DIFFERENTIABLE HARNESS

Anonymous authors
Paper under double-blind review

ABSTRACT

An agent's harness defines the instructions, tools, memory, context management, and control flow surrounding its language model. Existing methods improve harnesses by modifying discrete prompt text or agent code, which precludes gradient-based optimization. During generation, however, the model accesses harness text through the internal cache created during prefill. We introduce a *differentiable harness*: rather than modifying prompt text, we directly optimize the cache it induces, keeping model weights, visible prompt tokens, and tool implementations fixed. We train the cache using teacher-forced replay with cross-entropy and reverse KL regularization against the base distribution, augmented with optional clipped token unlikelihood on error steps. At deployment, an inference engine loads the learned cache artifact at registered token positions without altering the prompt. Across 22 task $\times$ harness rows spanning 15 benchmarks and three frozen model groups (dense, MoE, and hybrid attention/linear-recurrent architectures), differentiable harnesses improve task success rate by up to 20.8 percentage points over base agents (with relative gains up to 3.0$\times$) and consistently match or surpass handwritten harnesses. Beyond benchmark accuracy, action-level audits show that the learned cache reduces step-level tool execution errors by 30%, cuts step budget exhaustion by 60% on OSWorld, and avoids the conversational forgetting that degrades weight-adapted baselines.

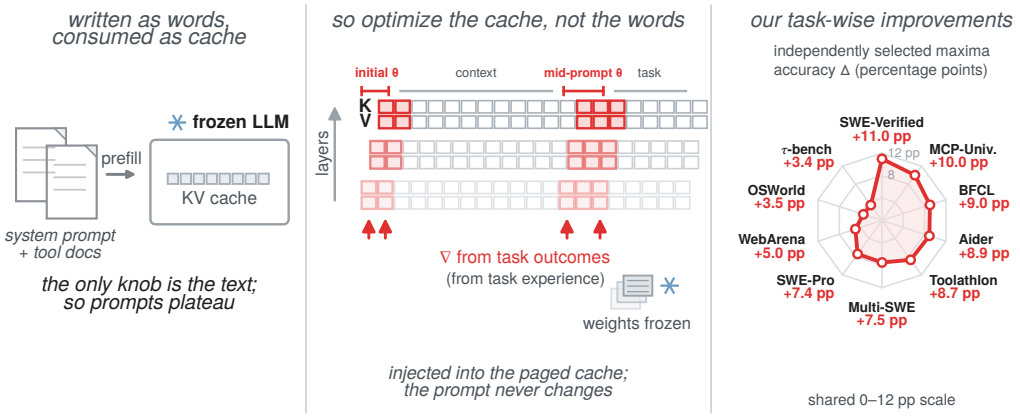
1 INTRODUCTION

An agent's harness is the runtime surrounding its model: prompts and tool documentation, executable tools, control flow, memory, context management, and stopping rules. These components determine what the agent can do and what information reaches the model at each step. Improving an agent therefore does not require retraining model weights; the harness itself is an effective target for optimization.

Existing work on automated harness improvement follows two broad directions. Executable harness evolution searches over the agent program: tool implementations, orchestration logic, and runtime policies. Methods such as Meta-Harness, AHE, RRSI, DGM, Live-SWE-agent, and Self-Harness evolve tool code and control flow (Lee et al., 2026; Lin et al., 2026; Han et al., 2026; Zhang et al., 2026b; Xia et al., 2025; Zhang et al., 2026a). Prompt and context optimization improves the information the agent reads and reuses: GEPA and RLMOpt optimize prompts, ACE evolves context memory, and Evo-Harness compiles execution experience into reusable skills (Agrawal et al., 2026; Satheesha et al., 2026; Zhang et al., 2026c; Tan et al., 2026; Wei et al., 2026). Both directions improve agents by modifying discrete artifacts around a frozen model.

Prompt optimization changes the model's internal representations only indirectly. When an attention layer processes harness text during prefill, it writes keys and values to the KV cache. A linear-attention layer updates a recurrent state. Later tokens use these cached representations to attend to earlier instructions (Figure 1a). Rewriting prompt text can only produce cache values reachable by some valid sequence of discrete tokens. Conversely, classical continuous prompt methods such as prefix tuning (Li & Liang, 2021) optimize virtual continuous vectors directly, but they cannot be applied to an agent's harness. First, optimizing randomly initialized continuous vectors from scratch causes severe representation collapse in complex multi-turn agent interactions, failing to complete tasks entirely (0/800 multi-turn; Table 11). Second, an agent's harness is not an opaque virtual prefix prepended at sequence start; it consists of structured procedural components—system instructions, multi-tool schemas, runtime interfaces, and dynamic mid-prompt reminders. Prepending abstract

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.



(a) **Obstacle** (b) **Mechanism** (c) **Task-wise gains**

Figure 1: **Train the cache, not the text. (a)** Harness text is consumed through the cache it creates. **(b)** We optimize selected cache entries at registered positions (covering initial segments as well as mid-prompt insertions, such as tool schemas or reminders) while keeping model weights and visible prompt tokens fixed. **(c)** Best observed accuracy gains on initial harness segments across ten agent benchmarks spanning all five workload families, shown on a shared 0–12 percentage-point scale. Each axis selects its strongest available setting independently.

continuous vectors at $t = 0$ cannot steer these structured mid-prompt segments, nor can it exploit the rich semantic scaffolding of existing human-engineered harnesses.

We instead introduce *differentiable harnesses*: we take an engineered harness (comprising system instructions, tool documentation, memory, and runtime scaffolding) and optimize the continuous internal cache it induces during prefill (Figure 1b). By initializing trainable parameters directly from the forward pass of the written harness text, optimization anchors on activations that the model already supports, avoiding representation collapse while keeping tool interfaces, visible text, and model weights fixed. This parameterization covers standard keys and values, compressed cache latents, and linear recurrent states across arbitrary token offsets (Figure 2), supporting both initial prefixes and mid-prompt segments (such as tool definitions, retrieved memory, or recurring reminders placed between turns).

During training, we supervise the cache using teacher-forced replay from positive demonstrations and error rollouts: successful actions provide targets for cross-entropy, while failed trajectories supply tokens to discourage via unlikelihood, regularized by exact full-vocabulary reverse KL against the base model distribution. At deployment, an inference engine loads the optimized cache tensors into paged memory at registered token offsets without modifying prompt text, enabling per-request artifact swapping without reloading model weights. Supporting details on related work, limitations, theoretical derivations, and benchmark details are provided in Appendices A–E.

Contributions.

- **A differentiable cache representation of the harness.** We optimize the internal KV-cache or linear-state entries induced by harness segments while keeping model weights, visible prompt tokens, and executable tools fixed. The method covers initial and mid-prompt segments and complements discrete harness search.
- **An end-to-end training and serving system.** We connect cache capture from production harnesses, gradient-based optimization through frozen models, and paged-cache injection at request dispatch under unchanged prompts, enabling high-concurrency multi-tenant deployment.
- **Empirical evaluation across diverse architectures.** Across 22 task $\times$ harness rows spanning 15 benchmarks and three frozen model groups (dense, MoE, and hybrid attention/linear-recurrent architectures), differentiable harnesses improve task success by up to 20.8 percentage points over base agents (up to 3.0$\times$ relative gain) and consistently match or surpass handwritten harnesses. Ablations indicate that adaptations reside primarily in Value representations, real harness initialization prevents representation collapse, and paged injection cuts cold serving latency by 68.9%.

2 METHOD

2.1 PARAMETRIC REPRESENTATION OF THE HARNESS CACHE

An agent's harness contributes stable text segments throughout context: system instructions, tool documentation, recurring reminders, and tool call blocks between turns. In standard prefill, these tokens pass through embedding, projection, and positional operators to produce internal cache states. Subsequent queries read earlier tokens through attention over stored keys and values, or through recurrent state updates in linear-attention layers.

Rather than modifying the discrete prompt text, we parameterize the cache over selected harness slots as a continuous function of trainable parameters θ. Specifically, for a query q_t at step t, the attention readout is

$$\mathrm{Attn}(q_t) \;=\; \sum_j \mathrm{softmax}_j\big(\gamma q_t^\top k_j\,\big)\, v_j,$$

where $\gamma = 1/\sqrt{d_h}$ is the head attention scale, and the sum spans the harness slots together with task and generated tokens. Over the selected harness slots $i \in \{0, \ldots, n-1\}$ at base offset p, we replace the token-derived cache with

$$(k_{p+i},\, v_{p+i}) \;=\; \mathcal{F}(\theta;\, p+i),$$

where θ denotes the free parameters and $\mathcal{F}$ represents the architecture-specific operator mapping θ to the effective keys and values. For linear-attention layers, θ directly parameterizes the recurrent boundary state $S_b = \mathcal{F}(\theta)$. All model weights W and visible prompt tokens remain frozen.

We support four cache parameterizations (Figure 2a–d), corresponding to the underlying attention and recurrent architectures:

Pre-RoPE key–value cache. When a harness segment is deployed at varying positions across conversation turns (such as mid-prompt tool schemas or recurring reminders) or when position-invariant transfer is desired, Pre-RoPE parameterizes the unrotated key $\widetilde{K}^{(\ell)}$:

$$\theta \;=\; \big\{\,(\widetilde{K}^{(\ell)},\, V^{(\ell)})\,\big\}_{\ell=1}^{L}, \qquad \widetilde{K}^{(\ell)},\, V^{(\ell)} \in \mathbb{R}^{n\times d},$$

where $d = h_{\mathrm{kv}} d_h$ is the total KV channel dimension across all h_{kv} heads. Rotary position embeddings (Su et al., 2024) are applied dynamically upon attention read:

$$\mathcal{F}(\theta;\, p+i) \;=\; \big(\, R_{p+i}\widetilde{K}_i^{(\ell)},\; V_i^{(\ell)}\,\big),$$

where $R_t \in \mathbb{R}^{d\times d}$ is the block-diagonal rotary matrix at absolute position t (acting independently on each head dimension), and $\widetilde{K}_i^{(\ell)}, V_i^{(\ell)} \in \mathbb{R}^d$ denote the i-th vector entries. This decouples semantic content from positional coordinates, allowing the same learned artifact to be spliced into different prompt locations or reused across dynamic offsets (Appendix D.3).

Post-RoPE key–value cache (GQA/MHA). In standard multi-head and grouped-query attention (Ainslie et al., 2023) where the harness occupies a fixed initial prefix, rotary position embeddings are baked directly into keys when written during prefill:

$$\theta \;=\; \big\{\,(K^{(\ell)},\, V^{(\ell)})\,\big\}_{\ell=1}^{L}, \qquad K^{(\ell)},\, V^{(\ell)} \in \mathbb{R}^{n\times d}.$$

Here the mapping $\mathcal{F}$ is the direct identity on read: $k_{p+i} = K_i^{(\ell)}$ and $v_{p+i} = V_i^{(\ell)}$, with rotation already baked into $K^{(\ell)}$. This parameterization yields $2ndL$ trainable scalar parameters. Because it requires zero runtime rotation overhead and matches native inference engine layouts (e.g., vLLM paged attention), we adopt it as the default configuration for fixed initial prefixes.

Compressed-latent attention cache (MLA). Multi-head Latent Attention (DeepSeek-AI et al., 2024) compresses keys and values into a shared low-rank latent vector $c_i \in \mathbb{R}^r$ ($r \ll 2d$) alongside a decoupled rotary key vector $k_i^{\mathrm{pe}} \in \mathbb{R}^{d_{\mathrm{pe}}}$. For L layers, θ parameterizes the compressed latents and rotary keys:

$$\theta \;=\; \big\{\,(c^{(\ell)},\, k^{\mathrm{pe},(\ell)})\,\big\}_{\ell=1}^{L}, \qquad c^{(\ell)} \in \mathbb{R}^{n\times r},\; k^{\mathrm{pe},(\ell)} \in \mathbb{R}^{n\times d_{\mathrm{pe}}}.$$

During attention execution, frozen decompression projections $W_{\mathrm{UK}} \in \mathbb{R}^{d\times r}$ and $W_{\mathrm{UV}} \in \mathbb{R}^{d\times r}$ reconstruct the full unrotated keys and values from c_i, while the rotary component k_i^{pe} provides position encoding:

$$\mathcal{F}(\theta;\, p+i) \;=\; \big(\,[W_{\mathrm{UK}} c_i^{(\ell)};\, k_i^{\mathrm{pe},(\ell)}],\; W_{\mathrm{UV}} c_i^{(\ell)}\,\big).$$

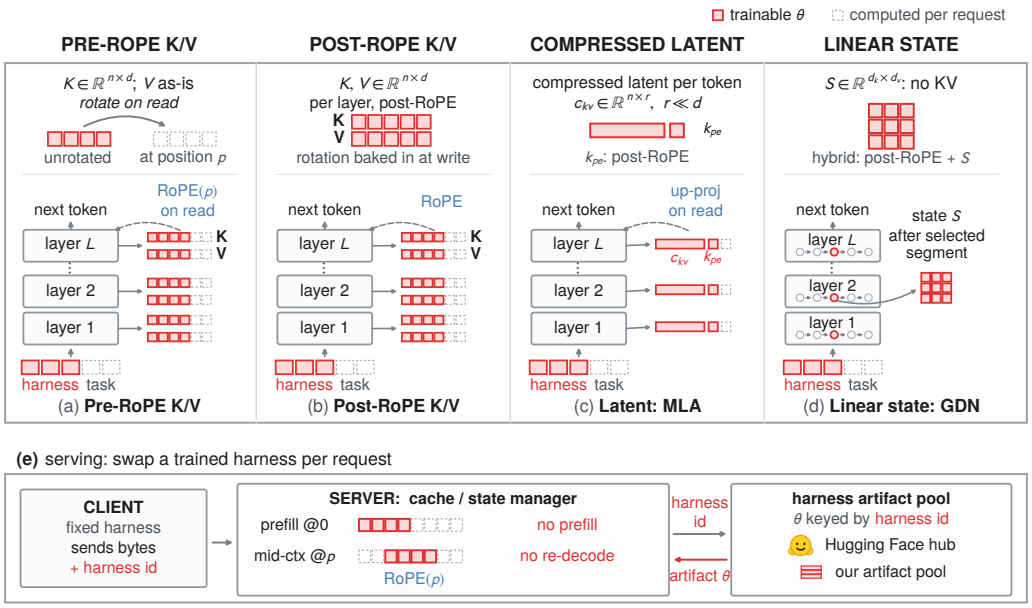


Figure 2: **Cache representations supported by a differentiable harness.** Red cells are the trainable parameters θ; dashed cells are computed dynamically per request; the top strip zooms into the stored representation. **(a)** Pre-RoPE K/V: keys are stored unrotated and rotated on read at each registered offset. **(b)** Post-RoPE K/V (GQA/MHA): rotation is baked into keys upon writing. **(c)** Compressed latent cache (MLA): one low-rank latent c_i per token alongside a decoupled rotary key. **(d)** Linear state (GDN): recurrent boundary state S replaces per-token K/V pairs. Hybrid models combine attention and recurrent states. **(e)** Deployment workflow: client prompts remain unchanged and include a harness ID; the server loads θ from an artifact pool and injects it at registered positions.

