q variant, identical except that the weight carries the sign of $Q_{i,t}$ so that $\hat{A}'_{i,t}$ can flip, is *worse*: 72.0 $\rightarrow$ 68.8 average accuracy (84.7 $\rightarrow$ 82.5 completion), a significant drop (two-sample $p \approx 0.02$) that is largest on Sokoban (70.0 $\rightarrow$ 66.5), where irreversible dynamics compound judge noise. The decomposition is clean: the judge's sign hurts by 3.2 points while its magnitude helps by 2.5 (72.0 vs. 69.5 without the $|Q|$ gate, Table 3). An additive variant, $\hat{A}'_{i,t} = \hat{A}_i + \beta\,(Q_{i,t} - \bar{Q}_i)$ with $\bar{Q}_i$ the mean judge value over τ_i (unbiased within a trajectory, well-defined when $\hat{A}_i = 0$), reaches 70.5 and also beats every prior baseline, so the gain is not tied to the multiplicative form; we keep $|Q|$ for its simplicity and best performance.

Sign-alignment statistics from the training logs (Appendix C.7) explain why. On non-degenerate turns ($\hat{A}_i \neq 0$), a CRITICAL_ERROR turn lands in a positive-advantage rollout only 4.0% of the time, 89.2% of CRITICAL_ERROR turns lie in negative-advantage rollouts where $\omega = 1.5$ correctly amplifies suppression, and $\mathrm{sign}(Q)$ disagrees with $\mathrm{sign}(\hat{A})$ on only 5.4% of turns; the pathology is confined to a small minority, and fixing it by handing the direction to a $\kappa \approx 0.6$ judge is a net loss. The one sizable off-diagonal case, useful turns inside failed rollouts (18.9%), is handled by balanced retrieval, which surfaces them as positive precedents; zero-variance groups ($\hat{A}_i = 0$, 11.9%) receive no update under any multiplicative scheme and cost no measurable performance, as the additive variant shows. Finally, $|Q|$ is a confidence gate rather than a fixed emphasis: in Appendix G.1, a CRITICAL_ERROR turn with $|Q| = 0.6$ receives $w \approx 1.02$, whereas a high-confidence error ($|Q| \rightarrow 1$) receives $w \rightarrow 1.8$, above any PROGRESS turn.

6 Conclusion

VIVID uses uncompressed trajectory memory as a training-time supervision bank: it stores successful and failed rollouts, annotates turns with outcome and value information, retrieves balanced precedents during training rollouts, and shapes GRPO advantages per turn in the reward-consistent direction, while deploying a standalone policy with no memory, retrieval, or judge. Across three environments and three backbones it consistently outperforms Vanilla GRPO, Skill-Aug GRPO, and Skill-SD; controlled analyses show that the gain comes from the content of per-turn labels rather than from reweighting machinery, hindsight alone, or external-LLM spend, and that trusting the judge with magnitude but not direction is both safer and better. Training-time memory should preserve the local structure of experience, so that agents learn from failure, recovery, and high-stakes decision points, not only from success.

Ethics Statement

This work trains language agents in simulated environments (AppWorld, ALFWorld, Sokoban) and involves no human subjects beyond the hand-labeling of 100 turns used to validate the judge. Because VIVID stores failed trajectories, the training-time memory can contain incorrect or undesirable actions; the retriever marks such precedents as patterns to avoid, turn-level shaping never reinforces them against the verified reward, and the deployed policy carries no memory. Appendix E discusses propagation of harmful behavior, judge bias, and compute accessibility in more detail. We see no conflicts of interest or legal-compliance concerns.

Reproducibility Statement

All training and evaluation settings, identical across methods, are listed in Appendix A (Table 5); the training loop is given as Algorithm 1; the judge prompt, few-shot examples, and precedent-block format are reproduced verbatim in Appendix F; benchmark splits and the Sokoban curriculum follow Skill-SD and are stated in Appendix A. Hyperparameters were fixed a priori (Section 3.5), all results average $\geq$3 seeds (Section 4.1), and Skill-SD is our re-implementation run under the same protocol. We will release the training code, the memory-store schema, the judge-annotation pipeline, per-seed logs for every table, and the human-labeled judge-validation set.

AI Use Statement

In this work we used generative AI tools to help revise portions of the manuscript for clarity, grammar, and style, and to assist with code used for generating figures and tables. We did not use generative AI tools to generate research ideas, to design the method or experiments, or to produce or interpret experimental results, and the remaining tasks with required disclosure are not applicable to this work. All AI-assisted text was reviewed and edited by the authors, and AI-assisted plotting code was checked against the underlying logs by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI. Separately, an LLM (GPT-4.1-mini) is a *component of the method*, used as an offline judge for turn annotation as described in Section 3.3; its role, cost, and the controls that bound its influence are reported in Section 5.3.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Jaime G Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, volume 98, pp. 290941–291025, 1998.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. *IEEE Transactions on Big Data*, 2025.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17490–17502, 2026.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

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.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE transactions on big data*, 7(3):535–547, 2019.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*, pp. 1–22, 2023.

Max-Philipp B. Schrader. gym-sokoban. https://github.com/mpSchrader/gym-sokoban, 2018.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, et al. Skill-sd: Skill-conditioned self-distillation for multi-turn llm agents. *arXiv preprint arXiv:2604.10674*, 2026.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024a.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 2026.

Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, and Yizhe Zhang. Embarrassingly simple self-distillation improves code generation. *arXiv preprint arXiv:2604.01193*, 2026.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 19724–19731, 2024.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *arXiv preprint arXiv:2402.19446*, 2024.

A Implementation Details

Models and training backend. We use Qwen3-4B-Instruct-2507 (Yang et al., 2025) as the main backbone for all analyses, following Skill-SD (Wang et al., 2026) for direct comparability, and additionally evaluate Qwen3-1.7B-Instruct-2507 and Llama-3.1-8B-Instruct (Grattafiori et al., 2024) (Table 1); the three design-inversion ablations are replicated on Llama-3.1-8B (Table 3). All models are trained with LoRA (Hu et al., 2021) of rank 32 applied to all attention and MLP projections. Training uses the Tinker asynchronous RL API with `loss_fn="ppo"`, and rollout sampling uses the Tinker sampling client with the latest LoRA checkpoint at each step.

Table 5: **Per-method training and evaluation protocol.** Every value is identical for VIVID and all baselines; Skill-SD is our re-implementation run under this protocol.

