CONTINUOUS CONTEXT MANAGEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Long-horizon large language model (LLM) agents commonly retain their complete interaction history until compaction is triggered at a predefined threshold. We study Continuous Context Management (CCM), which performs compaction at every turn to prevent interaction history from accumulating in the active prompt. At each turn, a CCM agent emits an updated memory together with an environment action; its next prompt contains the original task, retained memory, and newest observation rather than the complete transcript. We first evaluate CCM without fine-tuning on TerminalBench-2 using Claude Sonnet 4.6, Claude Opus 4.6, GLM-5, and Kimi K3. CCM substantially reduces cumulative input usage and active-prompt size, although it lowers task success for most models while preserving performance for Kimi K3. We use GRPO with privileged full-history distillation to improve CCM in open-weight models. A frozen copy of the student's initial model scores each sampled student action under the complete history reconstructed from that student's rollout, providing dense action-token supervision without a separate teacher rollout or reference solution. On WebShop, this objective substantially improves CCM over GRPO at both evaluated model scales and surpasses full-history GRPO for Qwen3-4B-Instruct, though not for Qwen3-8B. On Endless Terminals, the augmented method provides a modest improvement over GRPO, with both CCM policies outperforming the untrained full-history baseline. These results demonstrate that CCM is a viable inference paradigm for agents operating with substantially reduced retained context and that its performance can be improved through reinforcement learning with privileged full-history distillation.

1 INTRODUCTION

Long-horizon language agents must preserve useful information across repeated interactions with their environments. A shopping agent may need to remember product attributes encountered several pages earlier, while a terminal agent may need to retain the outcome of a previous command. Keeping the complete interaction history makes this information directly accessible, but causes the active context to grow as the task proceeds. The accumulated history is bounded by the model's context window, and model performance can degrade even within this limit as inputs grow longer (Liu et al., 2024; Du et al., 2025; Hong et al., 2025) or include irrelevant information (Shi et al., 2023).

A common solution is to compact the interaction history when the context reaches a predefined token threshold, replacing earlier content with a shorter summary (Wu et al., 2025; Li et al., 2026c). Repeated compaction allows a session to continue beyond the length of a single context window. However, the context still accumulates between compaction events and can occupy a large fraction of the available window before being compressed again. An ultimate form of this accumulate-compact paradigm is that memory management become part of every action step, with the agent continually updating the information it carries forward. We study this setting which we call Continuous Context Management (CCM), where the agent produces an updated memory alongside each action, its next context constructed from the updated memory and the newest observation. "Continuous" refers to memory management at the finest step resolution, not its meaning in mathematics. This is particularly useful when GPU memory is a major constraint, such as deploying LLMs locally on consumer-grade GPUs, where keeping the active context compact can reduce the token cache memory, allowing larger models to run within the same GPU memory budget.

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.

We first evaluate CCM as an out-of-the-box inference paradigm on TerminalBench-2 using Claude Sonnet 4.6, Claude Opus 4.6, GLM-5, and Kimi K3. Across these models, CCM substantially reduces cumulative input usage and typical active-prompt size. Success declines for Sonnet, Opus, and GLM-5, but increases slightly for Kimi K3. The resulting monetary savings depend strongly on provider-specific prompt-caching policies and token prices: prefix caching can substantially reduce the cost of full-history prompting despite its greater token usage. These results demonstrate CCM's potential to reduce context usage while revealing a performance gap for three of the four models, motivating our investigation of training methods for CCM agents.

Prior work has established the feasibility of learning such memory-management behaviors, such as MEM1, MemAgent, MEMENTO, and Compaction RL (Zhou et al., 2025; Yu et al., 2025; Kontonis et al., 2026; Li et al., 2026c). Building on these approaches, particularly the per-turn consolidation used in MEM1, we combine two training signals to train agents both to manage memories and to use them to generate actions. These two capabilities are closely coupled. An agent may preserve the relevant facts yet fail to use them when choosing an action; conversely, an effective action policy cannot reliably compensate for essential information omitted from its memory. Reinforcement learning can optimize both capabilities through task rewards, but episode-level outcomes provide limited guidance about individual task actions. To provide additional supervision for actions generated from compact memory, we use an additional distillation objective building on on-policy context distillation (Ye et al., 2026) and the gated self-distillation approach of SDAR (Lu et al., 2026), where the teacher sees the full interaction history as privileged information. The teacher is initialized with the same weights as the student before training and remains frozen throughout training. At each step, it receives the full history preceding the current action and scores the student-sampled action tokens autoregressively. A token-level teacher–student confidence-gap gate weights the policy gradient on action tokens. This procedure provides action-token supervision without a separately trained teacher, additional teacher rollout, or successful reference trajectory.

We evaluate this training recipe on WebShop and Endless Terminals, focusing on improving agent performance while using only a fraction of the prompt context required by full-history baselines. On WebShop, adding full-history distillation substantially improves task success over CCM trained with GRPO alone for both models. On Endless Terminals, both trained CCM policies outperform the untrained full-history baseline while maintaining compact model-written memories. Distillation provides a modest improvement over CCM + GRPO. Together, these results show that our training recipe improves CCM's task performance while retaining its substantial savings in prompt length.

Our main contributions are:

- We study Continuous Context Management (CCM), in which agents update a compact memory at every action step. Our evaluation on TerminalBench-2 demonstrates substantial reductions in input-token usage without additional training, and characterizes the task performance and API-cost tradeoffs.
- We introduce a training recipe that combines GRPO with privileged full-history distillation. Experiments on WebShop and Endless Terminals show that training improves CCM's task performance while substantially reducing context length.

2 Related Works

Context management for long-horizon agents. ReAct-style agents retain the full sequence of reasoning, actions, and observations in subsequent prompts (Yao et al., 2022b), so the context grows with the horizon and performance can degrade on long inputs (Du et al., 2025). Inference-time methods extend the horizon through external memory, periodic summarization, structured compression, and evolving context representations (Packer et al., 2023; Wu et al., 2025; Kang et al., 2025; Wan et al., 2025; Zhang et al., 2026a; Li et al., 2026a), and several works train models to generate summaries or decide when to compress (Lu et al., 2025; Li et al., 2026b; Zhang et al., 2026c; Yu et al., 2025; Kontonis et al., 2026). MEM1 (Zhou et al., 2025) is the closest in formulation to our method. At every turn the agent emits an internal state that consolidates its previous state with the newest observation, and earlier turns are pruned. MEM1 learns this behavior from outcome rewards with PPO, training on the concatenated trajectory with an attention mask that restricts each token to the context available when it was generated. Our study differs from MEM1 in several respects. First,

we show that frontier models can use CCM out-of-box without additional training, substantially reducing context usage, although most evaluated models incur a drop in task performance. Second, rather than relying on outcome rewards alone to train the model to manage its state, we combine episode-level GRPO with a designed token-level distillation objective.

Reinforcement learning and on-policy distillation. Reinforcement learning methods such as GRPO provide trajectory-level supervision from environment or verifier rewards (Shao et al., 2024; Dong et al., 2025; Feng et al., 2025). On-policy distillation, including on-policy self-distillation, complements this feedback with dense token-level guidance from a stronger teacher or from the same policy conditioned on privileged information (Ye et al., 2026; Yang et al., 2026; MiMo Team, 2026; GLM-5 Team, 2026; Zhao et al., 2026; He et al., 2026; Zhang et al., 2026b). Lu et al. (2026) proposed Self-Distilled Agentic Reinforcement Learning (SDAR), which combines reinforcement learning and on-policy self-distillation using token-level gates based on student uncertainty or the teacher–student confidence gap. We adapt its confidence-gap gating to multi-turn CCM agents. For each sampled action token, we compare its probability under the current policy's compact CCM context with its probability under a frozen copy of the student's initial model, conditioned on the reconstructed full history. The teacher weights remain fixed throughout training. This provides an auxiliary action-token distillation signal without requiring a separately trained teacher, an additional teacher rollout, or a successful reference trajectory.

3 Continuous Context Management

3.1 Method Description

For a task x, let o_0 denote the initial environment observation and m_0 the initially empty memory. At turn $n \geq 1$, a Continuous Context Management (CCM) agent receives observation o_{n-1} and retained memory m_{n-1}, and generates updated memory m_n and action a_n according to

$$(m_n, a_n) \sim \pi_\theta(\cdot \mid x,\, m_{n-1},\, o_{n-1})\,, \tag{1}$$

The generated response consists of an updated memory followed by an environment action, $y_n = [\, m_n \,;\, a_n \,]$. Executing a_n produces the next observation o_n. The agent then constructs the prompt for turn $n+1$ from x, m_n, and o_n. Previous responses and observations are not included directly. Information from earlier turns therefore remains available only if the agent decides to preserve it in m_n.

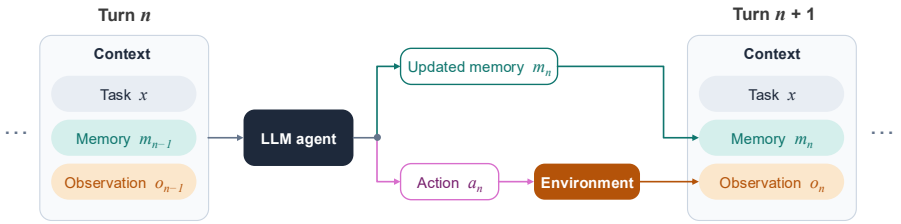


Figure 1: Continuous Context Management. At each turn, the agent receives the original task, its memory from the preceding turn, and the current observation. It emits an updated memory followed by an environment action. The next prompt is constructed from the updated memory and newest observation rather than the complete interaction transcript.