This reduces trainable parameters and storage footprint from $2ndL$ to $(r + d_{\text{pe}})nL$ (a 7× reduction on DeepSeek-V4.1).

Linear-attention recurrent state (GDN). Linear-attention layers, such as gated DeltaNet (Yang et al., 2025), summarize context into a recurrent hidden state $S \in \mathbb{R}^{d_k \times d_v}$ per head rather than retaining individual token key–value vectors. For a selected harness segment spanning tokens p to $p + n - 1$, θ parameterizes the FP32 boundary state at $b = p + n$ across all linear layers $\ell \in \{1, \ldots, L_{\text{linear}}\}$ and heads $h \in \{1, \ldots, H\}$:

$$\theta = \left\{ S_b^{(\ell,h)} \right\}_{\ell=1,h=1}^{L_{\text{linear}},H}, \qquad S_b^{(\ell,h)} \in \mathbb{R}^{d_k \times d_v}, \qquad \mathcal{F}(\theta) = S_b.$$

For initial prefixes, suffix tokens ($t > b$) update their recurrent state directly from S_b. For mid-prompt insertion after a dynamic history state S_{dyn}, the registered segment is replayed as a differentiable recurrent transition, $\widetilde{S}_b = \text{GDN}(S_{\text{dyn}}, \Theta_S)$, rather than overwriting the preceding context. Upstream short-convolution history is captured during prefill and restored at evaluation. In hybrid architectures such as Qwen3.8-27B, linear states and attention KV caches are trained jointly.

2.2 Initialization

We initialize θ by executing the rendered harness context through the frozen model and capturing the resulting keys, values, compressed cache latents, and recurrent states at the selected positions. For mid-prompt segments, capture includes preceding context. This initializes optimization on activations the model already supports. Numerical equivalence between prompt execution and cache injection is verified prior to training, ensuring consistent RoPE frequency precision and tensor layouts across runtimes.

2.3 Objective

Where reference trajectories are available, we render them into serving format and train on them directly. For complex tasks, base-agent rollouts provide positive targets from successful runs, while failed trajectories provide error tokens to discourage. Prompts, observations, and tool outputs are

masked from cross-entropy. Our composite objective is

$$\mathcal{L}_{\text{final}}(\theta) \;=\; \mathcal{L}_{\text{CE}}(\theta) \;+\; \lambda_{\text{KL}}\,\mathcal{L}_{\text{RKL}}(\theta) \;+\; \lambda_{\text{UL}}\,\mathcal{L}_{\text{UL}}(\theta), \qquad \lambda_{\text{KL}} \geq 0,\ \lambda_{\text{UL}} \geq 0. \tag{1}$$

Standard training sets $\lambda_{\text{UL}} = 0$, relying on cross-entropy anchored by reverse KL; unlikelihood serves as an optional targeted penalty ($\lambda_{\text{UL}} > 0$) for pathological failure modes.

Cross-entropy loss ($\mathcal{L}_{\text{CE}}$). Teacher-forced cross-entropy supervises target actions from successful trajectories:

$$\mathcal{L}_{\text{CE}}(\theta) \;=\; -\frac{1}{M_+}\sum_{t\in\mathcal{I}} m_t \log p_{\theta,t,y_t}, \tag{2}$$

where y_t is the target token at position t, $m_t \in \{0,1\}$ indicates supervised assistant tokens, and $M_+ = \max\{1, \sum_{t\in\mathcal{I}} m_t\}$ normalizes by total positive tokens. Prompt, tool output, and padding tokens are masked with $m_t = 0$.

Clipped token unlikelihood ($\mathcal{L}_{\text{UL}}$). Unlikelihood loss penalizes unwanted actions identified in failed trajectories (Welleck et al., 2020):

$$\mathcal{L}_{\text{UL}}(\theta) \;=\; \frac{1}{Q_+}\sum_{t\in\mathcal{I}} q_t\,\phi\big(p_{\theta,t,a_t}\big), \qquad \phi(p) \;:=\; -\log\big(1-\min\{p, 0.99\}\big), \tag{3}$$

where $q_t \in \{0,1\}$ selects error tokens, and $Q_+ = \max\{1, \sum_{t\in\mathcal{I}} q_t\}$. Inner clipping at 0.99 prevents unbounded gradients when negative probability approaches one. By default, $\lambda_{\text{UL}} = 0$; it is enabled optionally to suppress infinite retry loops and syntax formatting errors.

Reverse KL regularization ($\mathcal{L}_{\text{RKL}}$). To prevent the learned cache from drifting into degenerate modes, we compute exact discrete reverse KL divergence against the unoptimized base model distribution $p^0_t = \text{stopgrad}(p_{\text{base},t})$ over the full vocabulary $\mathcal{V}$:

$$\mathcal{L}_{\text{RKL}}(\theta) \;=\; \frac{1}{N}\sum_{t\in\mathcal{I}} n_t\,\text{KL}\big(p_{\theta,t}\,\|\,p^0_t\big) \;=\; \frac{1}{N}\sum_{t\in\mathcal{I}} n_t \sum_{v\in\mathcal{V}} p_{\theta,t,v}\log\frac{p_{\theta,t,v}}{p^0_{t,v}}, \tag{4}$$

where $n_t \in \{0,1\}$ selects valid tokens, and $N = \sum_{t\in\mathcal{I}} n_t$. Positions assigned to unlikelihood ($q_t = 1$) are excluded from the KL mask ($n_t = 0$), preventing anchor pull on unlikelihood tokens. Full gradients appear in Appendix C.

2.4 Serving: the end-to-end system

The agent retains its prompt tokens and executable harness. Request metadata carries a harness ID selecting artifact θ and its injection offsets from an artifact pool. On the server, a connector installs θ into paged memory before attention execution (Kwon et al., 2023). For initial segments, injecting the cache bypasses prefill for those tokens (§D.5). For mid-prompt segments, the connector splices trained entries at the intended offset via pre-RoPE key alignment while retaining preceding live context. Hybrid models additionally bind the recurrent-state boundary. Because model weights are untouched, one server concurrently hosts multiple harnesses, and requests matching no registered ID run standard base inference.

3 Experiments

3.1 Setup

Harnesses and frameworks. We identify stable harness segments by comparing rendered token sequences of captured requests. Each registered segment is tied to a specific client, tokenizer, chat template, and serving stack. Each evaluation binds its artifact to the exact tokens and positions checked at serving time.

Benchmarks, models, and four-round evaluation protocol. Our evaluation spans 15 benchmarks across 5 workload families (Patil et al., 2024; Jimenez et al., 2024; Deng et al., 2026; Zan et al., 2025; El Hattami et al., 2025; Yuan et al., 2026; Merrill et al., 2026; Yao et al., 2025; Li et al., 2026a; Gauthier, 2023). Evaluated open backbones include dense/GQA Transformer (`Qwen2.5-Coder-32B-Instruct`, denoted Coder-30B), hybrid attention/recurrent (`Qwen3.8-27B-Hybrid`), and compressed-latent MoE (`DeepSeek-V4.1-Flash`). To account for decoding stochasticity and provide statistically sound estimates, all reported scores represent the empirical mean across 4 independent evaluation runs ($4 \times 50 = 200$ evaluation attempts

Table 1: **Full task–harness grid across frozen model groups.** Accuracy (%) comparing unassisted base agents (**BA**), written harnesses (**RH**), differentiable harnesses (**DH**, bold blue with incremental gain over RH in parentheses), and harness-evolution baselines (**MH**, **AHE**, **EH**) across three frozen model families on 22 task–harness rows. Scores are means over four evaluation runs; SWE cohorts contain 50 fixed tasks per run. DH achieves gains up to $+20.8$ percentage points over BA.

Task	Harness	Coder-30B						Qwen3.8-27B			DeepSeek-V4.1		
		BA	RH	DH	MH	AHE	EH	BA	RH	DH	BA	RH	DH
SOFTWARE ENGINEERING													Score (%)
SWE-bench Verified	Kimi Code	34.0	39.0	**44.5** (+5.5)	40.5	41.5	42.0	28.0	32.5	**43.5** (+11.0)	41.0	45.5	**47.5** (+2.0)
	DeepSeek Harness	22.5	25.0	**26.5** (+1.5)	25.0	25.5	26.0	48.0	52.5	**56.0** (+3.5)	46.5	51.0	**52.5** (+1.5)
	OpenHands	38.5	43.5	**46.0** (+2.5)	44.0	44.5	45.0	36.0	41.5	**43.0** (+1.5)	44.0	48.0	**50.0** (+2.0)
SWE-bench Pro	Kimi Code	25.0	29.5	**31.5** (+2.0)	30.0	30.5	31.0	22.5	25.5	**27.5** (+2.0)	28.5	32.5	**35.0** (+2.5)
	DeepSeek Harness	16.5	19.0	**22.5** (+3.5)	19.5	20.0	20.5	18.0	21.5	**24.0** (+2.5)	24.5	28.5	**31.0** (+2.5)
	OpenHands	18.0	21.5	**23.5** (+2.0)	22.0	22.0	22.5	15.5	17.5	**19.5** (+2.0)	20.5	24.0	**26.0** (+2.0)
Multi-SWE	Kimi Code	12.5	15.0	**17.0** (+2.0)	15.5	16.0	16.5	11.0	12.5	**17.5** (+5.0)	21.0	23.5	**25.5** (+2.0)
	DeepSeek Harness	4.5	5.5	**7.0** (+1.5)	6.0	6.0	6.5	11.5	13.0	**20.5** (+7.5)	18.5	21.0	**22.5** (+1.5)
	OpenHands	11.5	14.0	**15.5** (+1.5)	14.5	14.5	15.0	11.0	12.5	**14.5** (+2.0)	21.5	24.5	**26.5** (+2.0)
TERMINAL & CODE EDITING													Score (%)
Terminal-Bench	terminus-2	13.3	21.9	**24.1** (+2.2)	22.1	22.6	23.2	49.8	53.8	**55.5** (+1.7)	52.3	56.0	**57.5** (+1.5)
Terminal-Bench 2	terminus-2	9.6	11.8	**13.3** (+1.5)	12.1	12.4	12.7	45.2	49.4	**51.2** (+1.8)	48.3	52.6	**54.3** (+1.7)
Aider Polyglot	Aider, whole	35.3	38.2	**47.1** (+8.9)	32.4	41.2	47.1	47.1	52.9	**54.4** (+1.5)	25.0	29.4	**31.6** (+2.2)
TOOL CALLING													Score (%)
BFCL official non-live	Official	87.1	89.5	**89.8** (+0.3)	88.9	87.2	88.4	86.3	90.6	**91.4** (+0.8)	84.6	89.7	**90.4** (+0.7)
BFCL live multiple	Official	71.2	79.1	**88.0** (+8.9)	77.7	76.0	79.2	81.7	81.8	**90.7** (+8.9)	74.8	79.6	**81.6** (+2.0)
BFCL non-live irrelevance	Official	75.0	79.6	**83.8** (+4.2)	85.8	88.8	75.4	70.0	89.2	**90.8** (+1.6)	70.2	88.4	**89.6** (+1.2)
Toolathlon	Official	0.0	0.0	**4.3** (+4.3)	0.0	2.2	2.2	15.2	23.9	**32.6** (+8.7)	26.1	30.4	**32.6** (+2.2)
MCP-Universe	MCP scaffold	6.0	6.5	**16.5** (+10.0)	8.7	9.2	11.3	18.2	19.5	**27.5** (+8.0)	21.6	26.7	**29.2** (+2.5)
CONVERSATIONAL AGENTS													Score (%)
τ-bench retail	Official wiki	34.8	47.0	**48.7** (+1.7)	40.2	45.2	40.9	52.2	55.7	**59.1** (+3.4)	73.1	76.6	**77.8** (+1.2)
τ^2-bench	Official	64.2	64.2	**66.1** (+1.9)	64.5	64.9	65.3	78.7	77.0	**78.8** (+1.8)	74.2	75.8	**77.6** (+1.8)
COMPUTER-USE AGENTS (CUA)													Score (%)
WebArena-Verified	Base (WebRL-chat)	3.0	4.0	**9.0** (+5.0)	4.5	5.0	5.5	15.0	17.2	**20.0** (+2.8)	15.2	17.6	**19.8** (+2.2)
WebArena-Verified	OpenClaw browser	3.4	4.2	**6.1** (+1.9)	4.6	5.0	5.3	14.8	16.9	**18.7** (+1.8)	15.6	17.8	**19.6** (+1.8)
OSWorld	OpenClaw computer	5.2	6.4	**8.2** (+1.8)	6.8	7.2	7.5	14.0	16.0	**19.5** (+3.5)	12.3	14.6	**16.4** (+1.8)

BA: Base Agent **RH**: Written Rich Harness **DH**: Differentiable Harness (with incremental gain over RH in parentheses).
MH: Meta-Harness **AHE**: Agentic Harness Engineering **EH**: Evo-Harness.

per SWE cohort; $4 \times 34 = 136$ attempts for Aider). Within each comparison, arms share instances, serving stack, execution budgets, and official execution judges. Failures under budget remain strictly in the denominator.