Setting	Value (all methods)
Base model	Qwen3-4B-Instruct-2507; also Qwen3-1.7B-Instruct-2507, Llama-3.1-8B-Instruct
LoRA	rank 32, all attention + MLP projections
Rollouts per task / sampling temperature	$G = 4$ / $T_{\text{sample}} = 1.0$
Training horizon / max new tokens	$H = 10$ turns / 1024 tokens per turn
Optimization steps	200
Optimizer	AdamW, $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, lr 10^{-6}
Clipping (DAPO)	$\epsilon_l = 0.2$, $\epsilon_h = 0.28$
AppWorld split	90 train / 57 dev tasks
ALFWorld	standard train/eval splits, 50-turn limit
Sokoban	6×6, two boxes; train depths 15/20/25 (30/50/20%); test 64 levels, depth 25
Evaluation	greedy ($T_{\text{eval}} = 0$), $H_{\text{eval}} = 40$, **retrieval-free** (no memory/retrieval/judge)
Seeds	≥3 per cell; seed std ≈1.0 pass@1
VIVID-only settings	$K = 8$ (4 success / 4 failed), $\lambda_{\text{M}} = 0.7$, ≤2 precedents per source task, ω as in Eq. (7), $\alpha = 0.2$, judge GPT-4.1-mini; all fixed a priori

Rollout and optimization. For each task we sample $G = 4$ trajectories at temperature $T_{\text{sample}} = 1.0$, with a maximum horizon of $H = 10$ turns and 1024 max new tokens per turn. Cross-task rollouts are parallelized with an `asyncio.Semaphore` of 8, exploiting AppWorld's per-task state isolation; the G samples for the same task are generated sequentially to avoid environment-state interference. We optimize with GRPO (Shao et al., 2024) and DAPO-style asymmetric clipping (Yu et al., 2026), $\epsilon_l = 0.2$ and $\epsilon_h = 0.28$. The optimizer is AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, and learning rate 10^{-6}. Each training step runs one `forward_backward_async` call followed by `optim_step_async`; we save sampler-eligible weights via `save_weights_for_sampler` and resumable state via `save_state`. Main runs use 200 training steps.

Memory and retrieval. State embeddings are computed with a frozen `all-MiniLM-L6-v2` encoder (Wang et al., 2020), producing 384-dimensional representations that we ℓ_2-normalize and index with FAISS (Johnson et al., 2019; Douze et al., 2025) (exact inner-product search). At each rollout turn t we encode the textual history h_t and retrieve $K = 8$ precedents with the balanced retriever, which targets a 50/50 split between successful and failed or partial trajectories, applies MMR diversity with $\lambda_{\text{M}} = 0.7$, and allows at most two precedents from the same source task. The retrieved precedents are prepended as a user-role message to the sampling prompt for the current turn only; they are not appended to the running chat history. We prevent self-retrieval by excluding the current `trajectory_id`. To avoid sibling leakage, memory items produced at step n are first stored in a `PendingBuffer` and become retrievable only at step $n + 1$. Precedent retrieval is used during training rollouts only; evaluation never queries the store.

LLM judge. After rollouts complete, the judge annotates each trajectory with one batched call per trajectory. The output follows a strict JSON schema containing a 5-way outcome label, a bounded $Q \in [-1, 1]$, and a 15–25 word reflection for each turn. The judge sees the task prompt, the full trajectory, and the final reward (the causal variant of Section 5.3 sees only the prefix up to the labeled turn). We use GPT-4.1-mini as the default judge, dispatch calls asynchronously with an `asyncio.Semaphore` of 10, and retry each call up to three times before skipping the trajectory; skipped trajectories are excluded from memory. The total judge cost for a 200-step run is approximately $20 (≈$0.5 per step). We validate judge quality on 100 hand-labeled turns and report Cohen's $\kappa \geq 0.6$ for every outcome category (Appendix C.8).

Per-turn weighting. The per-turn weight and shaped advantage are exactly those of Equation (6), $w_{i,t} = \omega(o_{i,t})\,|Q_{i,t}|\,(1 + \alpha f_{i,t})$ and $\hat{A}'_{i,t} = \hat{A}_i\, w_{i,t}$, with the outcome weights of Equation (7) and $\alpha = 0.2$; no other weighting rule is used anywhere. The fork score is the normalized Shannon entropy of schema-canonicalized sibling actions at the same turn position, $H(\text{sibling actions})/\log G$ (Section 3.3). We clip $|Q_{i,t}|$ to $[0, 1]$ to guard against occasional judge drift, so $w_{i,t} \in [0, 1.8]$. The shaped advantage is passed to the clipped surrogate of Equation (8).

Baselines. Skill-SD is re-implemented from its description and run under the protocol of Table 5: same backbones, LoRA setup, rollout budget, optimization steps, optimizer, clipping, splits, Sokoban curriculum, and retrieval-free evaluation. Its skill-extraction teacher calls are included in its wall-clock accounting (Figure 5, the "memory build / summarizer" stage). For the budget-matched rows of Table 4, Skill-SD and Vanilla GRPO are each given an external-LLM call budget equal to VIVID's judge budget (≈$20 per run), spent through their own external-LLM interface (Skill-SD's skill-extraction teacher, and LLM feedback without per-turn structure for GRPO).

Hyperparameter selection. ω, α, K, λ_{M}, and the per-task cap were fixed a priori and were not tuned on any evaluation split. The sensitivity analysis of Appendix C.5 is a post-hoc robustness check around these defaults, not a selection procedure, so there is no dev-set selection on the 57-task AppWorld dev set.