3.2 TerminalBench-2 results

Terminal-Bench 2.0 (Merrill et al., 2026) contains 89 challenging, human-curated tasks that require agents to complete realistic workflows through terminal interaction. We evaluate CCM on all 89 tasks using Claude Sonnet 4.6, Claude Opus 4.6, GLM-5, and Kimi K3. For each model, we compare CCM against a standard full-history baseline using pass@1.

Table 1 reports task success, token usage, and estimated API cost per task; Table 5 additionally reports interaction length and detailed token accounting. Cost estimates for Claude Sonnet 4.6,

Table 1: Terminal-Bench 2.0 pass@1 results over 89 tasks. Costs are estimated means per task and include prompt caching where available; prompt caching was unavailable for GLM-5 through Amazon Bedrock.

Model	Mode	Success, n (%)	Peak prompt tokens		Maximum retained memory	Estimated cost per task
			Median	Maximum		
Claude Sonnet 4.6	Baseline	**38 (42.7)**	16,673	131,157	–	**$0.468**
	CCM	28 (31.5)	**3,568**	**50,614**	1,219	$0.472
Claude Opus 4.6	Baseline	**52 (58.4)**	15,405	103,512	–	**$0.831**
	CCM	36 (40.4)	**4,280**	**51,457**	1,403	$1.087
GLM-5	Baseline	**31 (34.8)**	17,560	116,407	–	$0.515
	CCM	26 (29.2)	**4,279**	**47,287**	561	**$0.124**
Kimi K3	Baseline	56 (62.9)	9,472	74,313	–	**$0.302**
	CCM	**57 (64.0)**	**3,243**	**27,645**	1,058	$0.622

Claude Opus 4.6, and Kimi K3 account for provider-side prompt caching. GLM-5 is reported without prompt caching because Amazon Bedrock does not support it for this model. Inference and token-accounting configurations are provided in Appendix A.

As shown in Table 1, CCM substantially reduces peak prompt size across all four models. Median episode-level peaks decrease from 9,472–17,560 tokens under full-history prompting to 3,243–4,280 tokens under CCM. The largest retained CCM memories contain only 561–1,403 tokens. Maximum CCM prompts can nevertheless be considerably larger because each prompt also includes the newest terminal observation, which may contain up to 60,000 characters. CCM limits accumulated interaction history but does not compress the current observation.

CCM also reduces mean cumulative input tokens per task by 83.1% for Claude Sonnet 4.6, 81.5% for Claude Opus 4.6, 84.3% for GLM-5, and 75.8% for Kimi K3. This compression is accompanied by lower pass@1 success for Sonnet, Opus, and GLM-5, with decreases of 11.2, 18.0, and 5.6 percentage points, respectively. In contrast, Kimi K3 improves slightly under CCM, from 62.9% to 64.0%.

The monetary effect depends strongly on provider-side prompt caching. For Sonnet, the two methods have nearly identical estimated costs: $0.468 per task for full-history prompting and $0.472 for CCM. For Opus and Kimi, caching makes full-history prompting less expensive despite its substantially greater cumulative input usage: $0.831 versus $1.087 for Opus and $0.302 versus $0.622 for Kimi. Because our GLM-5 endpoint does not support prompt caching, its input reduction lowers estimated cost from $0.515 to $0.124 per task. Overall, CCM consistently reduces peak and cumulative context usage, but its effects on task success and monetary cost depend on the model and the provider's caching and pricing policies.

4 Improving Continuous Context Management via Reinforcement Learning

4.1 GRPO with Self-Distillation

We improve the CCM policy by combining GRPO and a distillation-based objective. We refer to the CCM policy defined in Section 3 as the *student*, and denote its response at turn n by $y_n^S := y_n = [\, m_n \,;\, a_n \,]$. The student rollouts follow the CCM interaction process illustrated in Figure 1. GRPO optimizes all valid tokens in the student response, including memory and action tokens (details in Appendix B).

Beyond GRPO, we also use a distillation-based objective. The *teacher* policy is a frozen copy of the student model before training. Its weights remain fixed throughout training, while the student is updated. At each turn, the teacher uses the full environment history to provide action-token supervision for the student acting from compact memory.

Concretely, we reconstruct the student's environment history before the current action a_n as privileged information, $h_n^S = (o_0, a_1, o_1, \ldots, a_{n-1}, o_{n-1})$, where o_0 is the initial observation. This history includes only prior actions and observations, excluding the current action a_n. For token position t within the current action a_n, the student is conditioned on $s_{n,t}^C = P_C(x, m_{n-1}, o_{n-1}, m_n, a_{n,<t})$, where P_C is the CCM prompt renderer, m_n has already been generated as the first part of the current response, and $a_{n,<t}$ denotes the preceding action-token prefix. The privileged teacher is conditioned on $s_{n,t}^+ = P_F(x, h_n^S, a_{n,<t})$, where P_F is the full-history prompt renderer. When scoring $a_{n,t}$, the teacher receives only h_n^S and the preceding action-token prefix $a_{n,<t}$ in addition to the task x; neither the target token nor subsequent action tokens are included in its conditioning context. The student and teacher score the same sampled action token from the same trajectory. The student uses its current weights and compact memory, whereas the teacher uses fixed initial weights and the complete pre-action environment history.

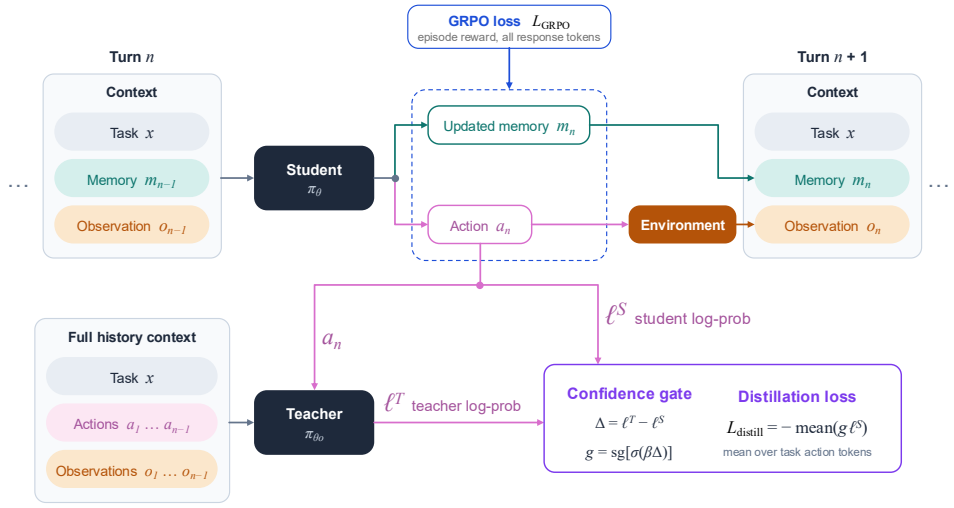


Figure 2: Training CCM agents with GRPO and full-history self-distillation. **Top:** the student π_θ rolls out under CCM. GRPO with episode task reward optimizes all response tokens, both memory and action. **Bottom:** the teacher π_{θ_0} is a copy of the student's initial model, with weights held fixed throughout training. It receives the pre-action environment history reconstructed from the student's rollout and scores the student-sampled action autoregressively. For each action token, the teacher log-probability ℓ^T and the student log-probability ℓ^S give the confidence gap Δ. The gap sets the gate g, which weights the distillation loss $\mathcal{L}_{\text{distill}} = -\text{mean}(g\,\ell^S)$ over valid action tokens only. The total training loss is composed of the GRPO loss and distillation loss.

For a student sampled action token $a_{n,t}$, we define the teacher student confidence gap as

$$\Delta_{n,t} = \text{stopgrad}\left[\log \pi_{\theta_0}(a_{n,t} \mid s_{n,t}^+) - \log \pi_\theta(a_{n,t} \mid s_{n,t}^C)\right], \tag{2}$$

where θ_0 denotes the student's initial parameters before any training updates. The teacher π_{θ_0} remains frozen throughout training and is never refreshed from the updated student. The operator $\text{stopgrad}[\cdot]$ treats its argument as constant during backpropagation.

Following the gap-gating formulation of SDAR (Lu et al., 2026), we convert the confidence gap into a token-level gate, $g_{n,t} = \text{stopgrad}\left[\sigma(\beta\Delta_{n,t})\right]$, where σ is the logistic sigmoid and $\beta > 0$ controls gate sharpness. Tokens receiving greater confidence from the privileged teacher have $\Delta_{n,t} > 0$ and receive larger gating weights. Tokens already receiving greater confidence from the student have $\Delta_{n,t} < 0$ and are attenuated, but retain a small positive weight.