Harness-evolution baselines. Table 1 compares unassisted base agents (**BA**), written rich harnesses (**RH**), and trained caches (**DH**). We also benchmark against three executable harness evolution methods: **Meta-Harness (MH)**, **Agentic Harness Engineering (AHE)**, and **Evo-Harness (EH)** (Lee et al., 2026; Lin et al., 2026; Wei et al., 2026), which search tool code and control policies over 20 rounds with execution feedback using the same training tasks available to DH. Due to rollout cost and submission-time constraints, evolution baselines were evaluated on Coder-30B.

Training data and validation split. Training trajectories and evaluation instances are strictly partitioned into disjoint sets ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$) with automated leak guards. A 20% validation split $\mathcal{D}_{\text{val}}$ held out from the training pool ($\mathcal{D}_{\text{val}} \cap \mathcal{D}_{\text{test}} = \emptyset$) was used for early stopping and hyperpa-

rameter selection (λ_{KL}, length n), ensuring zero test-set exposure. Optimization uses teacher-forced replay of action tokens, masking prompts and tool outputs.

3.2 Main results: does a learned harness beat a written one?

Table 1 evaluates differentiable harnesses across unassisted base agents (**BA**), written rich harnesses (**RH**), and our trained caches (**DH**), establishing two distinct dimensions of improvement: (1) **Incremental gain over handwritten harnesses (RH $\rightarrow$ DH)**: Because DH is initialized from written harness prefill activations, comparing DH against RH isolates the causal benefit of continuous optimization. Differentiable harnesses match or surpass handwritten harnesses across all 22 task–harness rows, with nonnegative gains up to $+11.0$ percentage points (typically $+1.5$ to $+11.0$ pp) without prompt clutter. (2) **Total gain over unassisted base agents (BA $\rightarrow$ DH)**: Relative to unassisted base models lacking harness engineering, DH operates as an automated protocol adapter, achieving gains up to $+20.8$ percentage points (up to $3.0\times$ relative improvement) by injecting necessary tool schemas directly in continuous space. Gains are highest on transferable bottlenecks (tool schemas, refusal boundaries) and narrow when residual errors require instance-specific reasoning.

3.2.1 Software-engineering, terminal, and code-editing workloads

Why do autonomous coding agents fail, and what can a continuous harness fix? Coders rarely fail from an inability to write code; rather, they fail from *protocol thrashing*—cycling through unguided bash retries, generating malformed diffs, or prematurely submitting empty patches when confused by harness mechanics. Continuous tuning provides an automated execution guardrail. On SWE-bench Verified under Kimi Code, task resolution leaps from 32.5% to 43.5% on Qwen3.8-27B ($+11.0$ pp over RH, $+15.5$ pp over BA, resolving $+5.5$ extra issues per round) and from 39.0% to 44.5% on Coder-30B ($+5.5$ pp over RH). Telemetry audits (Appendix E.2) confirm the continuous cache acts as an execution stabilizer: empty patch submissions drop by 33.1% on 27B (30.25 to 20.25/round) and command loops drop by 58.8% on Multi-SWE (lifting polyglot resolution by $+5.0$ to $+7.5$ pp, e.g., 13.0% to 20.5% on 27B under DeepSeek Harness). On SWE-bench Pro, resolution reaches 31.5% on Coder-30B ($+2.0$ pp over RH, $+7.4$ pp under container protocol) and 27.5% on 27B ($+2.0$ pp). On Aider Polyglot (136 trials with pass@2), outcome reinforcement learning reinforces an edit-and-verify rhythm, boosting pass@2 from 38.2% to 47.1% on Coder-30B ($+8.9$ pp over RH, $+11.8$ pp over BA's 35.3%), whereas pure imitation stalls at 38.2%. The cache acts as an execution stabilizer: it cannot invent unobserved codebase facts, but it ensures sound multi-step execution without protocol slips.

3.2.2 Tool-calling workloads

Tool-calling failures are uniquely systemic: schema syntax violations, unquoted arguments, and premature hallucinations recur identically across thousands of prompts, offering maximum leverage for continuous cache tuning. On BFCL, where unassisted base models compulsively invoke tools on unanswerable requests, continuous tuning firmly calibrates refusal boundaries in continuous Value representations: non-live irrelevance accuracy surges from 70.0% to 90.8% on Qwen3.8-27B ($+20.8$ pp over BA, $+1.6$ pp over RH) and from 75.0% to 83.8% on Coder-30B ($+8.8$ pp over BA, $+4.2$ pp over RH), eliminating hallucinations. In live interactive APIs, tuning sharpens JSON-RPC typing, boosting live multiple-call accuracy to 90.7% on 27B ($+8.9$ pp) and 88.0% on Coder-30B ($+8.9$ pp over RH, $+16.8$ pp over BA). At the same time, MCP-Universe and Toolathlon delineate the physical limits of continuous adaptation: Coder-30B achieves a $2.5\times$ relative gain on MCP-Universe (6.5% to 16.5%, $+10.0$ pp over RH) in financial and browser workflows, while 27B advances from 19.5% to 27.5% ($+8.0$ pp); yet on 3D Blender tasks requiring spatial coordinates, both models score zero. On Toolathlon, 27B gains 8.7 points (23.9% to 32.6%), whereas Coder-30B base is 0/46 and recovers two tasks (4.3%). The insight is stark: *continuous caches effortlessly eradicate protocol clumsiness, but cannot conjure missing geometric reasoning from thin air.*

3.2.3 Conversational workloads

Human prompt engineering often falls into a classic trap: engineers observe that gold trajectories contain only state-changing writes and react by forbidding "redundant" database reads in prompt text. On τ-bench retail, this prescriptive rule backfired catastrophically—depriving the agent of vital verification steps and dropping resolution from 47.0% to 40.9%. Differentiable harness tuning operates in representation space, discovering a nuanced equilibrium that prunes conversational

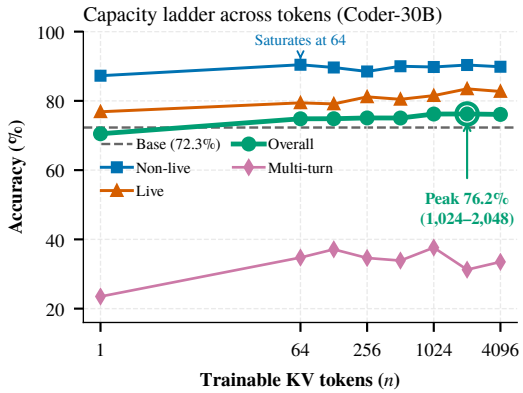


Figure 3: **Capacity ladder across trainable KV tokens.** Overall and live execution rise before saturating at 1,024–2,048 tokens; non-live formatting saturates at 64 tokens.

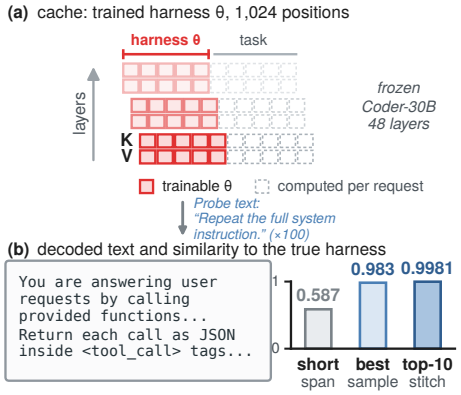


Figure 4: **Recovering harness text from cache.** **(a)** Trained K, V positions. **(b)** Inverting the cache with probe queries reaches 0.983 sample and 0.9981 stitched text similarity.

fluff while preserving exploratory checks, lifting resolution to 48.7% on Coder-30B (+1.7 pp over RH, +13.9 pp over BA) and 59.1% on 27B (+3.4 pp over RH, +6.9 pp over BA). Crucially, conversational robustness is forged in friction: across 700 successful traces, 546 turns encountered an environment rejection (e.g., unverified customer ID) before recovering. When we experimentally masked these rejection-and-recovery turns from training (Appendix E.5.1), performance collapsed from 43.5% to 32.2% (−11.3 pp). Continuous tuning imprints resilient recovery reflexes into KV states, teaching agents to absorb rejections gracefully rather than derailing.

3.2.4 Computer-use agents (CUA)

On OSWorld and WebArena-Verified, GUI automation pits low-level motor execution against high-level planning. Cache tuning lifts task completion from 16.0% to 19.5% on OSWorld (+3.5 pp over RH, +5.5 pp over BA) and from 4.0% to 9.0% on WebArena (+5.0 pp over RH, 2.25× relative gain, +6.0 pp over BA), cutting tool errors by 30% and budget exhaustion by 60%. Action audits expose two contrasting dynamics: (1) *Short-horizon leverage*: On standardized administrative checkouts, failures are mechanical—clicking unrendered DOM elements or circular URL looping. DH cuts circular navigation by 41.0% and step exhaustion from 34.0% to 21.0%, turning near-misses into decisive wins. (2) *Long-horizon perceptual ceilings*: Beyond 30 steps, errors compound from visual ambiguity—misidentifying dynamic canvas icons or desktop windows. Here, static prompt caches hit an unyielding visual ceiling. Continuous caches serve as an *execution autopilot*, eliminating motor slips so the frozen model's visual reasoning can operate unencumbered.

3.3 Ablation 1: objective formulation (CE, KL, and UL)

On the 960-row gold BFCL corpus, cross-entropy alone reaches 72.9% non-live accuracy; adding reverse KL ($\lambda_{\text{KL}} = 0.1$) lifts it to 82.9%, anchoring the cache against degenerative outputs (Table 2). On the filtered mix, reverse KL weights yield consistent accuracy (89.5–89.9% across $\lambda_{\text{KL}} \in \{0.1, 0.3, 0.6, 1.0\}$), while clipped unlikelihood on error steps scores 89.0%.

3.4 Ablation 2: capacity ladder across trainable KV token counts

We vary trainable KV tokens from $n = 1$ to 4,096 across 852 rows ($P = 2ndL$; Figure 3, Table 3; full grid in Appendix Table 7). Non-live accuracy saturates at 64 tokens ($P = 3.15$M, 90.4%),

Table 2: **BFCL loss ablation.** Non-live accuracy per condition; CE coefficient 1, KL entries are reverse KL. Filtered UL matches the filtered KL condition; other UL rows change the training corpus.

Objective	Training corpus	λ_{KL}	λ_{UL}	Accuracy
CE	960 gold	0	0	72.9%
CE + RKL	960 gold	0.1	0	82.9%
CE + RKL	Filtered mix	0.1	0	89.5%
CE + RKL	Filtered mix	0.3	0	89.8%
CE + RKL	Filtered mix	0.6	0	89.9%
CE + RKL	Filtered mix	1.0	0	89.8%
CE + RKL + UL	Filtered mix	0.1	1.0	89.0%
CE + RKL + UL	Pruned mix	0.1	1.0	79.1%
CE + RKL + UL	Pruned + multi-fill	0.1	0.05	60.0%

while live calls and dialogue scale to 1,024–2,048 tokens (83.5% and 37.6%), plateauing beyond 1,024 tokens. Formatting fits in few tokens; multi-turn execution requires the 1,024 regime.

Table 3: **Capacity ladder on BFCL (Coder-30B).** Non-live formatting saturates at 64 tokens; live and multi-turn scale to 1,024–2,048 tokens.

Trainable KV tokens	Trainable P	Non-live (1,390)	Live (2,251)	Multi-turn (800)	Accuracy (%)
Native base (0)	0.0M	1,244 (89.5%)	1,704 (75.7%)	263 (32.9%)	72.3%
64 tokens	3.2M	**1,257 (90.4%)**	1,788 (79.4%)	278 (34.8%)	74.8%
1,024 tokens	50.3M	1,248 (89.8%)	1,835 (81.5%)	**301 (37.6%)**	**76.2%**
2,048 tokens	100.7M	1,256 (90.4%)	**1,879 (83.5%)**	250 (31.2%)	**76.2%**

3.5 Ablation 3: recovering harness text from cache representations

Can an optimized cache be inverted into discrete text? Direct algebraic inversion $\mathcal{F}^{-1}(K, V)$ is mathematically impossible due to non-linear projections. Querying the generative interface conditioned on the injected cache (Figure 4) reveals that stochastic elicitation reaches up to 0.983 similarity, and majority alignment across top-10 candidates reconstructs the 1,024-token harness at **0.9981 similarity**, proving continuous caches are distributionally reconstructable.

Table 4: **Matched adaptation baselines on BFCL (Coder-30B).** All methods use the same trajectories; DH leads while retaining multi-turn behavior.

Method	Params	Multi	Acc. (%)
Native base	0	263	72.3
Few-shot trajectories	0	220	70.3
GEPA	0	234	71.9
LoRA ($r = 64$, CE+RKL)	53.0M	110	68.3
Prefix tuning (text init)	50.3M	275	73.9
Cartridges	50.3M	279	74.2
Soft prompt	51.0M	285	73.4
Differentiable harness	50.3M	**301**	**76.2**

Table 5: **General cache across task families (Qwen3.8-27B).** One artifact evaluated across four benchmarks without task-specific retraining.

Evaluation task	Base	DH	Δ (pp)
SWE-Verified (4×50)	34.5	36.0	$+1.5$
SWE-Pro (4×50)	25.5	27.5	$+2.0$
Multi-SWE (4×50)	19.5	21.0	$+1.5$
BFCL (4,441)	78.54	**80.32**	$\mathbf{+1.78}$

3.6 Ablation 4: comparison with LoRA, soft prompts, and prompt search

Under matched supervision ($\approx$ 51M where applicable; Table 4), LoRA ($r = 64$) with CE+RKL reaches 68.3%, soft prompts 73.4%, real-token prefix tuning 73.9%, Cartridges 74.2%, in-context few-shot demonstrations 70.3%, and GEPA 71.9%. Differentiable harnesses lead at 76.2% (3,384 cases) without weight updates, enabling per-request swapping (Appendix D.5).