B TRAINING ALGORITHM

Algorithm 1 VIVID training with reward-consistent turn-level advantage shaping

Input: policy π_θ, memory store $\mathcal{M}$, pending buffer $\mathcal{P}$, embedder E, judge J, environment $\mathcal{E}$
Hyperparameters: group size G, horizon H, retrieval size K, MMR coefficient λ_{M}, fork coefficient α, outcome weights ω
for training step $n = 1, \ldots, N_{\text{steps}}$ **do**
  Promote all items in $\mathcal{P}$ into $\mathcal{M}$ and clear $\mathcal{P}$ ▷ iteration boundary
  for each task x in the batch **do**
    for sample $i = 1, \ldots, G$ **do**
      Initialize interaction history $h_{i,0}$
      for turn $t = 0, \ldots, H-1$ **do**
        if $\mathcal{M}$ is not empty **then**
          $e_{i,t} \leftarrow E(h_{i,t})$; retrieve K precedents $P_{i,t}$ (balanced MMR, Sec. 3.4)
          Build the sampling prompt from $h_{i,t}$ and $P_{i,t}$ (this turn only)
        else
          Build the sampling prompt from $h_{i,t}$
        end if
        Sample thought and action from π_θ; execute in $\mathcal{E}$; append observation to $h_{i,t+1}$
        if $\mathcal{E}$ terminates **then**
          break
        end if
      end for
      Record rollout τ_i and verified reward R_i
    end for
    Compute group-relative advantages $\{\hat{A}_i\}_{i=1}^{G}$ (Eq. 1)
    Compute fork scores $\{f_{i,t}\}$ from canonicalized sibling-action entropy (Eq. 4)
  end for
  for each collected rollout τ_i **do**
    Annotate all turns with J (offline, hindsight): $\{(o_{i,t}, Q_{i,t}, \rho_{i,t})\}_t$
  end for
  for each turn (i, t) **do**
    $w_{i,t} \leftarrow \omega(o_{i,t})\,|Q_{i,t}|\,(1 + \alpha f_{i,t})$; $\hat{A}'_{i,t} \leftarrow \hat{A}_i\, w_{i,t}$ ▷ $\mathrm{sign}(\hat{A}'_{i,t}) = \mathrm{sign}(\hat{A}_i)$
  end for
  Update θ with the clipped surrogate of Eq. 8 using $\hat{A}'$
  Convert annotated rollouts into memory items and append them to $\mathcal{P}$
end for
Evaluation: run π_θ greedily with no retrieval, no memory, and no judge

C ADDITIONAL RESULTS

All results in this appendix use the evaluation protocol of Table 5 (retrieval-free, $\geq$3 seeds) unless otherwise noted.

C.1 Main Results Across Backbones

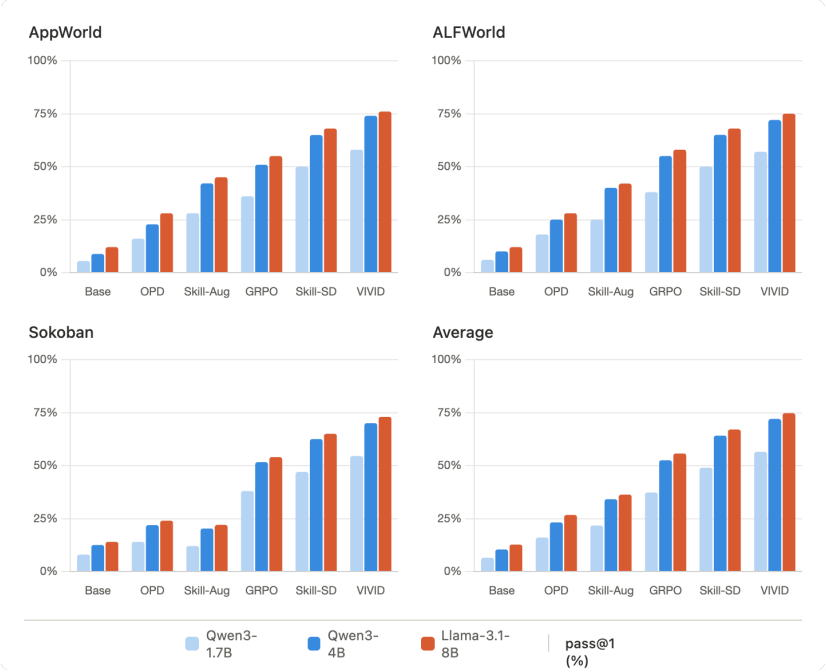

Figure 2: **Main results across three backbones** (visualization of Table 1). Pass@1 on AppWorld, ALFWorld, Sokoban, and their average for Qwen3-1.7B, Qwen3-4B, and Llama-3.1-8B. VIVID is best on every backbone–benchmark combination with a consistent +7–9 point margin over Skill-SD; Skill-Aug GRPO collapses on Sokoban, where directly injecting skills into the student prompt fails in an irreversible environment.

Figure 2 visualizes pass@1 across methods, benchmarks, and backbones. Performance increases with model scale for all methods, but VIVID remains the top-performing method at each scale, so the gains are not tied to a single backbone. Qwen3-1.7B benefits strongly from memory because it starts from a weaker base policy; Qwen3-4B and Llama-3.1-8B also benefit, which suggests that larger models exploit the retrieved precedents and the turn-level credit signal more effectively. Skill-SD improves with scale, but its gap to VIVID remains stable, which suggests that the advantage of VIVID comes from the structure of the supervision signal rather than from better use of model capacity.

Domain-level analysis. The largest absolute scores appear on AppWorld and ALFWorld, where actions are compositional and partial progress is common; there the completion score is especially informative, and VIVID improves both accuracy and completion, helping with intermediate progress as well as final success. Sokoban requires precise planning and contains many irreversible mistakes. The strong performance of VIVID on Sokoban suggests that failed trajectories are useful when preserved with step-level diagnostic labels: a failed Sokoban rollout may still contain correct early moves, a critical wrong push, and later recovery attempts, distinctions that a trajectory-level label collapses and that VIVID weights separately.

C.2 Memory Composition (Figure)

Figure 3 visualizes the composition analysis of Table 2.

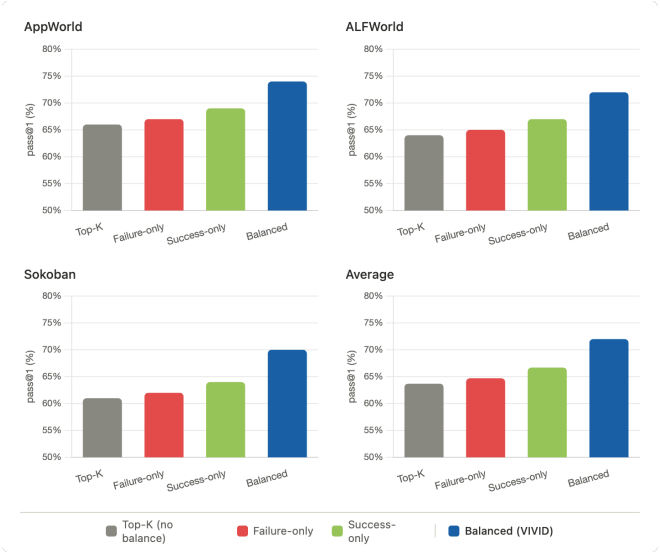

Figure 3: **Effect of memory outcome composition** on pass@1 (visualization of Table 2, left). Balanced retrieval performs best on AppWorld, ALFWorld, Sokoban, and on average: failed trajectories are most useful when paired with successful trajectories as contrastive precedents.