Let $M_{i,n,t} \in \{0, 1\}$ indicate that token t belongs to a valid action block at turn n of rollout i and has valid student and teacher scores. The distillation loss is averaged over all eligible action tokens:

$$L_{\text{distill}} = -\sum_{i,n,t} M_{i,n,t} g_{i,n,t} \log \pi_\theta(a_{i,n,t} \mid s_{i,n,t}^C) / \sum_{i,n,t} M_{i,n,t}. \tag{3}$$

The distillation loss applies to all valid student-generated action tokens (task actions only, unlike GRPO which is on both memory and task actions). It weights each token's gradient according to the teacher–student log-probability gap. When the full-history teacher assigns a token higher probability than the compact-context student, the token receives a larger change of gating weight, encouraging the student to reproduce the token supported by the additional context. Negative-gap tokens receive smaller gating weights, scaling down the magnitude of gradients. In Appendix C, we show at fixed contexts under on-policy sampling, the expected update is equivalent to using a bounded, centered sigmoid transformation of OPD's signed log-probability gap, recovering a scaled OPD gradient locally for small gaps.

The combined training loss is $L_{\text{total}} = L_{\text{GRPO}} + \lambda_{\text{distill}} L_{\text{distill}}$. We set $\lambda_{\text{distill}} = 0.01$ and $\beta = 5$. Other training config details are shown in Appendix E.

4.2 WebShop Results

WebShop (Yao et al., 2022a) is a simulated shopping benchmark that tests an agent's ability to navigate webpages, select product options, and complete purchases. Figure 3 compares CCM + GRPO with CCM + GRPO + distillation across training checkpoints for Qwen3-4B-Instruct and Qwen3-8B. The untrained CCM and full-history conditions provide fixed reference points. Table 2 additionally reports full-history GRPO and summarizes task performance, memory behavior, and prompt size at each trained method's best observed checkpoint.

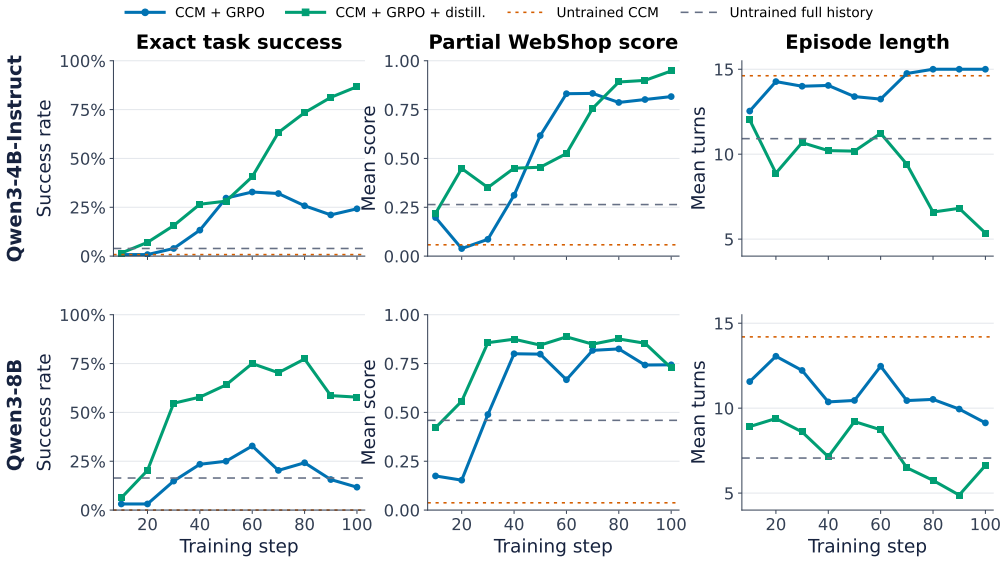


Figure 3: WebShop checkpoint evaluation for Qwen3-4B-Instruct (top) and Qwen3-8B (bottom) on a fixed 128-task pass@1 evaluation with temperature 0.4, seed 42, and a 15-turn limit. Columns report exact task success, partial WebShop score, and mean episode length. Solid curves show CCM + GRPO and CCM + GRPO + distillation across checkpoints 10–100. Dashed lines indicate the corresponding untrained CCM and full-history performance.

At step 100, adding privileged distillation substantially improves CCM over CCM + GRPO at both model scales. For Qwen3-4B-Instruct, adding privileged distillation improves exact success by 62.50 percentage points, from 24.22% to 86.72%, while partial score increases from 0.817 to 0.948. The distillation-augmented CCM policy also exceeds full-history GRPO, which achieves 75.78% exact success and a partial score of 0.839. It completes tasks in 5.36 turns on average, compared with 5.85 turns for full-history GRPO and 15.00 turns for CCM + GRPO.

For Qwen3-8B at step 100, adding privileged distillation improves exact success by 46.09 percentage points over CCM + GRPO, from 11.72% to 57.81%. Its partial score is slightly lower, at 0.728 versus 0.744. Full-history GRPO remains the strongest condition for this model, reaching 80.47%

Table 2: Best observed task success and context behavior on the fixed 128-task WebShop evaluation. Prompts are truncated at the 4,096-token model-input limit.

Condition	Best performance		Memory behavior		Memory size		Peak prompt tokens	
	Exact success (%)	Step	Update rate (%)	Prefix retention (%)	Mean	Maximum	Median	Maximum
Qwen3-4B-Instruct								
Untrained full history	3.91	–	–	–	–	–	4,096	4,096
Full-history GRPO	81.25	70	–	–	–	–	1,721	3,076
Untrained CCM	0.78	–	57.6	69.6	225.7	861	1,232	1,908
CCM + GRPO	32.81	60	12.9	98.5	95.6	147	1,045	1,327
CCM + GRPO + distillation	**86.72**	100	100.0	77.2	60.1	89	**1,041**	**1,279**
Qwen3-8B								
Untrained full history	16.41	–	–	–	–	–	2,258	4,096
Full-history GRPO	**82.81**	70	–	–	–	–	1,738	4,096
Untrained CCM	0.00	–	15.0	96.7	511.8	1,013	1,582	2,127
CCM + GRPO	32.81	60	9.1	99.4	510.6	994	1,200	1,954
CCM + GRPO + distillation	77.34	80	96.9	82.3	185.3	421	**1,149**	**1,435**

exact success and a partial score of 0.907. Mean episode length is 6.63 turns for CCM + GRPO + distillation, 9.13 turns for CCM + GRPO, and 5.55 turns for full-history GRPO.

Condensed matched trajectories are reported in Appendix F.1. In the selected task, the untrained policies fail during search or navigation, while CCM + GRPO reaches the correct product but repeatedly selects an already chosen option. CCM + GRPO + distillation instead follows the required product–color–size–purchase sequence.

Table 2 reports memory-update behavior and prefix reuse at each trained method's best observed checkpoint. The CCM + GRPO policies modify their memories infrequently. Qwen3-4B-Instruct and Qwen3-8B have memory-update rates of 12.9% and 9.1%, while retaining nearly their entire preceding memory prefixes. Adding distillation produces more active memory management. Qwen3-4B-Instruct changes its memory on every consecutive turn while retaining 77.2% of the preceding prefix; Qwen3-8B changes it on 96.9% of turns while retaining 82.3%. The distillation objective is intended to improve memory use rather than memory management directly. However, improve memory use helps the GRPO to be more effective, consequently improves both memory action and task action.

Relative to CCM + GRPO, adding privileged distillation reduces cumulative input from 10,636 to 3,720 tokens for Qwen3-4B-Instruct and from 9,929 to 5,578 tokens for Qwen3-8B, reductions of 65.0% and 43.8%. These differences partly reflect earlier task completion rather than context compression alone. CCM also does not automatically reduce total input, that the untrained Qwen3-8B CCM policy takes more turns than its full-history counterpart and consequently processes more tokens.

4.3 Endless Terminals Results

We also compare CCM + GRPO against CCM + GRPO + distillation on the Endless Terminals (Gandhi et al., 2026) using Qwen3-8B, where agents execute terminal commands to manage files, process data, analyze logs, write scripts, and operate databases. The Figure 4 reports task success and mean episode length. Dashed lines show the untrained CCM and full-history policies under the same evaluation protocol.

The untrained full-history and CCM policies perform similarly, solving 20.33% and 21.00% of tasks, respectively. This difference corresponds to only two tasks out of 300. The two conditions solve 43 tasks in common, while 18 are solved only with full history and 20 only with CCM. One possible explanation for such a small gap in contrast with the other benchmarks is that Endless Terminals externalizes part of the relevant state in its persistent sandbox as files, and shell state survive across turns, allowing omitted information from the context to be recovered through additional inspection commands.

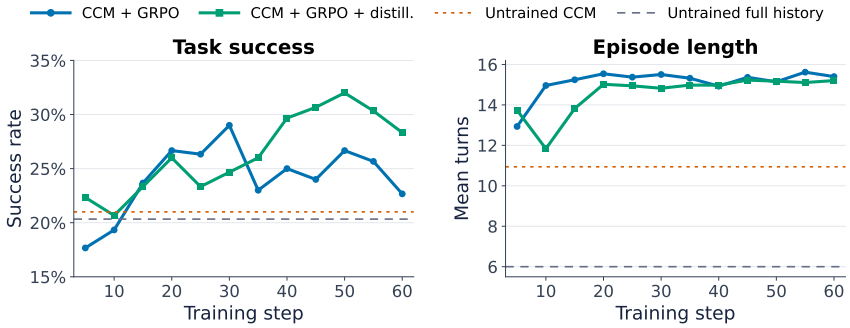


Figure 4: Endless Terminals checkpoint evaluation for Qwen3-8B on the fixed 300-task test set with a 16-turn limit. Curves report task success (left) and mean episode length (right) for CCM + GRPO and CCM + GRPO + distillation. Dashed lines indicate the corresponding untrained CCM and full-history policies.