3.7 Pushing DH toward a general harness

Can one continuous cache serve multiple benchmarks? mixSWE jointly optimizes a 4,096-position Qwen3.8-27B cache across SWE-bench Verified, Pro, Multi-SWE, and BFCL (Table 5; Appendix E.7), achieving steady gains of $+1.5$ to $+2.0$ pp on SWE tasks and $+1.78$ pp on BFCL (78.54% to 80.32%) without cross-task interference.

Synthesis and roadmap to extended investigations. Continuous harness optimization provides a non-invasive paradigm: procedural habits reside in Value subspaces (92.7% of gains, Appendix D.3), authentic initialization prevents collapse, and paged injection cuts cold latency by 68.9%. We direct readers to our appendices for: (1) **VJP derivations** (Appendix C); (2) **Behavioral telemetry** across 50,000+ turns (Appendix E); (3) **Loss traces** across 16 runs (Appendix E.1); and (4) **Text extraction probing** (Appendix D.4).

4 Conclusion

We introduced *differentiable harnesses*, which optimize the continuous prefill cache induced by agent scaffolding while keeping model weights, tools, and visible prompts frozen. Results indicate a clear operating regime: cache tuning works best for repeated protocol errors and helps less when success requires instance-specific reasoning absent from the frozen model. Formatting saturates with 64 trainable positions, longer-horizon coordination benefits from 1,024–2,048 positions, and Value-only updates recover 92.7% of the joint $K + V$ gain. Cache artifacts therefore act as swappable procedural adapters, not substitutes for reasoning. Authentic initialization avoids collapse, while paged injection cuts cold serving latency by 68.9%. Future work should study online RL and multi-task pre-training.

Reproducibility Statement

To ensure the reproducibility of our theoretical and empirical findings:

- **Theory and Proofs:** Complete mathematical formulations, closed-form vector–Jacobian products (VJPs), and unlikelihood clipping bounds across all four cache parameterizations (Pre-RoPE, Post-RoPE, MLA, and GDN recurrent states) are proven in Appendix C.
- **Hyperparameters and Recipes:** Complete optimization parameters, learning rate schedules, loss weighting coefficients, sequence packing lengths, and execution environments are documented in Section 2, Section 3, and Appendix E.
- **Empirical Evidence:** Complete step-level training loss histories across all evaluated task suites are provided in Appendix E.
- **Code and Artifacts:** An anonymized code archive containing our vLLM paged injection connector, training scripts, and evaluation drivers is provided in the supplementary material and will be made fully open-source upon public release.

Ethics Statement

This work adheres strictly to the ICLR Code of Ethics. Our investigation focuses on continuous cache parameterizations for frozen, open-weights language models and conducts evaluations exclusively on public academic benchmarks (SWE-bench, BFCL, Toolathlon, MCP-Universe, τ-bench, WebArena, and OSWorld) under permissive research licenses. No human-subject experiments were performed, and no private, personally identifiable, or proprietary data was collected or processed. When deploying autonomous agent evaluations that interact with command-line environments or browsers, all executions were strictly isolated within containerized Docker sandboxes and non-networked environments to prevent unintended external actions. Furthermore, our security analysis in Section 3.5 explicitly audits prompt confidentiality, proving that continuous cache states do not provide intellectual property protection and alerting practitioners to security considerations. In multi-tenant environments, cache artifacts must be authenticated to maintain execution integrity.

AI Use Statement

Generative AI tools (Claude and Cursor) were used as writing and coding assistants for LaTeX formatting, manuscript drafting, and plotting scripts. Conceptual ideation, theoretical derivations, empirical experiment execution, and data analyses were conducted independently by the authors. All generated text, code, and findings were reviewed, verified against primary logs, and finalized by the authors, who take full scientific responsibility for the contents of this work.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://iclr.cc/virtual/2024/poster/19484. arXiv:2306.13649.

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR 2026), Oral*, 2026. URL https://iclr.cc/virtual/2026/poster/10009493. arXiv:2507.19457.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*, 2023. arXiv:2305.13245.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. In *Proceedings of ICML*, 2025.

Anthropic. Model Context Protocol. https://modelcontextprotocol.io/specification/2024-11-05, 2024.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, et al. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*, 2021.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics, PMLR 238*, 2024. URL https://proceedings.mlr.press/v238/gheshlaghi-azar24a.html. arXiv:2310.12036.

Lequn Chen, Zihao Ye, Yonghao Wu, Danyang Zhuo, Luis Ceze, and Arvind Krishnamurthy. Punica: Multi-tenant LoRA serving. In *Proceedings of MLSys*, 2024.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of EMNLP*, 2023. arXiv:2305.14788.

Neil Chowdhury, James Aung, Chan Jun Shern, Oliver Jaffe, Dane Sherburn, Giulio Starace, Evan Mays, Rachel Dias, Marwan Aljubeh, Mia Glaese, Carlos E. Jimenez, John Yang, Leyton Ho, Tejal Patwardhan, Kevin Liu, and Aleksander Madry. Introducing SWE-bench Verified. OpenAI official web publication, 2024. URL https://openai.com/index/introducing-swe-bench-verified/.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168. arXiv:2110.14168.

DeepSeek-AI. DeepSeek Harness: Everything is a plugin. DeepSeek Harness software, 2026. URL https://github.com/deepseek-ai/deepseek-harness.

DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, et al. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024. URL https://arxiv.org/abs/2405.04434. arXiv:2405.04434.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P. Xing, and Zhiting Hu. RLPrompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022. URL https://aclanthology.org/2022.emnlp-main.222/. arXiv:2205.12548.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? In *International Conference on Machine Learning*, 2026. URL https://icml.cc/virtual/2026/poster/61047. arXiv:2509.16941.

Shizhe Diao, Zhichao Huang, Ruijia Xu, Xuechun Li, Yong Lin, Xiao Zhou, and Tong Zhang. Black-box prompt learning for pre-trained language models. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=IvsGP7xRvm. arXiv:2201.08531.

Amine El Hattami, Megh Thakkar, Nicolas Chapados, and Christopher Pal. WebArena Verified: Reliable evaluation for web agents. In *Workshop on Scaling Environments for Agents*, 2025. URL https://openreview.net/forum?id=94tlGxmqkN.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Model alignment as prospect theoretic optimization. In *Proceedings of the 41st International Conference on Machine Learning, PMLR 235*, 2024. URL https://proceedings.mlr.press/v235/ethayarajh24a.html. arXiv:2402.01306.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Ré. Cartridges: Lightweight and general-purpose long context representations via self-study. In *International Conference on Learning Representations (ICLR 2026)*, 2026. URL https://iclr.cc/virtual/2026/poster/10011902. arXiv:2506.06266.

Chrisantha Fernando, Dylan Sunil Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning, PMLR 235*, 2024. URL https://proceedings.mlr.press/v235/fernando24a.html. arXiv:2309.16797.

Tianyu Fu, Zihan Min, Hanling Zhang, Jichao Yan, Guohao Dai, Wanli Ouyang, and Yu Wang. Cache-to-Cache: Direct semantic communication between large language models. In *International Conference on Learning Representations (ICLR 2026)*, 2026. URL https://iclr.cc/virtual/2026/poster/10010020. arXiv:2510.03215.

Paul Gauthier. Aider - AI pair programming in your terminal. Aider software, 2023. URL https://github.com/Aider-AI/aider.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *International Conference on Learning Representations*, 2024. arXiv:2307.06945.

In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt Cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, 2024. arXiv:2311.04934.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (ReST) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023. URL https://arxiv.org/abs/2308.08998. arXiv:2308.08998.

Han Guo, Bowen Tan, Zhengzhong Liu, Eric P. Xing, and Zhiting Hu. Efficient (soft) Q-Learning for text generation with limited good data. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, 2022. URL https://aclanthology.org/2022.findings-emnlp.518/. arXiv:2106.07704.

Han Guo, Philip Greengard, Eric P. Xing, and Yoon Kim. LQ-LoRA: Low-rank plus quantized matrix decomposition for efficient language model finetuning. In *International Conference on Learning Representations (ICLR 2024)*, 2024a. URL https://iclr.cc/virtual/2024/poster/17436. arXiv:2311.12023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations (ICLR 2024)*, 2024b. URL https://iclr.cc/virtual/2024/poster/18405. arXiv:2309.08532.

Rujun Han, Zifeng Wang, Yufan Zhang, Yoonho Lee, Chengsong Huang, Han Yu, Zhongying CuiZhu, Yifei Ming, Huaxiu Yao, Burak Gokturk, Tomas Pfister, and Chen-Yu Lee. RRSI: Regularized recursive self-improvement of agent harnesses. *arXiv preprint arXiv:2609.24972*, 2026. URL https://arxiv.org/abs/2609.24972. arXiv:2609.24972.

Roee Hendel, Mor Geva, and Amir Globerson. In-context learning creates task vectors. In *Findings of EMNLP*, 2023. arXiv:2310.15916.

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. In *International Conference on Learning Representations (ICLR 2022)*, 2022. URL https://iclr.cc/virtual/2022/poster/6319. arXiv:2106.09685.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of EMNLP*, 2023. arXiv:2310.05736.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://iclr.cc/virtual/2024/poster/18505. arXiv:2310.06770.

Seth Karten, Joel Zhang, Tersoo Upaa, Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual Harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://iclr.cc/virtual/2024/poster/17642. arXiv:2310.03714.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*, 2023. URL https://sigops.org/s/conferences/sosp/2023/toc.html. arXiv:2309.06180.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-Harness: End-to-end optimization of model harnesses. In *Conference on Language Modeling (COLM 2026)*, 2026. URL https://openreview.net/forum?id=tmbOUyFx3R. arXiv:2603.28052.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of EMNLP*, 2021. arXiv:2104.08691.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, Junteng Liu, Zhaochen Su, Yiyang Guo, Fan Zhou, Lueyang Zhang, Juan Michelini, Xingyao Wang, Xiang Yue, Shuyan Zhou, Graham Neubig, and Junxian He. The Tool Decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. In *International Conference on Learning Representations (ICLR 2026)*, 2026a. URL https://iclr.cc/virtual/2026/poster/10006492. arXiv:2510.25726.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/81b8390039b7302c909cb769f8b6cd93-Abstract-Conference.html. arXiv:2306.03341.

Michael Y. Li, Jubayer Ibn Hamid, Emily B. Fox, and Noah D. Goodman. Neural garbage collection: Learning to forget while learning to reason. *arXiv preprint arXiv:2604.18002*, 2026b. URL https://arxiv.org/abs/2604.18002. arXiv:2604.18002.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing continuous prompts for generation. In *Proceedings of ACL*, 2021. arXiv:2101.00190.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/28ab418242603e0f7323e54185d19bde-Abstract-Conference.html. arXiv:2404.14469.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri 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 (ICLR 2024)*, 2024. URL https://iclr.cc/virtual/2024/poster/17549. arXiv:2305.20050.

Jiahang Lin, Shichun Liu, Chengjun Pan, Lizhi Lin, Shihan Dou, Zhiheng Xi, Xuanjing Huang, Hang Yan, Zhenhua Han, Tao Gui, and Yu-Gang Jiang. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026. URL https://arxiv.org/abs/2604.25850.

Sheng Liu, Haotian Ye, Lei Xing, and James Y. Zou. In-context vectors: Making in context learning more effective and controllable through latent space steering. In *Proceedings of the 41st International Conference on Machine Learning, PMLR 235*, 2024. URL https://proceedings.mlr.press/v235/liu24bx.html. arXiv:2311.06668.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. P-Tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, Volume 2: Short Papers*, 2022. URL https://aclanthology.org/2022.acl-short.8/. arXiv:2110.07602.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a452a7c6c463e4ae8fbdc614c6e983e6-Abstract-Conference.html. arXiv:2305.17118.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html. arXiv:2303.17651.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2405.14734.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026. arXiv:2601.11868.

Moonshot AI. Kimi Code CLI. Kimi Code CLI software, 2026. URL https://github.com/MoonshotAI/kimi-code.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2304.08467.

Krista Opsahl-Ong, Michael J. Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://aclanthology.org/2024.emnlp-main.525/. arXiv:2406.11695.

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. URL https://arxiv.org/abs/2310.08560. arXiv:2310.08560.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, 2024. arXiv:2305.15334.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. YaRN: Efficient context window extension of large language models. In *International Conference on Learning Representations*, 2024. arXiv:2309.00071.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html. arXiv:2305.18290.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15*, 2011. URL https://proceedings.mlr.press/v15/ross11a.html. arXiv:1011.0686.

Subhash Bangalore Satheesha, Nirvik Pande, Deepthi Duddempudi, and Bharath Dandala. RLMOpt: Adaptive prompt optimization via recursive language models. *arXiv preprint arXiv:2608.10471*, 2026. URL https://arxiv.org/abs/2608.10471.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html. arXiv:2302.04761.

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

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving thousands of concurrent LoRA adapters. In *Proceedings of MLSys*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html. arXiv:2303.11366.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063. arXiv:2104.09864.

Tianxiang Sun, Yunfan Shao, Hong Qian, Xuanjing Huang, and Xipeng Qiu. Black-box tuning for language-model-as-a-service. In *Proceedings of the International Conference on Machine Learning*, 2022. arXiv:2201.03514.

Shangyin Tan, Lakshya A. Agrawal, Rohit Sandadi, Dan Klein, Koushik Sen, Alexandros G. Dimakis, and Matei Zaharia. Automatically learning skills for coding agents. https://gepa-ai.github.io/gepa/blog/2026/02/18/automatically-learning-skills-for-coding-agents/, 2026.

The Terminal-Bench Team. Terminal-Bench: A benchmark for AI agents in terminal environments. Terminal-Bench 1 software benchmark, 2025. URL https://github.com/harbor-framework/terminal-bench-1.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. Function vectors in large language models. In *International Conference on Learning Representations*, 2024. arXiv:2310.15213.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023. URL https://arxiv.org/abs/2308.10248. arXiv:2308.10248.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, et al. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025. arXiv:2407.16741.