C.3 Training Dynamics and Cost

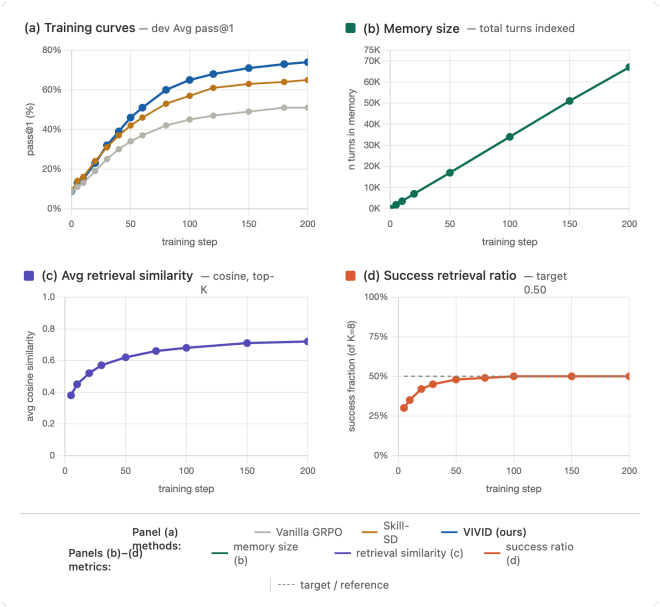

Figure 4: **Training dynamics and memory diagnostics** (Qwen3-4B). (a) Dev average pass@1 (retrieval-free) during training. (b) Indexed turns in the store. (c) Average cosine similarity of retrieved precedents. (d) Fraction of successful precedents in each retrieved block, converging to the 0.5 target. VIVID improves steadily as memory grows and keeps a clear gap over Vanilla GRPO and Skill-SD once the store is populated.

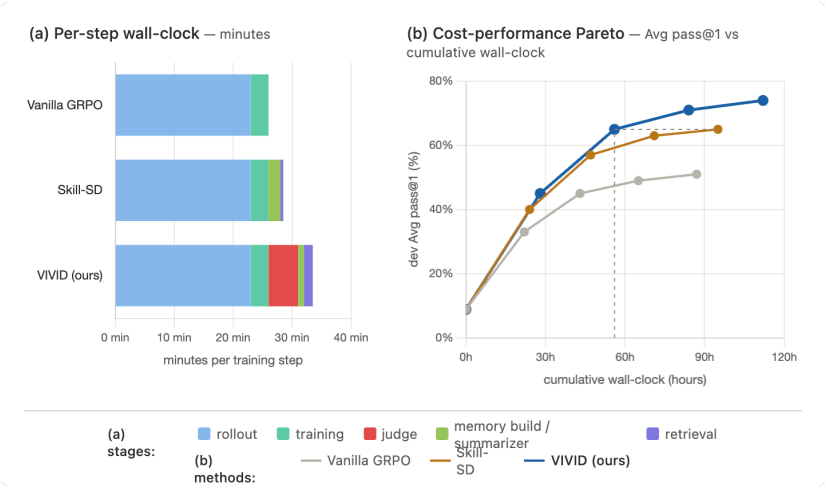

Figure 5: **Cost analysis.** (a) Per-step wall-clock breakdown: VIVID's overhead is mainly the judge, while memory construction and retrieval are small; Skill-SD's skill-extraction teacher calls are included in its bar ("memory build / summarizer"). (b) Dev average pass@1 against cumulative wall-clock: VIVID reaches Skill-SD's final pass@1 (65%) in 56 hours versus 95 hours, a 41% reduction in time-to-target.

Figures 4 and 5 support Sections 4.2 and 5.3. The number of indexed turns increases throughout training, average retrieval similarity rises as the store becomes denser, and the success-retrieval ratio moves toward the 0.5 target, so the retriever increasingly supplies a controlled mixture of successful and failed precedents. On cost, one training step takes 23 minutes for rollout, 3 minutes for optimization, 5 minutes for judging, 1 minute for memory construction, and 1.5 minutes for retrieval; the judge adds about $0.5 per step with GPT-4.1-mini.

C.4 Scaling, Repetition, Memory Size, and Transfer

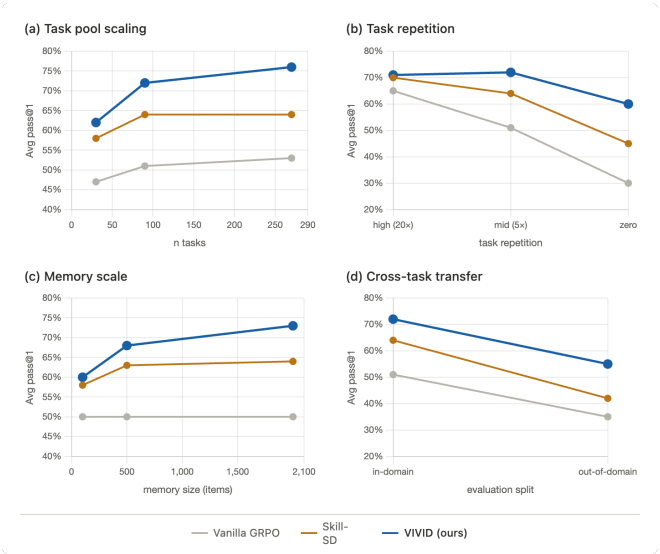

Figure 6: **Scaling and transfer analysis.** VIVID improves with larger task pools and larger memory stores, degrades more gracefully when task repetition is reduced, and remains strongest under out-of-domain evaluation.

Figure 6 studies how VIVID behaves as the training setting changes. VIVID improves as the task pool and memory store grow, so raw trajectory memory benefits from broader and denser experience. It also remains stronger than Skill-SD and Vanilla GRPO under reduced task repetition and under out-of-domain evaluation. The gap narrows when task repetition is zero or when evaluation is out of domain, but VIVID still performs best, indicating that the method benefits from repeated experience while retaining cross-task transfer. With 90 AppWorld training tasks and $\sim$34K indexed turns, part of the store consists of repeated visits to similar states; the per-task cap and MMR diversity limit near-duplicate precedents in any single block, and the zero-repetition and out-of-domain conditions in panels (b) and (d) measure transfer when repetition is removed.