Table 3: Best observed task success and memory behavior through training step 60 on the fixed 300-task Endless Terminals evaluation with Qwen3-8B. For each trained method, we report the evaluated checkpoint with the highest success, breaking ties in favor of the earlier checkpoint.

	Best performance		Memory behavior		Memory size (tokens)	
Condition	Success (%)	Step	Update rate (%)	Prefix retention (%)	Mean	Maximum
Untrained full history	20.33	–	–	–	–	–
Full-history GRPO	**35.33**	30	–	–	–	–
Untrained CCM	21.00	–	34.8	81.2	63.9	346
CCM + GRPO	29.00	30	16.9	94.6	77.1	720
CCM + GRPO + distillation	32.00	50	36.9	72.9	144.0	906

The best observed success is 3.00 percentage points higher with CCM + GRPO + distillation: 32.00% at step 50, compared with 29.00% at step 30 for CCM + GRPO. At step 60, the two methods achieve 22.67% and 28.33%, respectively. Both selected CCM policies use nearly the full 16-turn allowance. Full-history GRPO achieves the strongest performance, reaching 35.33% at step 30. Although both trained CCM methods outperform the untrained full-history and CCM policies, neither matches full-history GRPO. Examples of condensed matched trajectories appear in Appendix F.2. We report one task solved only by CCM + GRPO + distillation among the two trained CCM policies and one solved only by CCM + GRPO.

Table 3 reports task success and memory behavior through training. CCM + GRPO + distillation updates its memory more frequently than CCM + GRPO, at 36.9% versus 16.9% of consecutive turns, and preserves a smaller prefix of the preceding memory, at 72.9% versus 94.6%. The distillation-augmented policy also maintains a larger memory, with a mean of 144.0 tokens and a maximum of 906 tokens, compared with a mean of 77.1 tokens and a maximum of 720 tokens for CCM + GRPO. We omit peak-prompt statistics because the terminal observation can itself be very large in this benchmark, we set a cut off limit of the context length for all the conditions, and every condition hits this limit at least in one episode. Because CCM manages retained interaction history but does not compress the newest observation, these prompt-size statistics do not cleanly measure the amount of memory retained by the agent, but dominated by long observations and our hard cut-off limit on its length.

4.4 General-Capability Retention

Prior work shows that on-policy distillation can improve task performance while mitigating forgetting of existing capabilities (Shenfeld et al., 2026; Ye et al., 2026). We investigate whether privileged full-history distillation provides a similar benefit beyond GRPO alone during agent training under

Table 4: General-capability retention after agent post-training. Values are mean percentage scores over three decoding seeds, with sample standard deviations. Parentheses report absolute percentage-point changes from the corresponding pretrained base model, computed before rounding.

Method	MMLU-Pro	HellaSwag	IFEval strict
WebShop, Qwen3-4B-Instruct			
Pretrained base	65.25 ± 0.66	80.14 ± 0.07	83.30 ± 0.56
Full-history GRPO	62.48 ± 0.49 (-2.77)	67.92 ± 0.42 (-12.22)	80.35 ± 0.65 (-2.96)
CCM + GRPO	55.16 ± 0.60 (-10.09)	66.68 ± 0.71 (-13.46)	82.32 ± 0.47 (-0.99)
CCM + GRPO + distillation	64.23 ± 0.25 (-1.03)	74.53 ± 0.20 (-5.62)	83.18 ± 0.49 (-0.12)
WebShop, Qwen3-8B			
Pretrained base	56.88 ± 0.65	79.54 ± 0.39	81.95 ± 1.23
Full-history GRPO	54.28 ± 1.00 (-2.60)	76.68 ± 0.80 (-2.86)	79.48 ± 1.40 (-2.46)
CCM + GRPO	58.25 ± 0.40 $(+1.37)$	73.94 ± 0.44 (-5.60)	81.33 ± 0.67 (-0.62)
CCM + GRPO + distillation	56.95 ± 0.25 $(+0.07)$	76.71 ± 0.17 (-2.83)	81.58 ± 1.17 (-0.37)
Endless Terminals, Qwen3-8B			
Pretrained base	56.88 ± 0.65	79.54 ± 0.39	81.95 ± 1.23
Full-history GRPO	58.63 ± 0.80 $(+1.74)$	79.31 ± 0.42 (-0.23)	82.44 ± 0.67 $(+0.49)$
CCM + GRPO	60.61 ± 0.92 $(+3.73)$	77.38 ± 0.12 (-2.16)	82.44 ± 1.33 $(+0.49)$
CCM + GRPO + distillation	61.75 ± 0.37 $(+4.87)$	78.48 ± 0.79 (-1.06)	82.38 ± 0.56 $(+0.43)$

CCM. To measure retention, we evaluate the pretrained base models and checkpoint-100/60 (WebShop/Endless Terminal) policies on MMLU-Pro, HellaSwag, and IFEval. Table 4 reports mean scores over three decoding seeds.

On WebShop with Qwen3-4B-Instruct, privileged distillation substantially reduces the degradation associated with CCM + GRPO. Relative to the pretrained model, the distillation-augmented policy changes by -1.03, -5.62, and -0.12 percentage points on MMLU-Pro, HellaSwag, and IFEval. For WebShop with Qwen3-8B, the differences between methods are smaller. Distillation improves HellaSwag and IFEval retention relative to CCM + GRPO, while CCM + GRPO obtains the highest MMLU-Pro score.

On Endless Terminals, all three post-trained policies improve over the pretrained model on MMLU-Pro and remain close to it on IFEval. CCM + GRPO + distillation obtains the highest MMLU-Pro score and retains more HellaSwag performance than CCM + GRPO, while full-history GRPO remains closest to the pretrained HellaSwag score. Overall, privileged distillation provides the clearest retention benefit for the 4B WebShop model. Its effect is not uniform across model scales, training environments, or evaluation tasks.

5 Discussion and Conclusion

CCM demonstrates that agents can operate with compact, continually updated memory, substantially reducing retained context. Privileged full-history distillation improves task accuracy while mitigating forgetting of general capabilities, with the clearest joint benefit on WebShop with Qwen3-4B-Instruct. This approach is particularly relevant when model weights and the KV cache compete for limited GPU memory, such as local agents running on consumer GPUs or servers supporting many concurrent agent sessions. Smaller active contexts could accommodate larger models or more simultaneous tasks within the same memory budget. However, the benefits vary across models and environments, and shorter prompts do not necessarily translate into lower API costs, which depend on the prefix caching and pricing policy.

AI USE STATEMENT

Generative AI tools assisted with manuscript wording, LaTeX formatting, and checks of bibliographic metadata against primary sources.

ETHICS STATEMENT

This work studies context management for agents that act in shopping and terminal environments. Errors or omissions in retained memory can affect later actions, and retained memory may contain sensitive information from observations. Applications beyond these benchmarks should therefore use appropriate access controls, data-handling safeguards, and human oversight for consequential actions. The reported benchmark results do not establish that CCM agents are safe or reliable for unrestricted deployment.

REPRODUCIBILITY STATEMENT

Section 3 defines the CCM interaction protocol, and Section 4 describes the training objective. Appendix A provides the Terminal-Bench 2.0 inference settings, prompt templates, and token accounting. Appendices B and C specify the optimization settings and distillation objective. Appendix E documents the training and evaluation splits, rewards, decoding settings, rollout limits, model variants, and training framework; Appendix E.5 provides the CCM prompt templates. These details support reproduction of the reported protocol, although results obtained through hosted model APIs may vary with provider updates.

REFERENCES

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, Guorui Zhou, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Agentic reinforced policy optimization, 2025. URL https://arxiv.org/abs/2507.19849.

Yufeng Du, Minyang Tian, Srikanth Ronanki, Subendhu Rongali, Sravan Bodapati, Aram Galstyan, Azton Wells, Roy Schwartz, Eliu A Huerta, and Hao Peng. Context length alone hurts llm performance despite perfect retrieval. *arXiv preprint arXiv:2510.05381*, 2025.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for LLM agent training, 2025. URL https://arxiv.org/abs/2505.10978.

Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling RL environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026. URL https://arxiv.org/abs/2601.16443.

GLM-5 Team. GLM-5: From vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Yinghui He, Simran Kaur, Adithya Bhaskar, Yongjin Yang, Jiarui Liu, Narutatsu Ri, Liam Fowl, Abhishek Panigrahi, Danqi Chen, and Sanjeev Arora. Self-distillation zero: Self-revision turns binary rewards into dense supervision, 2026. URL https://arxiv.org/abs/2604.12002.

Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Technical report, Chroma, July 2025. URL https://trychroma.com/research/context-rot.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents, 2025. URL https://arxiv.org/abs/2510.00615.

Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. MEMENTO: Teaching LLMs to manage their own context, 2026. URL https://arxiv.org/abs/2604.09852.

Tianjian Li, Jingyu Zhang, William Jurayj, Xi Wang, Chuanyang Jin, Mehrdad Farajtabar, Eric Nalisnick, and Daniel Khashabi. Self-compacting language model agents, 2026a. URL https://arxiv.org/abs/2606.23525.

Xiaochuan Li, Ryan Ming, Meng Chu, Shuai Shao, Rong Jin, and Chenyan Xiong. ACM: Agentic context management for long horizon tasks, 2026b. URL https://arxiv.org/abs/2607.23809.