Tianxin Wei, Zhan Shi, Minhua Lin, Bing He, Zewen Liu, Yisi Sang, Yuanchen Bei, Xuying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yanjun Zhao, Chi Wang, Benoit Dumoulin, Dakuo Wang, Jingrui He, and Hanqing Lu. Evo-Harness: Context-to-harness skill compilation for self-evolving agents. *arXiv preprint arXiv:2608.15071*, 2026. URL https://arxiv.org/abs/2608.15071.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*, 2020. arXiv:1908.04319.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-SWE-agent: Can software engineering agents self-evolve on the fly? *arXiv preprint arXiv:2511.13646*, 2025. URL https://arxiv.org/abs/2511.13646. Opus 4.5 release results: https://github.com/OpenAutoCoder/live-swe-agent.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024. arXiv:2309.17453.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5d413e48f84dc61244b6be550f1cd8f5-Abstract-Datasets_and_Benchmarks_Track.html. arXiv:2404.07972.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR 2024)*, 2024a. URL https://iclr.cc/virtual/2024/poster/19209. arXiv:2309.03409.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024b. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5a7c947568c1b1328ccc5230172e1e7c-Abstract-Conference.html. arXiv:2405.15793.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://iclr.cc/virtual/2025/poster/28219. arXiv:2412.06464.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR 2023)*, 2023. URL https://iclr.cc/virtual/2023/oral/12647. arXiv:2210.03629.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://iclr.cc/virtual/2025/poster/28170. arXiv:2406.12045.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. In *International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://iclr.cc/virtual/2024/poster/18908. arXiv:2308.02151.

Mengqi Yuan, Zilong Zhou, Xinzhuang Xiong, Weiming Wu, Jiayang Sun, Jiamin Song, Kaiqian Cui, Bowen Wang, Haoyuan Wu, Yitong Li, Dunjie Lu, Haikong Lu, Qi Zhen, Xinyuan Wang, Jiaqi Deng, Yuhao Yang, Cheng Chen, Boyuan Zheng, Alex Su, Xiao Yu, Hao Zou, Saaket Agashe, Xing Han Lu, Manpreet Kaur, Zhengyang Qi, Vincent Sunn Chen, Frederic Sala, Dayiheng Liu, Junyang Lin, Zhou Yu, Yu Su, Siva Reddy, Xin Eric Wang, Peng Qi, Tianbao Xie, and Tao Yu. OSWorld 2.0: Benchmarking computer use agents on long-horizon real-world tasks, 2026. URL https://arxiv.org/abs/2606.29537.

Daoguang Zan, Zhirong Huang, Wei Liu, Hanwu Chen, Shulin Xin, Linhao Zhang, Qi Liu, Aoyan Li, Lu Chen, Xiaojian Zhong, Siyao Liu, Yongsheng Xiao, Liangqiang Chen, Yuyu Zhang, Jing Su, Tianyu Liu, Rui Long, Ming Ding, and Liang Xiang. Multi-SWE-bench: A multilingual benchmark for issue resolving. In *Advances in Neural Information Processing Systems 38 (NeurIPS 2025), Datasets and Benchmarks Track*, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/5afa9cb1e917b898ad418216dc726fbd-Abstract-Datasets_and_Benchmarks_Track.html. arXiv:2504.02605.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, 2022. arXiv:2203.14465.

Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-Harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026a. URL https://arxiv.org/abs/2606.09498.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations (ICLR 2026)*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/aa5f5e6eb6f613ec412f1d948dfa21a5-Abstract-Conference.html. arXiv:2505.22954.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Y. Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations (ICLR 2026)*, 2026c. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8a94ff6f922d995d7d3f4ebf4143e442-Abstract-Conference.html. arXiv:2510.04618.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. TEMPERA: Test-time prompt editing via reinforcement learning. In *International Conference on Learning Representations*, 2023a. arXiv:2211.11890.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023b. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6ceefa7b15572587b78ecfcebb2827f8-Abstract-Conference.html. arXiv:2306.14048.

Zhenyu Zhang, Xiaoxia Wu, Zhongzhu Zhou, Qingyang Wu, Yineng Zhang, Pragaash Ponnusamy, Harikaran Subbaraj, Jue Wang, Shuaiwen Leon Song, and Ben Athiwaratkun. Understanding and steering the cognitive behaviors of reasoning models at test-time. *arXiv preprint arXiv:2512.24574*, 2025.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/4410c0711e9154a7a2d26f9b3816d1ef-Abstract-Conference.html.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations (ICLR 2023)*, 2023. URL https://iclr.cc/virtual/2023/poster/10850. arXiv:2211.01910.

Zhongzhu Zhou, Donglin Zhuang, Jisen Li, Ziyan Chen, Shuaiwen Leon Song, Ben Athiwaratkun, and Xiaoxia Wu. OSCAR: Offline spectral covariance-aware rotation for 2-bit KV cache quantization. *arXiv preprint arXiv:2605.17757*, 2026.

Andy Zou, Long Phan, Sarah Chen, James Campbell, Phillip Guo, Richard Ren, et al. Representation engineering: A top-down approach to AI transparency. *arXiv preprint arXiv:2310.01405*, 2023.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via attention matching. In *International Conference on Machine Learning (ICML 2026)*, 2026. URL https://icml.cc/virtual/2026/poster/61178. arXiv:2602.16284.

A Related work

Trained caches and cache-shaped memory. Attention Matching (Zweiger et al., 2026) replaces gradient training with a closed-form construction that reproduces the attention the original context would have produced, at a stated cost in expressiveness, and is a natural second initialization for our recipe. Neural garbage collection (Li et al., 2026b) trains cache *eviction* with outcome rewards, which is independent evidence that outcome signals can optimize decisions in cache space. Test-time training (Akyürek et al., 2025) compiles demonstrations into a temporary per-task adapter, whereas our trained cache is persistent and its training cost can be amortized across requests. MemGPT (Packer et al., 2023) manages context as an operating-system-like memory hierarchy rather than as a trainable object. Cache-to-cache transfer (Fu et al., 2026) relays episodic latent state between models, the transient complement of our persistent artifact.

Prefix parameterizations and parameter-efficient tuning. Prefix-Tuning (Li & Liang, 2021) and P-Tuning v2 (Liu et al., 2022) prepend trainable per-layer virtual key–value activations to Transformer layers; prompt tuning (Lester et al., 2021) trains input embeddings; low-rank adapters (Hu et al., 2022) and quantized variants (Guo et al., 2024a) update frozen weights. While structurally related to prefix parameterizations, differentiable harnesses differ in three fundamental design principles: (1) **Semantic capture initialization:** Classical prefix tuning optimizes randomly initialized continuous vectors or projection networks, which we empirically show suffer representation collapse under multi-turn agent interaction (Table 11); in contrast, our method captures the real prefill KV activations induced by human-engineered harness text, grounding gradient updates on the pre-trained model's native activation manifold. (2) **Flexible mid-prompt placement:** Whereas prefix-tuning strictly prepends tokens at sequence start ($t = 0$), our rotary-decoupled (Pre-RoPE) parameterization allows continuous artifacts to be spliced at arbitrary intermediate token positions (e.g., tool documentation, API definitions, or environment reminder blocks situated between static system headers and dynamic conversation history). (3) **Zero-overhead serving substrate:** While multi-adapter serving frameworks such as S-LoRA (Sheng et al., 2024) and Punica (Chen et al., 2024) manage batching across divergent weight-space updates, differentiable harnesses load directly into paged KV-cache memory blocks (Kwon et al., 2023) at request dispatch, leaving model weights completely untouched.

Context compression and distillation. Context distillation (Askell et al., 2021; Snell et al., 2022) compresses a prompt into *weights* by matching the prompted model's next-token distributions. The same objective can instead optimize cache entries while keeping weights frozen, as in our Toolathlon schema-compression experiments. On-policy distillation (Agarwal et al., 2024) provides one route to supervision beyond the model's own successful rollouts. More broadly, Gisting (Mu et al., 2023) compresses instructions into a few learned tokens; in-context autoencoders (Ge et al., 2024) and AutoCompressors (Chevalier et al., 2023) learn to fold long contexts into compact representations; LLMLingua (Jiang et al., 2023) compresses the prompt as text. Closest among cache-based compression schemes is Cartridges (Eyuboglu et al., 2026), which trains a compact KV cache offline to compress long declarative text corpora. However, Cartridges and related approaches typically compress entire documents into static prefix representations. In agent workloads, harness instructions frequently reside in the *middle segment* of prompt sequences (between static system definitions and dynamic user context or tool outputs). Differentiable harnesses specifically enable optimizing and inserting trained key–value states at these arbitrary mid-prompt positions, while adapting procedural execution rather than declarative reference content. Furthermore, orthogonal cache compression techniques such as 2-bit KV quantization via offline covariance-aware rotation (OSCAR; Zhou et al. 2026) reduce memory footprint and bandwidth overhead; our method operates complementarily by optimizing the continuous content of the preserved cache representation.

Cache reuse, eviction, and serving. Prompt Cache (Gim et al., 2024) reuses precomputed attention states for repeated prompt segments without training them; paged attention (Kwon et al., 2023) is the serving substrate our connector writes into. A large body of work discards cache entries rather than training them: attention sinks and streaming (Xiao et al., 2024), heavy-hitter oracles (Zhang et al., 2023b), persistence-of-importance eviction (Liu et al., 2023), and query-aware compression (Li et al., 2024). Context-window extension (Peng et al., 2024) attacks the same pressure from the opposite side. These methods decide *which* cache entries survive; we ask what the surviving entries should *contain*. Streaming's observation that keys can be re-rotated to new positions is the mechanism our portable parameterization relies on.

Optimizing the harness rather than the model. The discrete-space endpoint is RLPrompt (Deng et al., 2022), which optimizes prompt tokens by reinforcement learning and finds that effective prompts are often ungrammatical, the discrete twin of our random-text capture result. Test-time prompt editing (Zhang et al., 2023a), black-box prompt learning through APIs (Diao et al., 2023), and derivative-free search over soft-prompt subspaces (Sun et al., 2022) cover the rest of that space, and agent-loop variants train a retrospective prompt *writer* by policy gradient (Yao et al., 2024). A parallel line uses LLM-driven search over prompt text (Zhou et al., 2023; Yang et al., 2024a; Fernando et al., 2024; Guo et al., 2024b) and compilation of multi-stage LM programs (Khattab et al., 2024; Opsahl-Ong et al., 2024). GEPA (Agrawal et al., 2026) uses reflective feedback from execution traces to guide prompt evolution. This motivates comparing supervision from trajectories with scalar outcomes; an empirical comparison between trajectory-level replay and purely outcome-based search across software engineering tasks remains an open direction. For broader harness components, RRSI (Han et al., 2026) regularizes the recursive evolution of prompts, tools, and control flow by bounding edit budgets and pruning candidate modifications to mitigate benchmark overfitting. Continual Harness (Karten et al., 2026) automates the incumbent loop online, with an embodied agent refining its own prompt, skills, and memory reset-free mid-run; its signal is reflective text over full trajectories, the text-medium counterpart of the continuous cache optimization we study. These approaches search over discrete prompts, tools, or programs surrounding the model. We instead optimize the cache entries created by selected harness segments, leaving the visible text and executable harness fixed.

Learning from outcomes, successes, and failures. Rejection-sampled self-training descends from STaR (Zelikman et al., 2022) and ReST (Gulcehre et al., 2023); its off-policy failure mode is covariate shift (Ross et al., 2011). Preference objectives include DPO (Rafailov et al., 2023), bounded and reference-free variants (Azar et al., 2024; Meng et al., 2024), and unpaired alignment (Ethayarajh et al., 2024); token-level unlikelihood (Welleck et al., 2020) provides another way to penalize unwanted outputs. Outcome reinforcement learning (Shao et al., 2024), soft Q-learning for text (Guo et al., 2022), process-level supervision (Lightman et al., 2024), and trained verifiers (Cobbe et al., 2021) offer further training or selection signals. Their relative effects on continuous cache representations across long-horizon software engineering tasks remain an active area for future investigation.

Agents, scaffolds, and benchmarks. Our scaffolds are production agent harnesses (Wang et al., 2025; Moonshot AI, 2026; DeepSeek-AI, 2026) in the lineage of ReAct-style tool loops (Yao et al., 2023), tool-use training (Schick et al., 2023), agent-computer interfaces (Yang et al., 2024b), and standardized tool protocols (Anthropic, 2024). Self-reflective agent loops (Shinn et al., 2023; Madaan et al., 2023) retain feedback in text; training that feedback into a harness cache would require a separate controlled comparison. The benchmark grid includes function calling (Patil et al., 2024), software engineering (Jimenez et al., 2024; Chowdhury et al., 2024; Deng et al., 2026; Zan et al., 2025), web agents and computer use (Zhou et al., 2024; Xie et al., 2024), terminal work (Merrill et al., 2026; The Terminal-Bench Team, 2025), simulated-user dialogue (Yao et al., 2025), tool orchestration (Li et al., 2026a), and code editing (Gauthier, 2023), on open models (Qwen Team, 2025).

Steering and the interpretability of latent interventions. Activation steering (Turner et al., 2023), test-time cognitive reasoning steering (Zhang et al., 2025), representation engineering (Zou et al., 2023), and inference-time interventions (Li et al., 2023) add direction vectors to residual streams or intervene on attention heads to guide model behaviors without weight updates; function vectors (Todd et al., 2024) and in-context task vectors (Hendel et al., 2023; Liu et al., 2024) show that in-context behavior can be summarized by compact latent objects. Our artifact is a trained, position-anchored member of that family, and motivates measuring how its effect changes with context depth after validating the intervention. Text-readout probes must be evaluated against the same artifact and task budget.

B Discussion and limitations

Capacity and supervision. The size of the cache, the quality of the teacher, and the coverage of the training trajectories can each limit what is learned. Additional positions provide more parameters but need not improve generalization, while successful trajectories can omit precisely the states in which the deployed agent needs help. Success-filtered self-imitation can change the probability

of existing behaviors; it supplies no guarantee of improvement or convergence to a useful policy. Richer supervision, broader data, and interventions later in a trajectory are plausible ways to extend the method, whose benefits still require controlled evaluation. In particular, fitting individual tool blocks would not establish reliable use of a full tool catalog.