C.5 Hyperparameter Sensitivity

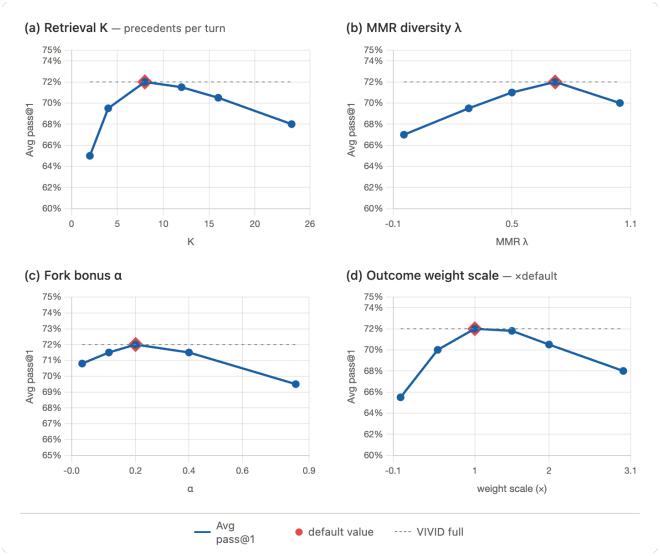

Figure 7: **Hyperparameter sensitivity** (post-hoc; red markers denote the a-priori defaults used in all experiments). VIVID is stable near the defaults, with best performance around $K = 8$, moderate MMR diversity, a small fork bonus, and the default outcome-weight scale.

Figure 7 reports sensitivity to retrieval size K, MMR coefficient λ_{M}, fork bonus α, and outcome-weight scale; the success/failure quota sweep is in Table 2. Performance is stable around the default configuration. Retrieval size peaks at $K = 8$: too few precedents provide limited context and too many introduce noise. Moderate MMR diversity and a small fork bonus work best. Outcome weights show a clear optimum near the default scale, confirming that turn labels are useful but should not dominate the policy gradient. These sweeps were run after the defaults were fixed and were not used to select them.

C.6 JUDGE LABEL DISTRIBUTION

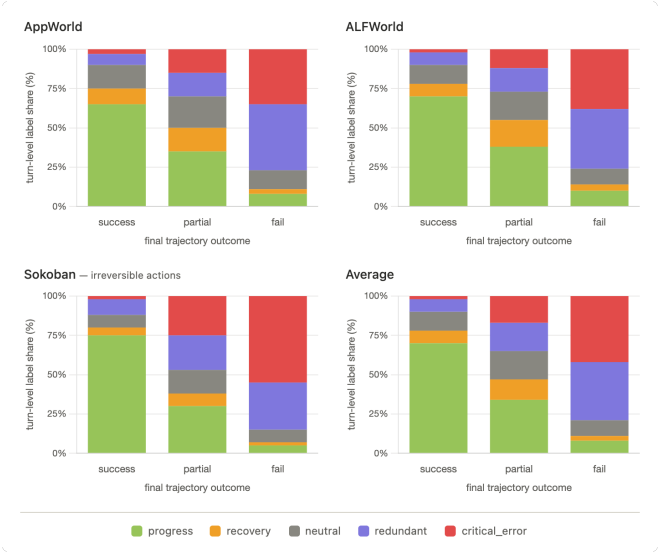

Figure 8: **Distribution of judge outcome labels by final trajectory outcome.** Successful trajectories are dominated by PROGRESS labels, failed trajectories contain larger shares of CRITICAL_ERROR and REDUNDANT labels, and partial trajectories show mixed distributions, so the judge captures turn-level structure beyond the final reward ($H(\text{label} \mid \text{outcome}) \approx 0.80$, Table 6).

Because VIVID relies on an LLM judge for turn-level annotations, we check whether the labels are merely a relabeling of the trajectory verdict. Figure 8 groups labels by final outcome. Successful trajectories contain mostly PROGRESS labels, failed trajectories much larger fractions of CRITICAL_ERROR and REDUNDANT, and partial trajectories lie in between with a mixture of progress, recovery, neutral, and error labels. Failed trajectories are not labeled uniformly as bad—they still contain progress and recovery turns—and successful trajectories contain occasional neutral or redundant turns. The judge is offline by design, so using the final outcome as evidence is the intended mechanism of hindsight credit assignment; the question is whether the labels carry information beyond the reward sign, and Section 5.3 answers it with controls (scrambled-content labels collapse to the GRPO floor while a future-blind judge retains most of the gain).

C.7 SIGN-ALIGNMENT STATISTICS

Table 6: **Sign-alignment and label-informativeness statistics** measured on training logs (Qwen3-4B; no retraining), over non-degenerate turns ($\hat{A}_i \neq 0$), in %; zero-variance groups are excluded and reported separately. Pooled "Overall" values lie within the per-benchmark range. $H(\text{label} \mid \text{outcome}) > 0$ shows that labels are not a deterministic function of the final outcome; their usefulness is established by the controls of Table 4, not by this quantity alone.

Statistic	AppWorld	ALFWorld	Sokoban	Overall
$P(\text{CRITICAL_ERROR} \mid \hat{A}_i > 0)$	4.4	3.4	4.1	4.0
$P(\hat{A}_i < 0 \mid \text{CRITICAL_ERROR})$	–	–	–	89.2
$P(\text{PROGRESS or RECOVERY} \mid \hat{A}_i < 0)$	18.1	19.2	19.5	18.9
$\text{sign}(Q_{i,t}) \neq \text{sign}(\hat{A}_i)$ rate	3.4	7.2	5.5	5.4
Zero-variance groups ($\hat{A}_i = 0$)	11.1	14.4	10.1	11.9
$H(\text{label} \mid \text{outcome})$	0.78	0.81	0.82	0.80

Table 6 quantifies how often the turn-weighting mechanism of Equation (6) acts in the intended direction. The adverse case raised in Section 5.4—a CRITICAL_ERROR turn whose rollout has

positive advantage—accounts for 4.0% of positive-rollout turns (10.8% of error turns), and the judge's sign disagrees with the advantage sign on 5.4% of turns. The largest off-diagonal entry, PROGRESS or RECOVERY turns inside negative-advantage rollouts (18.9%), is the case that balanced retrieval addresses by surfacing those turns as positive precedents.

C.8 JUDGE AGREEMENT WITH HUMAN LABELS

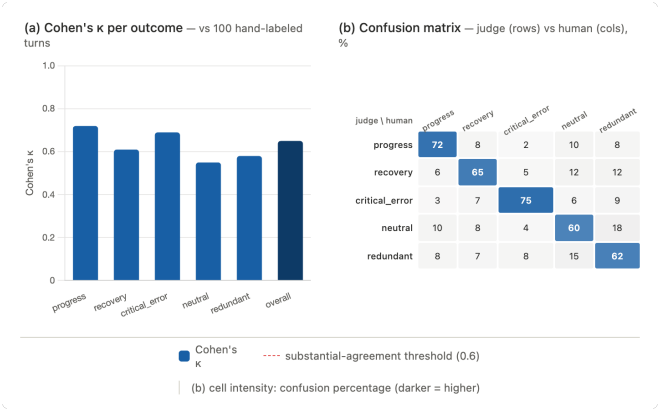

Figure 9: **Judge agreement with human labels.** Left: Cohen's κ by outcome class against 100 hand-labeled turns. Right: confusion matrix between judge labels (rows) and human labels (columns). Agreement is substantial overall, with most errors between semantically close labels.