Yujiang Li, Zhenyu Hou, Yi Jing, Jie Tang, and Yuxiao Dong. CompactionRL: Reinforcement learning with context compaction for long-horizon agents, 2026c. URL https://arxiv.org/abs/2607.05378.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Miao Lu, Weiwei Sun, Weihua Du, Zhan Ling, Xuesong Yao, Kang Liu, and Jiecao Chen. Scaling LLM multi-turn RL with end-to-end summarization-based context management, 2025. URL https://arxiv.org/abs/2510.06727.

Zhengxi Lu, Zhiyuan Yao, Zhuowen Han, Zi-Han Wang, Jinyang Wu, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Self-distilled agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2605.15155.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

MiMo Team. MiMo-V2-Flash technical report, 2026. URL https://arxiv.org/abs/2601.02780.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems, 2023. URL https://arxiv.org/abs/2310.08560.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 31210–31227. PMLR, 2023. URL https://proceedings.mlr.press/v202/shi23a.html.

Guangya Wan, Mingyang Ling, Xiaoqi Ren, Rujun Han, Sheng Li, and Zizhao Zhang. COMPASS: Enhancing agent long-horizon reasoning with evolving context, 2025. URL https://arxiv.org/abs/2510.08790.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization, 2025. URL https://arxiv.org/abs/2509.13313.

Wenkai Yang, Weijie Liu, Ruobing Xie, Kai Yang, Saiyong Yang, and Yankai Lin. Learning beyond teacher: Generalized on-policy distillation with reward extrapolation, 2026. URL https://arxiv.org/abs/2602.12125.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pp. 20744–20757, 2022a. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models, 2022b. URL https://arxiv.org/abs/2210.03629.

Tianzhu Ye, Li Dong, Xun Wu, Shaohan Huang, and Furu Wei. On-policy context distillation for language models, 2026. URL https://arxiv.org/abs/2602.12275.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. MemAgent: Reshaping long-context LLM with multi-conv RL-based memory agent, 2025. URL https://arxiv.org/abs/2507.02259.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026a. URL https://arxiv.org/abs/2510.04618.

Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, and Yizhe Zhang. Embarrassingly simple self-distillation improves code generation, 2026b. URL https://arxiv.org/abs/2604.01193.

Xuan Zhang, Longtao Zheng, Cunxiao Du, Bo An, and Xin Dong. AutoCompact: Learning when to compact context in long-horizon coding agents, 2026c. URL https://autocompact.github.io/.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models, 2026. URL https://arxiv.org/abs/2601.18734.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

APPENDIX CONTENTS

A Terminal-Bench 2.0 inference configuration **14**

B GRPO objective **16**

B.1 Optimization hyperparameters 16

C Details on the distillation objective **16**

D Additional Ablation Results **19**

D.1 Distillation without GRPO 19

E Experimental Details **19**

E.1 Benchmarks and data splits 19

E.2 Rollout configuration 20

E.3 Models and systems 20

E.4 Evaluation 20

E.5 CCM Prompt Templates 20

F Qualitative Trajectory Analysis **22**

F.1 WebShop 22

F.2 Endless Terminals 23

Table 5: Detailed Terminal-Bench 2.0 pass@1 and mean cumulative token usage over 89 tasks. Baseline denotes full-history prompting.

	Claude Sonnet 4.6		Claude Opus 4.6		GLM-5		Kimi K3	
Metric	Baseline	CCM	Baseline	CCM	Baseline	CCM	Baseline	CCM
Evaluation								
Success, n (%)	**38 (42.7)**	28 (31.5)	**52 (58.4)**	36 (40.4)	**31 (34.8)**	26 (29.2)	56 (62.9)	**57 (64.0)**
Average turns	20.88	30.33	21.60	26.00	25.90	32.50	14.49	19.02
Tokens per task								
Uncached input	14,955	48,056	23,914	86,052	480,325	75,481	102	27,039
Cache-read input	408,900	27,135	410,180	0	–	–	186,969	20,685
Cache-write input	27,151	936	30,440	0	–	–	14,776	1,128
Total input	451,006	76,126	464,533	86,052	480,325	75,481	201,846	48,852
Total output	13,265	21,084	12,662	26,286	10,808	15,184	12,659	35,354
Cache-read share (%)	90.7	35.6	88.3	0.0	–	–	92.6	42.3

A TERMINAL-BENCH 2.0 INFERENCE CONFIGURATION

We evaluate pass@1 on the fixed 89-task Terminal-Bench 2.0 set. The baseline uses the standard Terminus-2 agent and receives the complete interaction history at every turn. Under CCM, each prompt instead contains the original task, the memory written by the model on the preceding turn, and the latest terminal observation. For all four models, we use temperature 0.6, allow at most 8,192 generated tokens per turn, and limit each task to 50 turns. The CCM system prompt and per-turn input template are shown in Listings 1 and 2. The official task instruction is inserted verbatim into the {`instruction`} field.

Table 5 reports pass@1 performance and cumulative token usage. Token counts are provider-reported means per task and are summed across all model calls in an episode. Claude Sonnet 4.6, Claude Opus 4.6, and Kimi K3 used provider-side prompt caching. GLM-5 was evaluated without prompt caching because Amazon Bedrock did not support it for this model.

```
You are an AI assistant tasked with solving command-line tasks in a Linux environment. You
    will be given a task description and
the output from previously executed commands. Your goal is to solve the task by providing
   batches of shell commands.

IMPORTANT MEMORY BEHAVIOR: You have a self-managed context window. You will not see the
   full previous trajectory. On each turn,
you will receive the task description, your previous context_window, and the latest
   terminal output. Use context_window to
preserve durable state needed for future turns.

Format your response as JSON with the following structure:

{
  context_window: Compact durable memory to carry into the next turn. Include task goal,
   working directory, key files
  inspected, important command results, edits made, tests run, unresolved blockers, and
   verification status. Do not include long
  terminal transcripts or private reasoning.,
  analysis: Analyze the current state based on the terminal output provided. What do you
   see? What has been accomplished? What
  still needs to be done?,
  plan: Describe your plan for the next steps. What commands will you run and why? Be
   specific about what you expect each
  command to accomplish.,
  commands: [
    {
      keystrokes: ls -la\n,
      duration: 0.1
    },
    {
      keystrokes: cd project\n,
      duration: 0.1
    }
  ],
  task_complete: true
```

```
}

Required fields:
- context_window: Compact durable memory to carry into the next turn. Include task goal,
   working directory, key files
inspected, important command results, edits made, tests run, unresolved blockers, and
   verification status. Do not include long
terminal transcripts, private reasoning, or speculative plans.
- analysis: Your analysis of the current situation
- plan: Your plan for the next steps
- commands: Array of command objects to execute

Optional fields:
- task_complete: Boolean indicating if the task is complete

Command object structure:
- keystrokes: String containing the exact keystrokes to send to the terminal (required)
- duration: Number of seconds to wait for the command to complete before the next command
   will be executed (defaults to 1.0 if
not present)

IMPORTANT: The text inside keystrokes will be used completely verbatim as keystrokes.
   Write commands exactly as you want them
sent to the terminal:
- You must end every command with a newline (\n) or it will not execute.
- For special key sequences, use tmux-style escape sequences:
  - C-c for Ctrl+C
  - C-d for Ctrl+D

The duration attribute specifies the number of seconds to wait for the command to complete
    (default: 1.0) before the next
command will be executed. On immediate tasks (e.g., cd, ls, echo, cat) set a duration of
   0.1 seconds. On commands (e.g., gcc,
find, rustc) set a duration of 1.0 seconds. On slow commands (e.g., make, python3 [long
   running script], wget [file]) set an
appropriate duration as you determine necessary.

It is better to set a smaller duration than a longer duration. It is always possible to
   wait again if the prior output has not
finished, by running {keystrokes: , duration: 10.0} on subsequent requests to wait longer.
    Never wait longer than 60
seconds; prefer to poll to see intermediate result status.

Important notes:
- Each command's keystrokes are sent exactly as written.
- Do not include extra whitespace before or after the keystrokes unless it is part of the
   intended command.
- Extra text before or after the JSON will generate warnings but be tolerated.
- The JSON must be valid; use proper escaping for quotes and special characters within
   strings.
- The commands array can be empty when waiting.
```

Listing 1: CCM system prompt used for all four TerminalBench-2 models.

```
Task Description:
{instruction}

Previous context_window:
{context_window or (empty)}

Current terminal state:
Latest Terminal Output:
{latest_terminal_output or (none yet)}

Return the required JSON object for the next turn.
```

Listing 2: CCM per-turn input template. Braced fields are populated from the current task state.

B GRPO OBJECTIVE

For each task, GRPO samples a group of G student trajectories and computes group-normalized environment advantages,

$$A^{(i)} = \frac{R^{(i)} - \mu_R}{\sigma_R + \varepsilon_{\text{std}}}. \tag{4}$$

Using the importance ratio

$$r_t^{(i)} = \frac{\pi_\theta\big(y_t^{(i)} \mid s_t^{(i)}\big)}{\pi_{\theta_{\text{old}}}\big(y_t^{(i)} \mid s_t^{(i)}\big)}, \tag{5}$$

the clipped policy objective is

$$L_{\text{PG}} = -\operatorname{Agg}\Big[\min\big(r_t^{(i)} A^{(i)},\ \text{clip}(r_t^{(i)}, 1-\epsilon, 1+\epsilon)\, A^{(i)}\big)\Big], \tag{6}$$

where Agg denotes the masked token mean over valid student-generated tokens. Our implementation additionally uses dual clipping for sufficiently negative advantages. A reference-policy regularizer is applied as

$$L_{\text{refKL}} = \operatorname{Agg}\Big[D_{\text{KL}}\big(\pi_\theta(\cdot \mid s_t) \,\|\, \pi_{\text{ref}}(\cdot \mid s_t)\big)\Big], \tag{7}$$

implemented using a sampled KL estimator.