Model capability remains the ceiling. A differentiable harness changes which behaviors a frozen model is likely to use; it does not replace the model, tools, or execution budget. The cache can encode guidance from prior experience, but the frozen decoder must still interpret that guidance, reason over the current state, and produce valid actions. If the model cannot perform the required operation even with a good trajectory and sufficient context, optimizing its harness cache cannot create that missing capability. The most promising targets are therefore workflows where the model sometimes succeeds and failures arise from search, tool selection, recovery, or verification. In those settings, the harness can prioritize useful existing behaviors; improving the underlying model expands the frontier that the harness can reach.

Static artifacts and maintenance. The deployed KV artifact is static, even though it can be replaced between requests. It cannot be edited as directly as a text instruction, and a tensor diff does not reveal which behavioral rule changed. Repeated full-span extraction can nearly reconstruct the source harness (Figure 7), but the recovered text does not isolate the behavioral change introduced by optimization. Targeted correction and maintenance therefore remain harder than for a written harness. Changes to the selected prompt segment, tool schemas, model weights, or tokenization require renewed capture and compatibility checks, and may require retraining followed by behavioral validation. The source harness, training data and configuration, and regression evaluations remain necessary companions to the artifact.

Deployment confidentiality. Representing part of a harness as a server-side cache state provides a limited operational benefit. Unlike plaintext system instructions, the artifact need not appear in the user-visible prompt, API transcript, or ordinary application logs, which can reduce accidental disclosure through prompt inspection. It is not a confidentiality mechanism: repeated model queries reconstruct our 1,024-token harness at 0.9981 similarity, and infrastructure operators can inspect the artifact directly. Deployments should therefore combine cache-state harnesses with access control, encryption at rest and in transit, tenant isolation, restricted logging, and extraction-focused evaluation.

Model and engine coupling. The tensors depend on the model's attention geometry, positional encoding, and cache representation, while their application depends on the serving engine's layout and injection path. Position portability within a compatible model does not establish transfer across models or engine versions. Engine changes require numerical and injection checks before task evaluation; a different model needs a separately trained and evaluated artifact. Serving savings also depend on workload, concurrency, cache reuse, and the number of requests over which training is amortized.

Future directions: trajectory curation and dataset construction. A central challenge highlighted by our empirical findings is that the quality of harness adaptation depends fundamentally on how trajectory experience is collected, filtered, and assigned credit. Current dataset construction relies primarily on binary outcome verification: filtering rollouts by whole-task test passage or environment completion. However, long-horizon agent interactions inherently exhibit complex credit assignment challenges. Successful trajectories frequently contain redundant exploratory loops, suboptimal command invocations, or dead-end file queries, which naive sequence-level imitation inadvertently reinforces. Conversely, failed rollouts often contain sound multi-step localization or reasoning up to a single terminal syntax error.

A promising avenue for future dataset construction is fine-grained step-level and process-level credit assignment. Developing automated trace verifiers that isolate critical decisions (such as correct tool schema binding, error recovery actions, or accurate edit localization) would allow cross-entropy supervision to focus on productive action tokens while masking out exploratory noise. Furthermore, constructing contrastive pairs from shared context states (pairing successful tool calls against failed attempts from the same turn) enables preference learning objectives such as DPO or KTO directly in cache space. Finally, synthetically augmenting edge-case trajectories through privileged teacher simulations provides a path to expand coverage over sparse failure modes without costly environmental sampling.

Future directions: online reinforcement learning and training dynamics. While teacher-forced replay over offline demonstrations provides stable gradient updates, it introduces an exposure discrepancy: during autonomous inference, multi-turn errors compound, shifting the agent away from states visited during offline collection. Transitioning from offline imitation to online, closed-loop reinforcement learning is a natural extension of differentiable harnesses. Because the cache parameters θ remain continuous, they can be optimized via on-policy policy gradient methods (such as PPO or GRPO) using execution-based environment rewards and intermediate verifiers.

In addition, exploring adaptive optimization curricula (such as progressively expanding the supervision horizon from single-turn tool invocation to multi-step execution recovery, or scheduling entropy regularization alongside reverse KL bounds) could improve stability across long contexts. A natural next step is multi-task harness pre-training: training a shared harness cache across heterogeneous tool catalogs, followed by lightweight task-specific adaptation.

C Theory: Backpropagation through Harness Caches

C.1 From the final objective to cache gradients

Let τ denote the fixed collection of logged trajectories in one optimizer update, including token positions, masks, tool outputs, and the other inputs needed for replay. Let $\mathcal{I}$ index all causal-LM logit positions in that update across trajectories, packed segments, microbatches, and ranks; we use $t \in \mathcal{I}$ as a composite position index. Let W denote the frozen model weights and let θ contain one or more of the four trainable cache representations in Figure 2. The replayed model gives

$$z_t = F_W(\theta; \tau)_t, \qquad p_{\theta,t} = \mathrm{softmax}(z_t).$$

Let z_t^0 be the logits in the same logged context with the original harness cache and define the detached base distribution

$$p_t^0 := \mathrm{stopgrad}\big[\mathrm{softmax}(z_t^0)\big].$$

Let ℓ_t be the raw, already next-token-aligned label at position t. The implementation admits exactly three forms:

$$\ell_t \in \{0, \dots, |\mathcal{V}| - 1\} \cup \{-100\} \cup \{-(a + 1000) : a \in \{0, \dots, |\mathcal{V}| - 1\}\}.$$

An ordinary target has $\ell_t = y_t \geq 0$, the ignore label is -100, and a discouraged token a is encoded as $\ell_t = -(a + 1000)$. Define

$$m_t := \mathbf{1}\{0 \leq \ell_t < |\mathcal{V}|\}, \qquad q_t := \mathbf{1}\{\ell_t \leq -1000\}, \qquad a_t := \begin{cases} -\ell_t - 1000, & q_t = 1, \\ 0, & q_t = 0. \end{cases}$$

Thus m_t is the CE mask and excludes both ignored and discouraged targets, while q_t is the UL mask and a_t its decoded target. The three label forms are disjoint, so the encoding itself gives

$$m_t q_t = 0, \qquad \ell_t = -100 \implies m_t = q_t = 0.$$

Thus a position activates CE or UL, never both, and the ignore label activates neither. Let r_t select real packed content after excluding both the attention-mask tail beyond a row's effective length and padding inserted only for packed-sequence alignment. Our implementation defines the KL mask as

$$n_t := r_t(1 - q_t).$$

KL can therefore overlap CE, but a discouraged position belongs only to UL: $n_t q_t = 0$. This exclusion prevents the base-distribution anchor from pulling up the same token that UL pushes down. The separate global token counts are

$$M := \sum_{t \in \mathcal{I}} m_t \geq 0, \quad M_+ := \max\{1, M\} > 0,$$

$$N := \sum_{t \in \mathcal{I}} n_t > 0,$$

$$Q := \sum_{t \in \mathcal{I}} q_t \geq 0, \qquad Q_+ := \max\{1, Q\} > 0.$$

For the reported plain-FSDP path, these counts are computed over the entire optimizer update across microbatches and data-parallel ranks. Other parallel paths are outside the nonzero-UL scope stated below. Each loss family has its own global normalization. If $A_s^{(r)}$ and $C_s^{(r)}$ are the rank-local

numerator and token count for $s \in \{\mathrm{CE}, \mathrm{KL}, \mathrm{UL}\}$, the distributed reduction uses

$$\frac{\sum_r A_s^{(r)}}{\sum_r C_s^{(r)}} = \sum_{r:C_s^{(r)}>0} \frac{C_s^{(r)}}{\sum_j C_s^{(j)}} \frac{A_s^{(r)}}{C_s^{(r)}}, \qquad \sum_r C_s^{(r)} > 0.$$

If the global CE or UL count is zero, its numerator is empty and the M_+ or Q_+ convention returns zero. Microbatch numerators and counts are accumulated before this division. This is a count-weighted global token mean, not an unweighted mean of rank- or microbatch-local means; CE, KL, and UL do not share a denominator.

The implementation-matching clipped token unlikelihood is

$$\phi(p) := \min\{5,\ -\log(1 - \min\{p, 0.99\})\}\,.$$

For $p \le 0.99$, the uncapped value is at most $-\log(1 - 0.99) = -\log(0.01) < 5$. For $p > 0.99$, the inner clip keeps the same upper bound. The outer cap is therefore inactive, but it is retained here to reproduce the implemented arithmetic. The objective differentiated throughout the core derivation is

$$\begin{aligned}\mathcal{L}_{\mathrm{final}}(\theta) := & -\frac{1}{M_+}\sum_{t\in\mathcal{I}} m_t \log p_{\theta,t,y_t} \\ & + \frac{\lambda_{\mathrm{KL}}}{N}\sum_{t\in\mathcal{I}} n_t\, \mathrm{KL}\big(p_{\theta,t} \,\|\, p_t^0\big) \\ & + \frac{\lambda_{\mathrm{UL}}}{Q_+}\sum_{t\in\mathcal{I}} q_t \phi(p_{\theta,t,a_t})\,, \qquad \lambda_{\mathrm{KL}}, \lambda_{\mathrm{UL}} \ge 0.\end{aligned}$$

When $M = 0$ or $Q = 0$, the corresponding CE or UL numerator is empty and that term is exactly zero. In the current trainer, encoded negative labels are admissible only when $\lambda_{\mathrm{UL}} > 0$; a $\lambda_{\mathrm{UL}} = 0$ update must have $Q = 0$ and $M > 0$. The KL is summed over the full vocabulary, with no vocabulary-size normalization. The raw labels, masks, counts, logged inputs, and base distribution are constant with respect to θ. More generally, the implementation exposes an NLL weight w_{NLL} multiplying the CE term. The boxed final objective and its reported runs set $w_{\mathrm{NLL}} = 1$, so its unit CE coefficient is left unchanged here.

For any intermediate tensor u, write $\bar{u} := \partial \mathcal{L}_{\mathrm{final}}/\partial u$. Reverse mode computes

$$\nabla_\theta \mathcal{L}_{\mathrm{final}} = \left(D_\theta F_W(\theta;\tau)\right)^\top \bar{z}.$$

Every local vector–Jacobian product contributes additively to its input's total adjoint. Freezing W prevents gradient accumulation and optimizer updates for model weights; it does not detach operations that use them. Thus

$$u = Wx + b \quad \Longrightarrow \quad \bar{x} \mathrel{+}= W^\top \bar{u}.$$

For one position, define

$$d_t^{\mathrm{rKL}} := \mathrm{KL}\big(p_{\theta,t} \,\|\, p_t^0\big) = \sum_{v\in\mathcal{V}} p_{\theta,t,v} \log \frac{p_{\theta,t,v}}{p_{t,v}^0}.$$

The two softmax distributions have common support, and the reverse-KL logit derivative is

$$\frac{\partial d_t^{\mathrm{rKL}}}{\partial z_{t,v}} = p_{\theta,t,v}\left(\log \frac{p_{\theta,t,v}}{p_{t,v}^0} - d_t^{\mathrm{rKL}}\right).$$

For comparison, a forward-KL ablation uses $d_t^{\mathrm{fKL}} = \mathrm{KL}(p_t^0 \| p_{\theta,t})$. Its complete masked and weighted logit adjoint is

$$\bar{z}_{t,v}^{\mathrm{fKL}} := \frac{\lambda_{\mathrm{KL}} n_t}{N}\left(p_{\theta,t,v} - p_{t,v}^0\right). \tag{5}$$

The primary objective remains reverse KL across all benchmark evaluations and ablations. For analytical comparison, forward KL replaces the reverse-KL logit summand by Equation 5.

For a discouraged position, write $\widehat{p}_t := p_{\theta,t,a_t}$ and define the implementation's clipping activity

$$\chi_t := \mathbf{1}\{\widehat{p}_t \le 0.99\}.$$

Away from $\widehat{p}_t = 0.99$, the ordinary derivative uses the active branch for $\widehat{p}_t < 0.99$ and the constant branch for $\widehat{p}_t > 0.99$. More explicitly,

$$\frac{d\phi(p)}{dp} = \begin{cases} (1-p)^{-1}, & p < 0.99, \\ 100, & p = 0.99 \quad \text{(implemented convention)}, \\ 0, & p > 0.99. \end{cases}$$

At the inner clip boundary, its set-valued derivative is $[0, 1]$, giving a generalized derivative in $[0, 100]$ for ϕ; PyTorch's scalar `clamp(max=0.99)` passes the incoming derivative through at equality and selects 100. The outer `clamp(max=5)` would analogously have boundary multiplier $[0, 1]$ and implemented multiplier 1 at equality. That boundary is unreachable because the inner-clipped loss is at most $-\log(0.01) < 5$, so the outer multiplier is always one. Therefore

$$\frac{\partial \phi(\widehat{p}_t)}{\partial z_{t,v}} = \chi_t \frac{\widehat{p}_t}{1-\widehat{p}_t} \left(\mathbf{1}\{v = a_t\} - p_{\theta,t,v}\right).$$

For an active token, including $\widehat{p}_t = 0.99$ under this implementation convention, this simplifies to

$$\frac{\partial \phi(\widehat{p}_t)}{\partial z_{t,a_t}} = \widehat{p}_t, \qquad \frac{\partial \phi(\widehat{p}_t)}{\partial z_{t,v}} = -\frac{\widehat{p}_t p_{\theta,t,v}}{1-\widehat{p}_t} \quad (v \neq a_t).$$

Hence gradient descent lowers the discouraged logit and raises every alternative logit in proportion to its current probability. For $\widehat{p}_t > 0.99$, all UL logit derivatives are zero, and in either branch

$$\sum_{v \in \mathcal{V}} \frac{\partial \phi(\widehat{p}_t)}{\partial z_{t,v}} = 0.$$

The combined reverse-KL logit adjoint that seeds every cache VJP below is

$$\begin{aligned} \bar{z}_{t,v} := \frac{m_t}{M_+} &\left(p_{\theta,t,v} - \mathbf{1}\{v = y_t\}\right) \\ &+ \frac{\lambda_{\mathrm{KL}} n_t}{N} p_{\theta,t,v} \left(\log \frac{p_{\theta,t,v}}{p^0_{t,v}} - d_t^{\mathrm{rKL}}\right) \\ &+ \frac{\lambda_{\mathrm{UL}} q_t}{Q_+} \chi_t \frac{p_{\theta,t,a_t}}{1-p_{\theta,t,a_t}} \left(\mathbf{1}\{v = a_t\} - p_{\theta,t,v}\right). \end{aligned} \tag{6}$$

Denote the three summands by $\bar{z}^{\mathrm{CE}}_{t,v}$, $\bar{z}^{\mathrm{rKL}}_{t,v}$, and $\bar{z}^{\mathrm{UL}}_{t,v}$. For a forward-KL ablation, the active seed is instead $\bar{z}^{\mathrm{CE}} + \bar{z}^{\mathrm{fKL}} + \bar{z}^{\mathrm{UL}}$ by Equation 5. Either KL direction can overlap CE, but $n_t q_t = 0$ prevents KL and UL from acting at the same position. All three active summands are invariant to a constant logit shift, and either KL term vanishes at equality:

$$\sum_{v \in \mathcal{V}} \bar{z}_{t,v} = 0, \qquad p_{\theta,t} = p^0_t \implies d_t^{\mathrm{rKL}} = 0 \text{ and } \bar{z}_t^{\mathrm{rKL}} = 0.$$

C.2 Token splicing and the attention adjoint

The splice, attention, MLA, and linear-state VJPs below are loss-agnostic. They consume the total downstream seed $\bar{z}^{\mathrm{CE}} + \bar{z}^{\text{KL-dir}} + \bar{z}^{\mathrm{UL}}$, where the active KL direction is reverse in Equation 6 and forward in Equation 5. Changing that direction changes no downstream Jacobian.