To validate the offline judge, we compare its labels with 100 hand-labeled turns. Figure 9 shows Cohen's κ per outcome class and the judge–human confusion matrix. Agreement is above the substantial-agreement threshold of 0.6 for every class, strongest on PROGRESS and CRITICAL_ERROR, and most confusions occur between adjacent categories such as NEUTRAL and REDUNDANT. The human annotators also saw the final outcome, so this agreement certifies per-turn placement rather than outcome copying. A $\kappa \approx 0.6$ judge is reliable enough to set a bounded magnitude but not the direction of an update, which is the design rationale of Section 3.5.

C.9 JUDGE CHOICE AND COST–PERFORMANCE TRADE-OFF

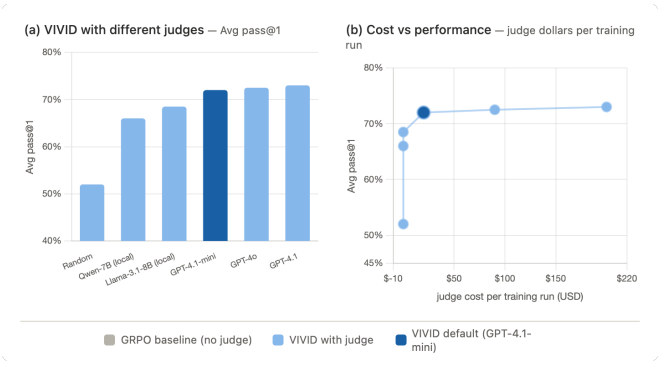

Figure 10: **Judge choice and cost–performance trade-off.** Random labels underperform, local judges provide moderate gains, and GPT-4.1-mini offers the best trade-off between accuracy and judge cost; larger judges provide only small additional gains.

Figure 10 compares judge choices. Random labels perform poorly, confirming that the gain does not come from arbitrary reweighting (see also the controls of Table 4). Local judges improve over random labels but underperform stronger API judges; GPT-4.1-mini reaches near-frontier performance at

much lower cost than larger judges, and more expensive judges yield only small additional gains, so we use it as the default. Together with the judge-free (64.5) and causal-judge (68.2) configurations, this shows that the method degrades gracefully as the supervisor weakens and that GPT-4.1-mini is a cost–performance sweet spot rather than a hard dependency.

C.10 Trajectory Length Analysis

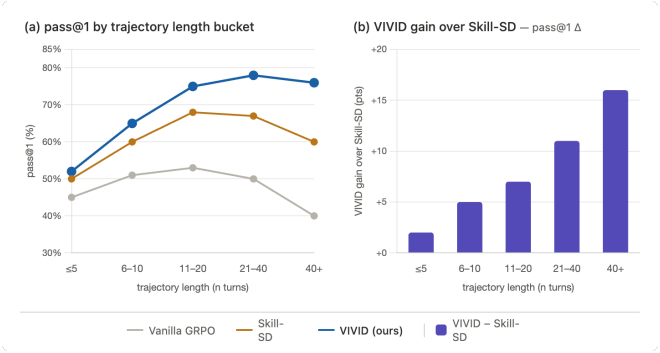

Figure 11: **Trajectory length analysis.** VIVID outperforms baselines in every length bucket, with larger gains over Skill-SD on longer trajectories, where long-horizon credit assignment is hardest.

Figure 11 analyzes performance by trajectory length. VIVID improves over Skill-SD and Vanilla GRPO in every length bucket, with the largest gains on longer trajectories: the advantage over Skill-SD grows from small gains on short trajectories to much larger gains for trajectories longer than 20 turns. As trajectories become longer, trajectory-level labels become less informative and fine-grained credit assignment more important.

D Memory Growth, Retrieval Cost, and Pruning

What store growth cannot affect. Memory is a training-time supervision bank. The trained policy carries no store at inference, per-turn prompt size is fixed by $K = 8$, and no search runs at test time. Store growth therefore cannot affect a deployed agent's latency, storage, or behavior, and it cannot affect any reported number, since evaluation is retrieval-free. This is an architectural guarantee rather than a short-run artifact: whatever the store's size at the end of training, the deliverable is a standalone policy.

Training-time cost at the tested scale. Per step, retrieval plus context construction is $\approx 1 + 1.5$ minutes against a 23-minute rollout (under 11% of a step; Section 5.3). Search is an exact FAISS inner-product search over 384-dimensional frozen MiniLM embeddings, sub-millisecond at the store sizes we reach (thousands of trajectories and tens of thousands of indexed turns by the end of a 200-step run; Figure 4), with a constant number of queries per step. The dominant training cost is the judge, not memory. Exact search is roughly linear in store size while per-step judge cost is constant, so for much larger stores an approximate index (IVF/HNSW) is the appropriate change; because it returns a different neighbor set, its effect on training quality should be measured against exact search rather than assumed equivalent.

What bounds low-quality accumulation today. Three train-time mechanisms already control the composition of any precedent block: 50/50 balancing, MMR diversity to suppress near-duplicates, and a per-source-task cap so that no single task dominates. One axis they do not target is staleness: as the policy improves, early precedents become off-policy relative to the current policy. Reporting the age and policy-version distribution of retrieved precedents and adding a recency-aware refresh are the natural next steps; our claims are scoped to the tested horizon of 200 steps.

A design constraint for pruning. Pruning, deduplication, and forgetting are needed for runs much longer than ours or for larger environments, and we present them as open problems. Our

results impose one substantive constraint: class balance drives the gain (Table 2), so pruning that breaks the success/failure ratio (naive size- or recency-based forgetting) would be actively harmful; a correct scheme must be class-balanced and must preserve the failure pool. Because the gain tracks label content rather than store size—judge-free retrieval already reaches 64.5, informative labels lift it to 72.0, and uninformative labels collapse to the GRPO floor—a class-balanced scheme that preserves informative, balanced entries should preserve the result, and the open work is engineering the store at long horizons rather than establishing that a large store produces quality. We also own the storage trade-off of design inversion I2: raw step-annotated trajectories are heavier per entry than compressed skills, deliberately, because raw turn-level annotation is what carries the turn-level credit signal; cold-entry compression (summarizing aged entries while keeping the retrieval key) is a natural mitigation that preserves the store's role. A store-stress ablation (inflating the store with stale, redundant, and class-imbalanced entries) and a minimal class-balanced cap are the first prototypes we would build.