B.1 OPTIMIZATION HYPERPARAMETERS

Table 6 reports the shared optimization configuration. CCM + GRPO and CCM + GRPO + Distillation use identical GRPO hyperparameters. The latter additionally performs privileged full-history teacher scoring and applies the distillation loss. Its distillation-specific hyperparameters are $\lambda_{\text{distill}} = 0.01$ and $\beta = 5$.

Table 6: Optimization hyperparameters used for WebShop and Endless Terminals.

Hyperparameter	WebShop	Endless Terminals
Training steps	100	60
Tasks per step	32	32
Rollouts per task	8	8
Trajectories per step	256	256
PPO mini-batch size	32	32
Micro-batch size per GPU	1	1
PPO epochs	1	1
Learning rate	1×10^{-6}	1×10^{-6}
LR schedule	Constant	Constant
Warmup steps	0	0
Adam betas	$(0.9, 0.999)$	$(0.9, 0.999)$
Weight decay	0.01	0.01
Gradient clipping	1.0	1.0
PPO clip ratio	0.2	0.2
Dual-clip coefficient	3.0	3.0
Entropy coefficient	0.001	0.0
Reference-KL coefficient	0.01	0.01
KL estimator	Low-variance	Low-variance
Context-SDAR coefficient λ	0.01	0.01
Gate sharpness β	5.0	5.0

C DETAILS ON THE DISTILLATION OBJECTIVE

This section explains our choice of the SDAR (Lu et al., 2026) objective, derives its gradient, and establishes its connection to on-policy distillation (OPD). Throughout, teacher probabilities are evaluated on student-generated action tokens using privileged full-history conditioning. The distillation objective supervises action tokens, while GRPO provides task-level supervision for both memory and action generation.

Motivation. The full interaction history provides information that may be missing from the student's compact memory. Comparing the probability of the same action token under these two contexts therefore provides a training signal for behavior favored under full-history conditioning. The teacher is a frozen copy of the student's initial model, with weights held fixed throughout training; it is not guaranteed to be stronger than the updated student. Access to the full history does not guarantee that every teacher judgment is correct, and large teacher–student probability gaps need not indicate proportionally reliable supervision.

We therefore adopt the gap-gated distillation objective of SDAR (Lu et al., 2026). It converts the teacher–student log-probability gap into a bounded gating weight, allowing full-history guidance to modulate token-level likelihood gradients while retaining GRPO as the task-level objective. The gate gives greater weight to tokens assigned higher probability under full-history conditioning and limits the gradient multiplier associated with extreme probability disagreements. This choice does not directly minimize reverse KL; its relationship to OPD is established below.

Objective and implemented gradient. Let

$$\ell^S_{i,n,t}(\theta) = \log \pi_\theta(a_{i,n,t} \mid s^C_{i,n,t}), \qquad \ell^T_{i,n,t} = \log \pi_{\theta_0}(a_{i,n,t} \mid s^+_{i,n,t}),$$

where θ_0 denotes the student's initial parameters before training. The teacher weights are fixed at θ_0 for the entire training run. Student and teacher probabilities are evaluated on the same sampled action token, conditioned on the compact context and privileged full-history context, respectively. Teacher scores are detached from gradient computation.

We define the detached log-probability gap and gating weight as

$$\Delta_{i,n,t} = \mathrm{sg}\left[\ell^T_{i,n,t} - \ell^S_{i,n,t}(\theta)\right], \tag{8}$$

$$g_{i,n,t} = \mathrm{sg}[\sigma(\beta \Delta_{i,n,t})], \tag{9}$$

where sg denotes stop-gradient, σ is the logistic sigmoid, and $\beta > 0$ controls gate sharpness. Let $M_{i,n,t}$ indicate a valid action token with available student and teacher scores, and let $Z = \sum_{i,n,t} M_{i,n,t}$. For a sampled batch with $Z > 0$, the distillation loss is

$$L_{\mathrm{distill}}(\theta) = -\frac{1}{Z} \sum_{i,n,t} M_{i,n,t} g_{i,n,t} \ell^S_{i,n,t}(\theta). \tag{10}$$

Batches without eligible action tokens contribute no distillation loss.

The sampled tokens, contexts, and masks are fixed during backpropagation. Because the gating weights are detached,

$$\nabla_\theta L_{\mathrm{distill}} = -\frac{1}{Z} \sum_{i,n,t} M_{i,n,t} g_{i,n,t} \nabla_\theta \ell^S_{i,n,t}(\theta). \tag{11}$$

The teacher therefore affects the distillation gradient through the gating weights, without requiring gradients through the teacher model. Positive-gap tokens have $g_{i,n,t} > 1/2$, whereas negative-gap tokens have $g_{i,n,t} < 1/2$. Negative-gap tokens retain positive gating weights, scaling down the magnitude of their individual distillation gradients rather than reversing their likelihood-gradient direction. These statements concern individual token gradients; the net probability change also depends on other tokens and the GRPO update through shared model parameters.

Equivalence to the SDAR loss. SDAR expresses its sampled-token objective using the weighted teacher–student log-probability difference:

$$\widetilde{L}_{\mathrm{distill}} = \frac{1}{Z} \sum_{i,n,t} M_{i,n,t} g_{i,n,t} \left[\ell^T_{i,n,t} - \ell^S_{i,n,t}(\theta)\right]. \tag{12}$$

This expression differs from Eq. (10) by

$$C = \frac{1}{Z} \sum_{i,n,t} M_{i,n,t} g_{i,n,t} \ell^T_{i,n,t}.$$

Since both the teacher scores and gating weights are detached, C has zero backpropagated gradient. The two expressions therefore produce identical distillation gradients on the same batch. We omit C from the displayed objective to make the weighted likelihood update explicit. This is a statement about backpropagation at a given update: the numerical value of C may change when scores or gates are recomputed.

Reverse-KL gradient in OPD. To compare the local token update with OPD, fix a student context s and its corresponding privileged context s^+. Write

$$p_\theta(a) = \pi_\theta(a \mid s), \qquad q(a) = \pi_{\theta_0}(a \mid s^+), \qquad \delta_\theta(a) = \log q(a) - \log p_\theta(a).$$

Here q is fixed, and both distributions are assumed positive on the token support under consideration.

At these fixed contexts, the reverse-KL objective is

$$J_{\mathrm{RKL}}(\theta) = D_{\mathrm{KL}}(p_\theta \| q) = \sum_a p_\theta(a) \left[\log p_\theta(a) - \log q(a)\right]. \tag{13}$$

Differentiating both the probability weights and the log-probabilities gives

$$\nabla_\theta J_{\mathrm{RKL}} = \sum_a \nabla_\theta p_\theta(a) \left[\log p_\theta(a) - \log q(a) + 1\right] \tag{14}$$

$$= -\mathbb{E}_{a\sim p_\theta}\left[\delta_\theta(a) \nabla_\theta \log p_\theta(a)\right]. \tag{15}$$

The constant term vanishes because

$$\mathbb{E}_{a\sim p_\theta}\left[\nabla_\theta \log p_\theta(a)\right] = \nabla_\theta \sum_a p_\theta(a) = 0. \tag{16}$$

Thus, OPD uses the signed log-probability gap as the coefficient of the negative likelihood gradient.

Expected SDAR update and its connection to OPD. For a token a sampled from the current student at the fixed context, the backpropagated SDAR gradient is

$$\widehat{G}_{\mathrm{SDAR}}(\theta; a) = -\sigma(\beta \delta_\theta(a)) \nabla_\theta \log p_\theta(a).$$

Define its expectation as

$$G_{\mathrm{SDAR}}(\theta) = \mathbb{E}_{a\sim p_\theta}\left[\widehat{G}_{\mathrm{SDAR}}(\theta; a)\right]. \tag{17}$$

This is the expectation of the implemented sampled-token gradient. It should not be confused with fully differentiating a current-policy expectation of the sampled-token loss, which would introduce additional sampling-distribution terms.

Using Eq. (16), we can subtract $1/2$ from the gating weight without changing the expected gradient:

$$G_{\mathrm{SDAR}} = -\mathbb{E}_{a\sim p_\theta}\left[\left(\sigma(\beta\delta_\theta(a)) - \frac{1}{2}\right) \nabla_\theta \log p_\theta(a)\right] \tag{18}$$

$$= -\frac{1}{2}\mathbb{E}_{a\sim p_\theta}\left[\tanh\left(\frac{\beta\delta_\theta(a)}{2}\right) \nabla_\theta \log p_\theta(a)\right]. \tag{19}$$

Consequently, the effective token weights in the expected OPD and SDAR gradients are

$$w_{\mathrm{OPD}}(\delta) = \delta, \qquad w_{\mathrm{SDAR}}(\delta) = \sigma(\beta\delta) - \frac{1}{2} = \frac{1}{2}\tanh(\beta\delta/2). \tag{20}$$

The centered SDAR weight is monotone, has the same sign as δ, and is bounded between $-1/2$ and $1/2$. It therefore preserves the sign and ordering of token-level teacher preferences while compressing their magnitudes. Although the implemented gate is positive, its expected on-policy gradient admits this equivalent signed representation.