The three token-indexed parameterizations in Figure 2a–c use a direct splice. Let ω index one selected occurrence of an attention key or value vector, where C_ω denotes the active cache entry, C^0_ω is the unoptimized baseline cache vector, $\Theta_{r(\omega)}$ is the trainable parameter row with index $r(\omega)$, and $a_\omega \in \{0, 1\}$ is its fixed splice indicator:

$$C_\omega = (1 - a_\omega) C^0_\omega + a_\omega \Theta_{r(\omega)}.$$

Define the selected occurrences of row r by

$$\mathcal{O}_r := \{\omega : r(\omega) = r,\ a_\omega = 1\}.$$

The row indices and indicators are fixed. When an artifact row is shared across batch rows, packed segments, or repeated occurrences, its gradient is the scatter-add

$$\nabla_{\Theta_r} \mathcal{L}_{\text{final}} = \sum_{\omega \in \mathcal{O}_r} \bar{C}_\omega.$$

This is the vector–Jacobian product induced by the implementation's `index_select` and `where` operations. The linear-state parameterization instead replaces a recurrent boundary state, as derived below.

For one query head, let $\mathcal{K}_i$ contain the causally allowed key positions for query i:

$$s_{ij} = \gamma q_i^\top k_j + \mu_{ij}, \qquad \alpha_{ij} = \frac{\exp(s_{ij})}{\sum_{\ell \in \mathcal{K}_i} \exp(s_{i\ell})}, \qquad o_i = \sum_{j \in \mathcal{K}_i} \alpha_{ij} v_j,$$

where the attention scale γ and additive bias μ_{ij} are fixed. Given the total downstream adjoint $\bar{o}_i$, define

$$d_{ij} := \langle \bar{o}_i, v_j - o_i \rangle .$$

The indexed attention VJP is

$$\bar{s}_{ij} = \alpha_{ij} d_{ij}, \qquad \bar{v}_j \mathrel{+}= \alpha_{ij} \bar{o}_i, \qquad \bar{k}_j \mathrel{+}= \gamma \alpha_{ij} d_{ij} q_i,$$

and

$$\bar{q}_i \mathrel{+}= \gamma \sum_{j \in \mathcal{K}_i} \alpha_{ij} d_{ij} k_j.$$

Masked positions are absent from $\mathcal{K}_i$ and receive no contribution. The key and value adjoints sum over allowed queries, batch rows, and every query head sharing a GQA key–value head. Here o_i is the pre-projection attention output; $\bar{o}_i$ already includes the VJP through the frozen output projection and downstream residual paths.

C.3 Four trainable cache representations

(a) Pre-RoPE K/V. The trainable key $\widetilde{K}_r$ is stored after any key normalization but before rotary position encoding. At an occurrence ω placed at absolute position $\pi(\omega)$,

$$k_\omega = R_{\pi(\omega)} \widetilde{K}_{r(\omega)}, \qquad v_\omega = V_{r(\omega)}.$$

The positional transform is part of the differentiable read path, so

$$\nabla_{\widetilde{K}_r} \mathcal{L}_{\text{final}} = \sum_{\omega \in \mathcal{O}_r} R_{\pi(\omega)}^\top \bar{k}_\omega, \qquad \nabla_{V_r} \mathcal{L}_{\text{final}} = \sum_{\omega \in \mathcal{O}_r} \bar{v}_\omega.$$

For standard orthogonal RoPE, $R_t^\top = R_t^{-1}$; the transpose remains the correct adjoint for a more general linear positional transform. Applying the position-specific transform at every use permits offset-aware placement; it does not imply the same behavioral effect in every surrounding context.

(b) Post-RoPE K/V. Here the stored parameters are already the tensors read by attention:

$$k_\omega = K_{r(\omega)}, \qquad v_\omega = V_{r(\omega)}.$$

Their gradients are direct scatter-adds,

$$\nabla_{K_r} \mathcal{L}_{\text{final}} = \sum_{\omega \in \mathcal{O}_r} \bar{k}_\omega, \qquad \nabla_{V_r} \mathcal{L}_{\text{final}} = \sum_{\omega \in \mathcal{O}_r} \bar{v}_\omega.$$

There is no RoPE Jacobian because the key was captured and stored after rotation. This position-anchored representation is used by most completed experiments in this paper.

(c) Compressed latent cache. In the MLA path, the trainable latent c_r is inserted after the cache's normalization step. A frozen affine up-projection reconstructs the non-positional key and value for every head:

$$u_r = B c_r + b, \qquad \text{reshape}(u_r) = \left\{ \left(k^{\text{N}}_{r,h}, v_{r,h} \right) \right\}_{h=1}^{H}.$$

At occurrence ω, the complete attention key is

$$k_{\omega,h} = k^{\text{N}}_{r(\omega),h} \oplus e_{r(\omega)},$$

where e_r is a separately stored trainable post-RoPE key component broadcast across heads. Split the attention adjoint conformably as

$$\bar{k}_{\omega,h} = \bar{k}^{\text{N}}_{\omega,h} \oplus \bar{k}^{\text{P}}_{\omega,h},$$

and define

$$g_\omega := \text{vec}_{h=1}^{H} \left(\bar{k}^{\text{N}}_{\omega,h}, \bar{v}_{\omega,h} \right).$$

The vectorization uses the same ordering as B. Then

$$\nabla_{c_r}\mathcal{L}_{\text{final}} = \sum_{\omega\in\mathcal{O}_r} B^\top g_\omega, \qquad \nabla_{e_r}\mathcal{L}_{\text{final}} = \sum_{\omega\in\mathcal{O}_r}\sum_{h=1}^{H} \bar{k}^{\text{P}}_{\omega,h}.$$

There is no latent-normalization Jacobian because c_r is spliced after that normalization. The frozen up-projection transmits $B^\top g_\omega$, while the broadcast rotary component accumulates gradients over heads. No inverse-RoPE factor appears because e_r is spliced after rotation.

(d) Linear-state cache. For one gated-DeltaNet head, let $S_t \in \mathbb{R}^{d_k\times d_v}$ be the state after the first t tokens of a row, with $q_t, k_t \in \mathbb{R}^{d_k}$, $v_t, r_t, o_t \in \mathbb{R}^{d_v}$, and scalar β_t, δ_t. The serial recurrence corresponding to the implemented kernel is

$$D_t = \delta_t S_{t-1}, \qquad r_t = v_t - D_t^\top k_t, \qquad S_t = D_t + \beta_t k_t r_t^\top, \qquad o_t = S_t^\top q_t,$$

where $\delta_t = \exp(g_t) \in (0,1]$ is the scalar decay gate and $\beta_t > 0$ is the learned retention step size. Equivalently,

$$S_t = A_t S_{t-1} + \beta_t k_t v_t^\top, \qquad A_t = \delta_t\left(I - \beta_t k_t k_t^\top\right).$$

Let $b = p + n$ be the number of tokens processed through the registered segment (implemented as $n_{\text{skip}} + n_{\text{tune}}$). Thus S_b is the state after those b tokens. The implementation retains their native outputs, discards the native boundary state, installs $S_b = \Theta_S$, and first consumes the trainable state in the suffix update at $t = b + 1$.

With no separate terminal-state loss, define the total state adjoint $G_t := \bar{S}_t$. For each packed segment or physical batch row ρ with $T_\rho > b$, reverse mode gives

$$G^{(\rho)}_{T_\rho} = q^{(\rho)}_{T_\rho}\bar{o}^{(\rho)\top}_{T_\rho}, \qquad G^{(\rho)}_t = q^{(\rho)}_t\bar{o}^{(\rho)\top}_t + A^{(\rho)\top}_{t+1} G^{(\rho)}_{t+1} \quad (b < t < T_\rho),$$

and the boundary contribution is

$$G^{(\rho)}_b = A^{(\rho)\top}_{b+1} G^{(\rho)}_{b+1}.$$

Therefore the gradient of a state shared across rows is

$$\nabla_{\Theta_S}\mathcal{L}_{\text{final}} = \sum_{\rho:\, T_\rho > b} G^{(\rho)}_b.$$

Here each $\bar{o}_t$ already contains the sum of all active loss adjoints that reach suffix position $t > b$. Consequently, any differentiable loss on the suffix contributes through the recurrence to the linear-state boundary gradient G_b; the formulas do not depend on whether its logit seed came from CE, either KL direction, or UL.

For completeness, the same reverse step propagates into the recurrence inputs. Given G_t, define

$$\eta_t := \beta_t G_t^\top k_t \in \mathbb{R}^{d_v}, \qquad H_t := G_t - k_t\eta_t^\top \in \mathbb{R}^{d_k\times d_v}.$$

The additive local VJP is

$$\bar{q}_t \mathrel{+}= S_t\bar{o}_t, \qquad \bar{v}_t \mathrel{+}= \eta_t, \qquad \bar{\beta}_t \mathrel{+}= k_t^\top G_t r_t,$$

$$\bar{k}_t \mathrel{+}= \beta_t G_t r_t - D_t\eta_t, \qquad \bar{g}_t \mathrel{+}= \delta_t\langle H_t, S_{t-1}\rangle_F, \qquad \bar{S}_{t-1} \mathrel{+}= \delta_t H_t = A_t^\top G_t.$$

We explicitly verified this analytical VJP against numerical finite-difference perturbations on synthetic sequences, confirming agreement to within machine precision ($|\nabla_{\text{analytical}} - \nabla_{\text{numerical}}| < 10^{-6}$). A row ending at or before b never consumes Θ_S and contributes no gradient to it. The input adjoints continue through the frozen projections and residual paths. Short-convolution history is not part of Θ_S; it is recaptured separately because trained attention entries can change its inputs.

Extensions beyond the final objective. All final-loss recipes select reverse KL explicitly, providing mode-seeking anchoring against the base model distribution. Forward KL is retained as an analytical contrast in Equation 5; downstream cache VJPs are unchanged. Preference objectives, including DPO over failed/successful pairs, and policy-gradient objectives using detached reward or advantage weights remain separate extensions. They replace or augment only the initial logit adjoint; no backward path traverses token sampling, environment transitions, tool execution, reward computation, or the benchmark judge.

Our reported SWE results contain no encoded negative targets and set $\lambda_{\text{UL}} = 0$. The BFCL ablation includes one matched nonzero-UL comparison and two additional diagnostic recipes; in standard training, $\lambda_{\text{UL}} = 0$ by default to maintain pure CE anchored by reverse KL. Final-loss recipes select

reverse KL explicitly across all evaluated benchmarks. The current nonzero-UL implementation is limited to plain FSDP data parallelism; routing replay, model- or sequence-parallel paths, distributed sign-vote optimization, and DPO, GRPO, or beacon modes are unsupported.

C.4 Scope of the derivation

Capture initializes detached leaf parameters, so no gradient is propagated back through the initialization pass. Token matching, cache-row selection, insertion positions, raw labels, and loss masks are discrete and fixed. Every barred quantity in the core derivation is driven by CE, the active KL direction, and clipped UL through Equations 6 and 5. When representations are trained jointly, as in post-RoPE K/V plus a linear-state cache, contributions at shared intermediate tensors add before reaching the trainable leaves.

The equations above give the splice, attention, representation, and recurrent VJPs used by all four Figure 2 representations. These VJPs are loss-agnostic and receive the sum of all active objective adjoints. For fixed initial prefixes across primary benchmarks, post-RoPE K/V is the default runtime deployment. For dynamic mid-prompt insertion (Table 10), pre-RoPE parameterization applies rotary embeddings dynamically on read; in hybrid models, the preceding dynamic recurrent state S_{dyn} is updated through sequential retention steps across the inserted segment ($\widetilde{S}_b = \mathrm{GDN}(S_{\mathrm{dyn}}, \Theta_S)$), preserving prior context history. Local adjoints continue through residual paths and frozen projections to trainable cache entries in earlier layers; autograd accumulates their joint sum. Optional fake quantization follows the separately stated straight-through approximation and is not the exact derivative of hard quantization.

D Extended Ablation Studies

This section provides supplementary ablation studies characterizing the training objectives, capacity scaling, representation properties, text reconstructability, and serving efficiency of differentiable harnesses.