E Potential Risks

Propagation of harmful or incorrect behavior. Because VIVID stores failed trajectories as well as successful ones, the training-time memory can contain incorrect, unsafe, or undesirable actions. The balanced retriever marks failed precedents as patterns to avoid, turn-level shaping never reinforces a turn against the verified reward, and the deployed policy carries no memory; nevertheless a model may still imitate a retrieved failure during training rollouts if the context is ambiguous. Deployments in high-stakes environments should filter unsafe actions, separate negative precedents clearly in the prompt, and audit retrieved examples.

Judge bias and annotation errors. The method relies on an LLM judge to assign outcome labels and continuous values. If the judge is biased, inconsistent, or systematically wrong for a task type, these errors can affect the magnitude of policy updates (never their direction). Our controls suggest that the labels are useful and that the damage from a mislabeled turn is bounded, but they do not eliminate this risk. Practical systems should use calibrated judges, disagreement checks, human audits, or task-specific validators when available.

Compute and accessibility. The LLM judge adds training-time cost. Although VIVID reaches target performance faster in wall-clock time, the judge cost may limit accessibility for researchers with smaller budgets. Cheaper local judges, distilled judges, selective judging, and the judge-free configuration (which already matches Skill-SD) are ways to reduce this cost.

F PROMPTS

Judge prompt (per-turn outcome and Q annotation)

You are an expert evaluator for multi-turn LLM agent trajectories. Given the task description, the full trajectory, and the final reward, annotate **each turn** with an outcome label, a continuous value Q, and a one-sentence reflection.

Inputs:
- `task`: {task_prompt}
- `trajectory`: {list of (turn_id, thought, action, execution_result)}
- `final_reward`: {0.0–1.0}

Outcome categories (assign exactly one per turn):
- PROGRESS: turn meaningfully advances toward the goal.
- RECOVERY: turn corrects a prior mistake or recovers from an error state.
- CRITICAL_ERROR: turn introduces a significant problem (wrong API call, incorrect assumption, irreversible damage).
- NEUTRAL: turn neither helps nor hurts (e.g., exploratory information-gathering with no commitment).
- REDUNDANT: turn repeats prior information or makes no state change.

Q calibration ($Q \in [-1, 1]$):
- $Q \approx +1.0$: clearly the right action; alternatives would have been worse.
- $Q \approx +0.5$: helpful but not unique; alternatives would have also worked.
- $Q \approx 0.0$: neutral.
- $Q \approx -0.5$: suboptimal; a better action was available.
- $Q \approx -1.0$: clearly wrong; led to or will lead to failure.

Reflection: a 15–25 word diagnostic explaining the assigned outcome. Be concrete about *what state* the turn was in and *why* the chosen action received this label.

Output format (strict JSON):

```
[{"turn_id": int, "outcome": str, "Q": float, "reflection": str}, ...]
```

You will see three labeled examples below before annotating the target trajectory. The judge has access to the *final reward*, so a turn that looked productive but led to failure should be labeled CRITICAL_ERROR or REDUNDANT, not PROGRESS.

Figure 12: **Prompt template for the per-turn LLM judge.** The same prompt is used across AppWorld, ALFWorld, and Sokoban; only the few-shot examples are benchmark-specific. The causal judge of Section 5.3 receives the same instructions but only the prefix up to the labeled turn and no final reward.

Judge few-shot example

Task: "Find my mom's birthday and add a reminder to my calendar one week before."
Final reward: 0.5 (calendar reminder created, but date off by one day)

Turn 0:
thought: I need to find my mom's contact first.
action: `apis.contacts.search(name="mom")`
result: `[{id: "c_47", name: "Mom", birthday: "1968-04-15"}]`

Turn 1:
thought: The birthday is April 15. The reminder should be April 8.
action: `apis.calendar.add_event(date="2024-04-09", title="Mom's birthday in 1 week")`
result: `{success: True, event_id: "e_812"}`

Turn 2:
thought: Wait, April 15 minus 7 days is April 8, not April 9. Let me fix it.
action: `apis.calendar.update_event(id="e_812", date="2024-04-08")`
result: `{success: True}`

Expected output:

```
[
{"turn_id": 0, "outcome": "progress", "Q": 0.9, "reflection":
"Correctly identifies the need to look up contact information
before scheduling; the natural first step."},
{"turn_id": 1, "outcome": "critical_error", "Q": -0.6,
"reflection": "Off-by-one date arithmetic; April 15 minus 7 days
is April 8, not April 9. Schedules wrong day."},
{"turn_id": 2, "outcome": "recovery", "Q": 0.7, "reflection":
"Agent self-corrects the arithmetic error and updates the event to
the correct date. Reward is partial due to wasted turn."}
]
```

Figure 13: **One of three judge few-shot examples.** The other two cover (i) a fully successful trajectory and (ii) a failure with two consecutive CRITICAL_ERROR turns.

Precedent block prepended to the sampling prompt at each training-rollout turn

`[RELEVANT PRECEDENTS]`
Below are similar states from past trajectories. Consider their outcomes when deciding the next action.

`===== Successful Precedents =====`

Precedent S1 (similar state from task `"smart_home_001"`, step 12):
[last 3 turns before the precedent, verbatim]
→ The agent took: `apis.smart_home.set_temperature(room="living", temp=72)`
→ This led to: PROGRESS. Agent correctly identified room from prior context.
→ At this state, alternative successful actions included: `apis.smart_home.get_status("living")`

Precedent S2 (similar state from task `"reminder_set_034"`, step 5):
[...]

`===== Failed Precedents =====`

Precedent F1 (similar state from task `"smart_home_007"`, step 8):
[last 3 turns before the precedent, verbatim]
→ The agent took: `apis.smart_home.set_temperature(room=None, temp=72)`
→ This led to: CRITICAL_ERROR. Used a placeholder room name without looking up the actual room.
→ Avoid this pattern.

Precedent F2 (similar state from task `"contacts_lookup_022"`, step 11):
[...]

`[INSTRUCTION]`
Based on the task, current state, and precedents above, produce the next thought and action. Learn from successful patterns and avoid failure patterns.