For $|\beta\delta| \ll 1$, a Taylor expansion gives

$$w_{\mathrm{SDAR}}(\delta) = \frac{\beta}{4}\delta + O\big((\beta\delta)^3\big). \tag{21}$$

When this approximation holds for the tokens contributing to the expectation,

$$G_{\mathrm{SDAR}}(\theta) \approx \frac{\beta}{4}\nabla_\theta D_{\mathrm{KL}}(p_\theta \| q). \tag{22}$$

The expected SDAR update thus approaches a scaled OPD gradient locally for sufficiently small log-probability gaps. For large gaps, the effective weight saturates rather than growing linearly. Because this transformation reweights tokens nonlinearly, the aggregate SDAR and OPD gradients need not have the same direction away from the small-gap regime.

Table 7: Distillation-only ablation on the fixed 128-task WebShop evaluation. Both models are evaluated at checkpoint 100 under CCM. Partial score is the mean percentage of WebShop constraints satisfied.

Model	Success (%)	Solved	Partial score (%)	Turns
Qwen3-4B-Instruct	0.00	0/128	0.59	14.97
Qwen3-8B	0.78	1/128	5.27	14.11

Scope of the connection. The centering identity and local approximation characterize a token update at fixed contexts under sampling from the current student distribution. They do not differentiate through the generation of memories, preceding action tokens, or the distribution of visited environment histories. They also do not establish exact reverse-KL minimization for the complete training procedure.

In practice, rollouts are generated by a behavior-policy snapshot and may be reused after parameter updates. The exact on-policy identity then need not hold without appropriate sampling corrections. Token-dependent filtering and normalization by a random number of eligible tokens can also change the expected batch gradient. The derivation therefore characterizes the distillation update locally; it does not establish an exact identity for the full masked trajectory loss.

Finally, subtracting $1/2$ preserves the expected gradient under the stated assumptions, but generally changes a finite-batch gradient. Our implementation retains the positive sigmoid gate; centering is used only for analysis. The bounded gate limits the per-token gradient multiplier, not the norm of the underlying model gradient, and does not guarantee monotonic KL reduction or task improvement.

D Additional Ablation Results

D.1 Distillation without GRPO

We evaluate whether auxiliary privileged distillation can train a CCM policy without the accompanying GRPO objective. These runs retain same-trajectory privileged full-history scoring on student action tokens but set the GRPO loss weight to zero. Because the distillation objective operates only on action tokens, memory tokens receive no direct training signal in this condition.

As shown in Table 7, both distillation-only policies achieve nearly zero exact success at checkpoint 100: Qwen3-4B-Instruct solves no tasks, while Qwen3-8B solves only one of 128. In these WebShop experiments, privileged action-token distillation therefore does not replace verifier-driven GRPO. GRPO provides trajectory-level supervision to both memory and action tokens, while privileged distillation is effective only as an auxiliary objective alongside GRPO in the main experiments.

E Experimental Details

E.1 Benchmarks and data splits

WebShop. We use the WebShop text environment with the small product configuration. Training tasks are sampled from all WebShop task IDs greater than or equal to 500. Evaluation uses the fixed set of task IDs 0–127. We train for 100 policy updates and evaluate saved checkpoints separately using pass@1, temperature 0.4, seed 42, and a 15-turn episode limit.

We use a shaped reward that combines WebShop's partial task score with a larger bonus for exact task completion. Let $r_{\text{partial}} \in [0, 1]$ denote the native WebShop task score and let $s \in \{0, 1\}$ indicate exact success. The training reward is

$$R_{\text{WebShop}} = 2r_{\text{partial}} + 8s. \tag{23}$$

Thus, unsuccessful trajectories receive rewards between 0 and 2 according to their partial progress, whereas an exactly successful trajectory receives reward 10.

Endless Terminals. We use the fixed Endless Terminals split containing 2,083 training tasks, 100 validation tasks, and 300 held-out test tasks. Training lasts for 60 policy updates. Final results are obtained by evaluating saved checkpoints on the fixed 300-task test set with pass@1, seed 42, temperature 0.6, and a 16-turn episode limit. The environment reward is binary task success. Invalid actions receive an auxiliary penalty with coefficient 0.1.

E.2 Rollout configuration

Table 8 summarizes the rollout and sequence limits. Thinking is disabled for all reported experiments. The self-managed student emits one memory block followed by one environment-action block at every turn.

Table 8: Rollout and sequence-length configuration. WebShop uses 1,024 generation tokens for Qwen3-4B-Instruct and 2,048 for Qwen3-8B.

Hyperparameter	WebShop	Endless Terminals
Maximum turns	15	16
Sampling temperature	1.0	0.6
Top-p	1.0	1.0
Top-k	Unrestricted	Unrestricted
Student prompt limit	4,096	16,384
Teacher prompt limit	32,768	16,384
Generation tokens per turn	1,024 / 2,048	4,096
Maximum model length	32,768	20,480
Thinking enabled	No	No

E.3 Models and systems

WebShop experiments use Qwen3-4B-Instruct-2507 and Qwen3-8B. Endless Terminals experiments use Qwen3-8B. Training is implemented in verl with FSDP policy updates and colocated vLLM rollout generation. Each run uses one node with four H100 or H200 GPUs.

E.4 Evaluation

WebShop checkpoint evaluation uses temperature 0.4, seed 42, and the fixed 128-task set. Endless Terminals use temperature 0.6, seed 42, and the fixed 300-task held-out test set.

E.5 CCM Prompt Templates

The following listings reproduce the prompts used for Continuous Context Management. WebShop uses a single per-turn prompt containing the task, retained memory, current observation, and admissible actions. Endless Terminals uses a fixed system prompt, followed by the original task as a separate user message and a per-turn user message containing the retained memory and newest terminal observation. Braced fields are populated at runtime.

Listing 3: WebShop CCM per-turn prompt template.

```
You are an expert autonomous agent operating in the WebShop e-commerce environment.
Your task is to: {task_description}.

MEMORY:
- Memory is the persistent record of useful information from earlier shopping
  turns. The original task is always shown separately.
- Your previous accepted memory appears below. It is empty on the first turn.
- Every response must contain exactly one <memory> </memory> block.
- Inside that block, retain concrete task constraints, product evidence,
  selected options, and unresolved requirements that could be needed later.
- If no useful information changed, repeat the previous memory unchanged.
- Never write instructions, labels, or generic filler in memory.

Previous memory:
<memory>{context_window}</memory>
```

```
You are now at step {current_step}.
Your current observation is: {current_observation}.
Your admissible actions are:
[
{available_actions}
].

Write the updated <memory> </memory> block. Then choose exactly one admissible
action and write it inside <action> </action>.
```

Listing 4: Endless Terminals CCM system prompt.

```
You are a highly capable Linux terminal agent operating strictly via a single-shell-command
   interface.
Goal: complete the user's task.

CRITICAL MEMORY REQUIREMENT:
- EVERY RESPONSE MUST BEGIN WITH <memory> AND INCLUDE EXACTLY ONE </memory> CLOSING TAG.
- THIS INCLUDES THE FIRST TURN, EVERY INTERMEDIATE TURN, AND THE FINAL TURN.
- A RESPONSE CONTAINING <command> OR <action>done</action> WITHOUT <memory>...</memory> IS
   INVALID AND WILL NOT EXECUTE.
- NEVER OUTPUT A BARE COMMAND. YOUR FIRST OUTPUT CHARACTERS MUST BE: <memory>

Rules:
- Every response must use exactly one of these two forms:

  <memory>FACTS_TO_REMEMBER</memory>
  <command>THE_SINGLE_SHELL_COMMAND</command>

  OR

  <memory>FACTS_TO_REMEMBER</memory>
  <action>done</action>

- Do not output a bare <command> or <action> block.
- Do not use <command_window> or any other wrapper names.
- Don't use interactive commands and confirmations; use non-interactive flags.
- Prefer simple, robust CLI tools; write files explicitly when needed.
- If you believe the task is solved, use the second form with <action>done</action>.
- Hint: you might want to run commands interactively to see the output and then write the
   command. Don't just pipe the commands.
- Only your first command in command tags will be executed. So don't respond with multiple
   commands.
- Hint: Verify your solution once you are done. Eg: you can use cat to see the input and the
    output.
- Do not just write long bash scripts. Write the commands that you would write in a terminal
   .

IMPORTANT MEMORY BEHAVIOR:
- Write in <memory> the facts from the task and terminal work that should be available on
   the next turn.
- The next turn will not receive the earlier conversation. It receives only the original
   task, your latest memory, and the newest terminal
observation.
- Update the memory as work progresses so it reflects the current durable state.
- Memory describes the state before the command in the same response executes.
- Record completed work only when confirmed by an earlier terminal observation.
- Do not describe the accompanying command as already completed.
```

In Endless Terminals, the original task instruction is supplied verbatim as a separate user message. The following additional user message is then reconstructed at every turn:

Listing 5: Endless Terminals CCM per-turn user-message template.

```
Previous memory:
<memory>{memory}</memory>

Current observation:
{observation}

RESPONSE REQUIREMENT: Begin your response with <memory>. Write the complete updated <memory
   >...</memory> block, then exactly one <command>...</
command> or <action>done</action>. Never return a bare command.
```

Table 9: Outcomes for the matched WebShop trajectory. Exact denotes exact task success; partial is the WebShop constraint-matching score.