Figure 14: **Precedent block format.** $K = 8$ precedents are organized into a successful pool ($K/2$ slots) and a failed pool ($K/2$ slots). The block is injected as a user-role message for the current turn only and does not persist into the running chat history (each turn retrieves fresh). In the judge-free retrieval control of Table 4, the "This led to: ..." outcome tags and reflections are stripped and pools are formed from the verified reward alone. The block is never used at evaluation time.

G Qualitative Analysis

We include three qualitative case studies to illustrate how VIVID uses fine-grained trajectory memory during training. They complement the quantitative analyses by showing where the training signal comes from: local error detection, contrastive precedent retrieval, and memory-conditioned action selection during rollouts.

G.1 Critical Error to Recovery

Figure 15 shows a trajectory segment in which the agent first makes a critical error and later recovers. The offline judge assigns different outcome labels and values to these turns, which yields different training weights: the PROGRESS turn ($\omega = 1.0$, $|Q| = 0.9$, $f = 0.15$) receives $w \approx 0.93$, the CRITICAL_ERROR turn ($\omega = 1.5$, $|Q| = 0.6$, $f = 0.65$) receives $w \approx 1.02$, and the RECOVERY turn ($\omega = 1.3$, $|Q| = 0.7$, $f = 0.40$) receives $w \approx 0.98$, all applied to the same rollout-level advantage and hence all in the reward-consistent direction. The example illustrates two points. First, a single final reward cannot distinguish a harmful decision, a useful recovery, and redundant follow-up actions, whereas VIVID assigns them different magnitudes. Second, the CRITICAL_ERROR weight is modest here *because the judge was only moderately confident* ($|Q| = 0.6$): $|Q|$ is a confidence gate, so a low-confidence error is not over-emphasized, while a high-confidence error ($|Q| \to 1$) receives $w \to 1.8$, clearly above any PROGRESS turn. This is the intended behavior, not a cancellation of the $\omega = 1.5$ emphasis.

G.2 Retrieved Precedents Example

Figure 16 shows the precedents retrieved at a single decision point during a training rollout. The balanced retriever returns both successful and failed precedents: the successful ones converge on the correct schema (an `attendee_id` from a prior lookup plus a valid `freq` enum), while the failed ones each expose a distinct, plausible failure mode (string as ID, invalid enum, wrong endpoint shape, arithmetic bug). This is the local negative evidence that compression into a single natural-language "mistake" sentence would erase. Two of the precedents (S1 and F1) have nearly identical cosine similarity to the query; without the success/failure balance constraint, both could have been drawn from the same outcome pool.

G.3 Precedent Injection During Training Rollouts

Figure 17 compares the same policy checkpoint with and without retrieved precedents in the prompt, under greedy decoding. Without precedents, the agent replays two failure patterns present in memory; with the precedent block, it selects the correct action. This is a *mechanism study of what precedents do during training rollouts*: the block changes the action distribution from which on-policy data are sampled, and the PPO datum stores the exact precedent-conditioned prompt, so the policy is trained on actions generated under this context. It is *not* a description of the evaluation protocol. Evaluation is retrieval-free (Section 3.5), so the effect shown here does not enter any reported score; the reported gains reflect what the policy internalized, which is why the retrieval-free test performance of VIVID exceeds that of every baseline. Together with the training objective, the example shows that VIVID uses memory in two complementary training-time ways: retrieval shapes rollout generation, and turn-level weights shape gradient updates.



Figure 15: **Case study: critical error followed by recovery.** The judge assigns turn-level outcome labels and values, and VIVID converts them into per-turn weights ($w \approx 0.93, 1.02, 0.98$ for the PROGRESS, CRITICAL_ERROR, and RECOVERY turns) that scale the same rollout-level advantage, rather than assigning uniform credit to the whole trajectory.



Figure 16: **Retrieved precedents at one agent state** during a training rollout. The retriever provides a balanced set of successful and failed precedents: successful precedents show actions to imitate, while failed precedents reveal distinct local failure modes to avoid.

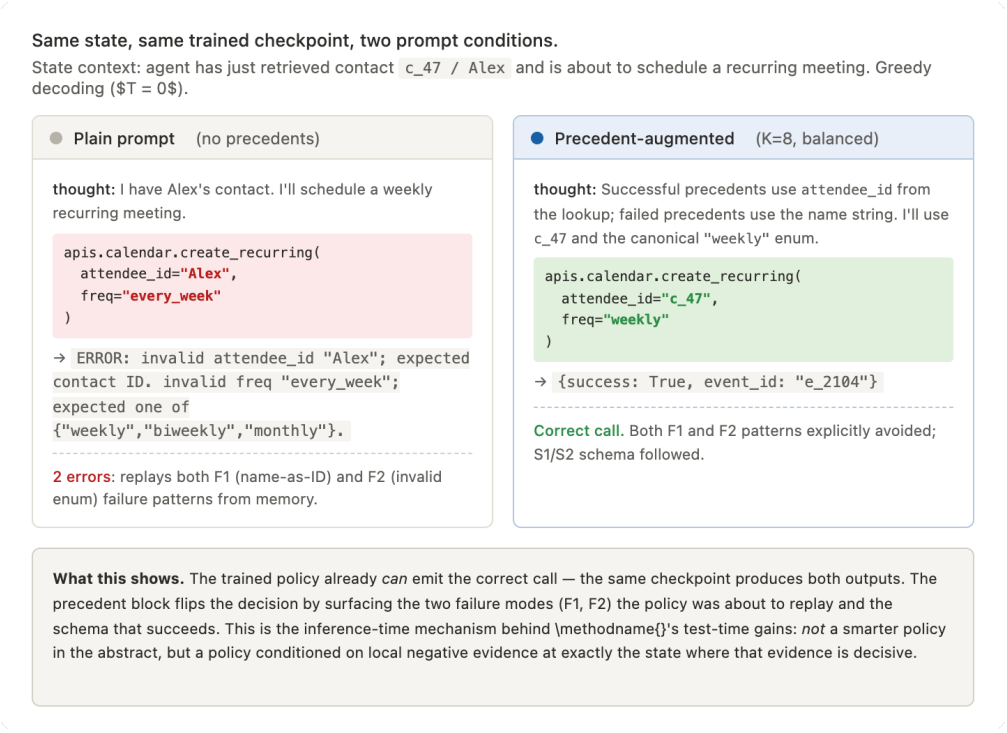

Figure 17: **Effect of precedent injection on the same policy checkpoint during a training rollout.** Without retrieved precedents, the agent chooses an incorrect action; with precedents, it uses the retrieved state–action evidence to choose the correct action. This illustrates how precedents shape the on-policy training distribution; it is not the evaluation setting, which is retrieval-free.