Model	Condition	Exact	Partial	Turns
Qwen3-4B-Instruct	Untrained full history	No	0.000	15
	Untrained CCM	No	0.000	15
	CCM + GRPO	No	0.857	15
	CCM + GRPO + Distillation	Yes	1.000	5
Qwen3-8B	Untrained full history	No	0.000	12
	Untrained CCM	No	0.000	15
	CCM + GRPO	No	0.857	10
	CCM + GRPO + Distillation	Yes	1.000	5

F Qualitative Trajectory Analysis

We present condensed matched trajectories to illustrate behavioral differences among the evaluated conditions. For each trajectory, we retain the initial turn, turns that materially change the retained memory or environment state, the first instance of each repeated failure mode, the final effective action, and the verifier outcome. Consecutive turns with equivalent behavior are summarized explicitly. Bracketed omissions inside excerpts are editorial and remove only irrelevant or repeated output.

F.1 WebShop

Task. The agent must purchase a men's shirt priced below $50 that satisfies the requested material, fit, sleeve, color, and size constraints. The matched task requires cotton spandex, classic fit, short sleeves, charcoal heather (light), and size 1x.

Untrained policies. The untrained full-history policies repeatedly issue broad searches without reaching a valid purchase. The untrained CCM policies preserve the task constraints in memory but continue paging through irrelevant results. Neither condition purchases a product satisfying the requested options.

CCM + GRPO failure. For Qwen3-4B-Instruct, CCM + GRPO finds the target product and opens its product page. Its memory, however, remains effectively unchanged after the initial search:

```
<memory>
Task: Find machine wash men's dress shirts with cotton spandex,
classic fit, short sleeve, color: charcoal heather (light),
size: 1x, price lower than $50.00.
Unresolved: No search or product options have been generated yet.
</memory>
<action>click[b07hrfsnl4]</action>
```

The resulting product page exposes both required options:

```
size: [..., 1x, ...]
color: [..., charcoal heather (light), ...]
Nautica Men's Solid Crew Neck Short-Sleeve Pocket T-Shirt
Price: $16.05 to $40.98
Buy Now
```

The policy selects the color but then repeats the same color-selection action through the remaining intermediate turns instead of selecting size 1x:

```
<action>click[charcoal heather (light)]</action>
```

At the final turn it purchases the product with only the color selected. The environment assigns partial score 0.857 rather than exact success:

```
Purchased
```

```
asin: B07HRFSNL4
options: {"color": "charcoal heather (light)"}
Reward: 0.8571428571428571
```

The Qwen3-8B CCM + GRPO trajectory exhibits the same high-level failure: it reaches the target product but purchases without satisfying every required option, receiving partial score 0.857.

CCM + GRPO + Distillation success. The distillation-augmented Qwen3-4B-Instruct policy records the task constraints and the current stage of the interaction. Its five actions form the required search–product–color–size–purchase sequence:

```
Turn 1: <action>search[cotton men's dress shirts cotton spandex
        classic fit short sleeve color charcoal heather (light)
        size 1x price under 50.00]</action>
Turn 2: <action>click[b07hrfsn14]</action>
Turn 3: <action>click[charcoal heather (light)]</action>
Turn 4: <action>click[1x]</action>
Turn 5: <action>click[buy now]</action>
```

The memory changes with the option-selection state. For example, immediately before the size action it records:

```
<memory>
Task: Find machine wash men's dress shirts with cotton spandex,
classic fit, short sleeve, color: charcoal heather (light),
size: 1x, price lower than $50.00.
Current step: 4 (Select size).
Action: Select size 1x.
Size selected: 1x.
Size constraint met: yes.
</memory>
<action>click[1x]</action>
```

The final observation confirms that both required options were selected:

```
Purchased
asin: B07HRFSNL4
options: {"color": "charcoal heather (light)", "size": "1x"}
Reward: 1.0
```

Qwen3-8B follows the same five-action sequence. Its memory uses a structured representation that retains the constraints and selected actions:

```
<memory>
{
  "constraints": {
    "type": "men's dress shirts",
    "material": "cotton spandex",
    "fit": "classic fit",
    "sleeve": "short sleeve",
    "color": "charcoal heather (light)",
    "size": "1x",
    "price_limit": 50.00
  },
  "actions_taken": [
    "click[charcoal heather (light)]",
    "click[1x]"
  ]
}
</memory>
```

It likewise purchases the target product with both options selected and receives exact reward 1.0.

F.2 Endless Terminals

The two Endless Terminals examples expose complementary outcomes: one is solved only by CCM + GRPO + Distillation among the matched conditions, while the other is solved by CCM + GRPO but not by the distillation-augmented policy. They also contrast clean early termination with continued action after a verifier-valid state has been created.

Table 10: Outcomes for the two matched Endless Terminals trajectories.

Task	Condition	Success	Turns
4d41da7b	Untrained full history	No	2
	Untrained CCM	No	2
	CCM + GRPO	No	16
	CCM + GRPO + Distillation	Yes	2
beff73f4	Untrained full history	Yes	4
	Untrained CCM	No	16
	CCM + GRPO	Yes	16
	CCM + GRPO + Distillation	No	16

F.2.1 PERMISSION-AUDIT SETUP: DISTILLATION-AUGMENTED CCM SUCCESS

Task. The agent must create a shell-script stub, set directory and file modes to 750 and 740, respectively, and write a three-line permission-verification log with exact contents.

Failure modes. The untrained full-history policy performs nearly all required operations, but constructs the third log line from `ls -l`. The resulting permission string contains a trailing SELinux marker, `-rwxr-----.`, rather than the required `-rwxr-----`. The untrained CCM policy creates only the directory, then records in memory that the unexecuted file and log operations have already been completed and terminates. CCM + GRPO initially creates a valid state, but its memory remains unchanged:

```
<memory>Initial state: No commands have been executed yet.</memory>
```

It continues issuing commands through turn 16 and ultimately overwrites the verification log with only its third line. The verifier therefore reports that the file contains one line rather than three.

CCM + GRPO + Distillation. The distillation-augmented policy performs the complete setup in one command:

```
<memory>Need to create a specific directory structure and files
with specific permissions and content.</memory>
<command>mkdir -p /home/user/project_utils/secure_scripts &&
chmod 750 /home/user/project_utils/secure_scripts &&
echo -e '#!/bin/bash\necho "Backup script stub"' >
/home/user/project_utils/secure_scripts/backup.sh &&
chmod 740 /home/user/project_utils/secure_scripts/backup.sh &&
echo -e "$(stat -c "%a %n" /home/user/project_utils/secure_scripts)\n\
$(stat -c "%a %n" /home/user/project_utils/secure_scripts/backup.sh)\n\
-rwxr----- /home/user/project_utils/secure_scripts/backup.sh" >
/home/user/perm_check.log</command>
```

After observing a zero exit code, it requests completion on turn 2:

```
<memory>Need to create a specific directory structure and files
with specific permissions and content.</memory>
<action>done</action>

... [100%]
3 passed in 0.01s
```

This trajectory provides a clean early-termination example: the policy reaches the verifier-valid state with one environment action and does not issue any redundant commands.

F.2.2 GIT TRACKED-FILE EXPORT: CCM + GRPO SUCCESS

Task. The agent must query Git inside `/home/user/demo_repo` and write all tracked paths, in Git's returned order and with a final newline, to `/home/user/git_files.log`.

Untrained full-history baseline. After two commands fail because they run outside the repository, the baseline changes directory and succeeds:

```
<command>cd /home/user/demo_repo && git ls-files |
tee /home/user/git_files.log</command>

README.md
scripts/run.sh
src/main.py
```

It requests completion on turn 4, and all five verifier tests pass.

CCM + GRPO. The first CCM + GRPO command also runs in the wrong directory and fails. The second turn corrects the working directory and queries Git:

```
<memory>
Initial state: No commands have been executed yet.
Task: List tracked files in /home/user/demo_repo and save to
/home/user/git_files.log.
</memory>
<command>cd /home/user/demo_repo && git ls-files</command>

Command succeeded. Output:
README.md
scripts/run.sh
src/main.py
```

The following memory preserves the command result, and the policy writes the three paths to the requested file:

```
<memory>
Task: List tracked files in /home/user/demo_repo and save to
/home/user/git_files.log.
Output of git ls-files: README.md scripts/run.sh src/main.py
</memory>
<command>echo -e "README.md\nscripts/run.sh\nsrc/main.py" >
/home/user/git_files.log</command>
```

Turns 4–16 repeatedly rewrite the same valid content instead of terminating. The final state passes all verifier tests:

```
..... [100%]
5 passed in 0.02s
```

CCM + GRPO + Distillation failure. The distillation-augmented policy initially mistakes a wrong working directory for an absent repository and unnecessarily reinitializes Git. It then repeatedly uses the invalid `-n` option with `git ls-files`. Although turn 15 finally obtains the correct tracked paths, turn 16 again uses the invalid option and leaves an incorrect output file. The verifier reports three failed tests, including incorrect contents and a missing trailing newline.

Interpretation. The two examples show that neither trained CCM objective dominates on every task. They also show that termination behavior varies substantially across conditions: the distillation-augmented policy terminates immediately after its successful action in the permission task, whereas CCM + GRPO establishes a correct state early in the Git task but continues issuing redundant commands until the turn limit.