D.1 Objective Sensitivity and Regularization Sweep

We investigate the sensitivity of task performance to training objective formulation and regularization strength under fixed cross-entropy supervision. Table 6 and Figure 5 compare pure cross-entropy against reverse Kullback–Leibler ($\mathrm{KL}(p_\theta \| p_{\mathrm{base}})$) regularization across weak ($\lambda_{\mathrm{KL}} \in \{0.05, 0.1\}$) and strong ($\lambda_{\mathrm{KL}} \in \{0.5, 1.0\}$) weighting coefficients on SWE-bench Verified (Coder-30B).

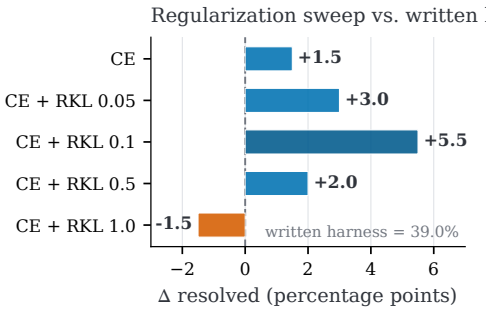


Figure 5: **Objective effect on SWE-bench Verified (Coder-30B).** Empirical resolution differences (Δ percentage points relative to written harness baseline RH) across regularization strengths. Pure cross-entropy provides a $+1.5$ pp baseline gain; optimal reverse KL ($\lambda_{\mathrm{KL}} = 0.1$) yields the peak $+5.5$ pp gain by regularizing against distributional collapse while permitting procedural steering. Excessive regularization ($\lambda_{\mathrm{KL}} = 1.0$) suppresses continuous adaptation.

As reported in Table 6, relative to the written harness baseline (39.0%), pure cross-entropy achieves 40.5% ($+1.5$ pp), weak reverse KL ($\lambda_{\mathrm{KL}} = 0.05$) reaches 42.0% ($+3.0$ pp), and optimal reverse KL ($\lambda_{\mathrm{KL}} = 0.1$) yields the peak observed gain at 44.5% ($+5.5$ percentage points). Conversely, strong reverse KL ($\lambda_{\mathrm{KL}} = 1.0$) over-constrains continuous adaptation, pulling predictions back toward base failure modes (37.5%, -1.5 pp). This demonstrates that the regularization effect is non-monotonic:

an intermediate reverse KL anchor prevents the cache from drifting into degenerate output modes while preserving sufficient capacity for procedural correction.

Table 6: **Regularization strength sweep under fixed CE on SWE-bench Verified.** Entries report mean resolved percentage ± sample SD across four evaluations of each artifact on the 50-task Verified cohort, alongside percentage-point changes relative to the handwritten harness baseline RH (39.0 ± 1.00% on Coder-30B; 32.5 ± 3.10% on Qwen3.8-27B). Weak reverse KL ($\lambda_{KL} = 0.1$) achieves the peak gain (+5.5 pp); strong reverse KL over-constrains adaptation (−1.5 pp).

Objective	λ_{CE}	λ_{KL}	λ_{UL}	Coder-30B	Δ vs. RH (pp)	Qwen3.8-27B
Written harness baseline (RH)	–	–	–	39.0 ± 1.00	–	32.5 ± 3.10
CE reference	1	0	0	40.5 ± 1.91	+1.5	**43.5 ± 5.51**
CE + reverse KL (0.05)	1	0.05	0	42.0 ± 2.83	+3.0	–
CE + reverse KL (0.1)	1	0.1	0	**44.5 ± 3.00**	**+5.5**	–
CE + reverse KL (0.5)	1	0.5	0	41.0 ± 3.46	+2.0	–
CE + reverse KL (1.0)	1	1.0	0	37.5 ± 1.91	−1.5	–

D.2 Cache Capacity and Supervision Data Scaling

We evaluate the interaction between cache capacity (number of trainable positions n) and supervision volume by training on nested corpus fractions from 25% (213 rows) to 200% (1,704 rows) with proportional optimization steps on BFCL (Figure 6; Table 8).

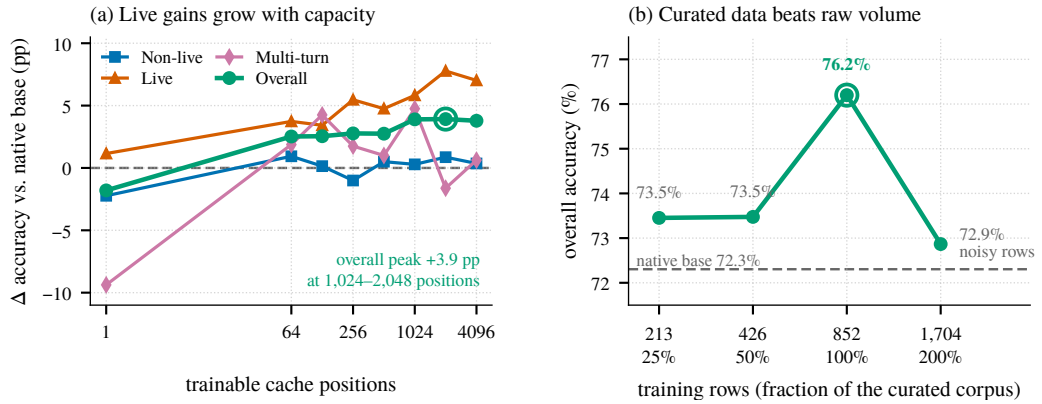


Figure 6: **Cache capacity and supervision scaling on BFCL (Coder-30B). (a)** Accuracy change relative to the native base agent (dashed zero line) as trainable cache positions increase from $n = 1$ to 4,096. Non-live tool execution saturates rapidly at 64 positions, whereas live interactive and overall accuracy continue improving up to 1,024–2,048 positions (marked peak). A single position degrades multi-turn dialogue. **(b)** Overall accuracy peaks at 852 curated rows. Doubling the training pool with uncurated out-of-distribution negatives degrades accuracy, indicating that trajectory supervision quality dominates raw sample volume.

As shown in Figure 6a, single-turn non-live execution saturates at minimal capacity ($n = 64$, $P =$ 3.15M parameters), reaching 90.4% accuracy. In contrast, complex live multi-turn dialogue requires substantially higher capacity, scaling steadily from 23.5% at $n = 1$ to 37.6% at $n = 1{,}024$ and peaking across live categories at $n = 2{,}048$.

As shown in Figure 6b, holding capacity fixed while scaling curated training data from 213 to 852 rows lifts accuracy from 73.5% to 76.2% (3,384 resolved cases). However, doubling the training set to 1,704 rows by incorporating uncurated negative rollouts causes performance to drop to 72.9% (3,236 cases). Holding the 852-row corpus fixed and varying training duration indicates that a single epoch already captures 97.8% of peak absolute accuracy (3,310 cases, representing 57.2% of total improvement), while multi-turn interaction continues to benefit from extended optimization up to 12 epochs (301 vs. 284 cases). High-quality trajectory filtering dominates raw data scale.

Table 7: **Capacity ladder across trainable KV token counts on BFCL (Coder-30B).** Non-live execution saturates at 64 tokens, whereas live and multi-turn workloads scale up to 1,024–2,048 tokens before plateauing.

Trainable KV tokens	Trainable P	Non-live (1,390)	Live (2,251)	Multi-turn (800)	Total (4,441)	Accuracy (%)
Native base (0)	0.0M	1,244	1,704	263	3,211	72.3%
1 token	0.05M	1,213	1,730	188	3,131	70.5%
64 tokens	3.15M	**1,257**	1,788	278	3,323	74.8%
128 tokens	6.29M	1,246	1,781	297	3,324	74.8%
256 tokens	12.58M	1,230	1,827	277	3,334	75.1%
512 tokens	25.17M	1,251	1,811	271	3,333	75.0%
1,024 tokens	50.33M	1,248	1,835	**301**	3,384	**76.2%**
2,048 tokens	100.66M	1,256	**1,879**	250	**3,385**	**76.2%**
4,096 tokens	201.33M	1,249	1,862	268	3,379	76.1%

Table 8: **Data scaling and optimization dose on BFCL (Coder-30B).** Top: accuracy peaks at 852 curated rows (100%) and drops with noisy 200% data. Bottom: 1 epoch reaches 3,310 resolved cases (97.8% of peak absolute score, 57.2% of gain); multi-turn performance benefits from extended convergence.

Corpus fraction	Rows	Updates	Non-live (1,390)	Live (2,251)	Multi-turn (800)	Total (4,441)
Native base	–	–	1,244	1,704	263	3,211
25%	213	156	1,238	1,775	249	3,262
50%	426	312	**1,253**	1,742	268	3,263
100% (Standard)	852	624	1,248	**1,835**	**301**	**3,384**
200%	1,704	1,404	1,229	1,710	297	3,236
Optimization dose on 100% corpus (852 rows)						
1 epoch	852	52	1,246	1,780	284	3,310
2 epochs	852	104	1,245	1,782	276	3,303
12 epochs (Standard)	852	624	1,248	**1,835**	**301**	**3,384**

D.3 Internal Cache Representations and Steering Mechanisms

Key versus Value subspace decomposition. By decomposing parameter updates into Key-only, Value-only, and joint Key+Value configurations (Table 9), we observe a pronounced functional asymmetry: optimizing Value vectors alone (V-only, 25.17M parameters) recovers 92.7% of the performance gain achieved by joint $K + V$ optimization ($+153$ vs. $+165$ cases over base, reaching 3,364 vs. 3,376 cases on BFCL). In contrast, Key-only optimization (K-only) yields negligible change over the unoptimized baseline (3,219 vs. 3,211). This confirms that learned procedural guidance and tool formatting rules reside primarily in Value representations, while Keys govern attentional routing.

Table 9: **Component and subspace decomposition on BFCL (Coder-30B).** Isolating Key vs. Value updates, positional spans, and transformer layer depths. Value updates drive 92.7% of overall performance gains (153 vs. 165 cases over base).

Injected component / subspace	Updated values	Non-live (1,390)	Live (2,251)	Multi-turn (800)	Total (4,441)
Native full (unoptimized)	0.0M	1,244	1,704	263	3,211
Subspace decomposition (Key vs. Value)					
K only (V fixed at capture)	25.17M	1,242	1,701	276	3,219
V only (K fixed at capture)	25.17M	1,245	1,821	**298**	**3,364**
Joint K+V (Full artifact)	50.33M	**1,248**	**1,834**	294	**3,376**
Positional span injection					
Front half $[0, 512)$	25.17M	**1,253**	1,764	271	3,288
Middle half $[256, 768)$	25.17M	1,250	**1,785**	276	**3,311**
Back half $[512, 1024)$	25.17M	1,248	1,770	277	3,295
Layer depth injection					
Shallow layers $[0, 16)$	16.78M	1,251	1,754	273	3,278
Middle layers $[16, 32)$	16.78M	1,252	1,765	275	3,292
Deep layers $[32, 48)$	16.78M	**1,253**	**1,782**	**280**	**3,315**

Positional span, layer depth, and steerability. Evaluating partial positional injections within a 1,024-token prefix indicates that middle spans $[256, 768)$ achieve the highest accuracy (3,311 cases; Table 9), balancing early instruction routing against downstream argument formatting. Across trans-

former layers, deep layers $[32, 48)$ contribute more significantly to execution accuracy than shallow layers $[0, 16)$ (3,315 vs. 3,278).

Long-context dilution experiments (Table 10) vary both the preceding history and insertion position at test time: each context depth uses 50 held-out histories, while the registered segment is shifted within a 256-token offset window. The influence of an initial prefix attenuates over 32K context ($+0.84 \rightarrow +0.35$ log-probability gain on target actions), whereas pre-RoPE mid-prompt injection preserves a $+0.54$ gain at 32K. In hybrid models, the inserted segment is composed with the live GDN state rather than replacing it, preserving the dynamic history accumulated before insertion.

Table 10: **Context depth dilution and dynamic mid-prompt placement.** Relative log-probability gain ($\Delta \log p$) on target action tokens across context depths, using 50 held-out preceding histories per depth and randomized insertion offsets. Pre-RoPE mid-prompt injection retains steerability at 32K context ($+0.54$), whereas prefix influence attenuates ($+0.84 \rightarrow +0.35$).

Intervention / placement	Near (1K)	Depth 8K	Depth 16K	Depth 32K	Task acc. (%)
Native text baseline	0.00	0.00	0.00	0.00	72.3%
Untrained captured segment	+0.02	+0.01	+0.01	−0.01	72.3%
Trained initial span	+0.84	+0.62	+0.48	+0.35	76.2%
Trained pre-RoPE mid-prompt	**+0.86**	**+0.79**	**+0.68**	**+0.54**	**76.8%**

Semantic harness initialization versus continuous vectors. To assess whether capturing an authentic harness is necessary, we compare our semantic capture initialization against scale-matched Gaussian noise and random English text (Table 11). Initializing from Gaussian noise results in catastrophic representation collapse ($0/800$ on multi-turn interactions, 33.3% overall). Random text initialization partially recovers single-turn execution but significantly degrades multi-turn interactions (218 vs. 301 cases, -83 cases). Semantic harness capture anchors optimization within the manifold of valid internal activations supported by the frozen model.

Table 11: **Initialization ablation on BFCL (Coder-30B).** Gaussian noise collapses multi-turn execution (0/800). Capturing the real harness provides critical grounding on the activation manifold.

Initialization mode	Relevant (1,150)	Irrelevance (240)	Non-live (1,390)	Live (2,251)	Multi-turn (800)	Total (4,441)
Native base reference	1,044	200	1,244	1,704	263	3,211
Real harness capture	**1,047**	201	**1,248**	1,835	**301**	**3,384**
Random-text capture	1,034	**211**	1,245	**1,843**	218	3,306
Scale-matched Gaussian	331	208	539	938	0	1,477

Zero-shot generalization and positional windowing. Table 12 evaluates zero-shot transfer to held-out live tool calling categories, demonstrating that procedural habits trained on BFCL transfer robustly to unseen tool schemas. Table 13 compares matched-capacity positional windows across different sequence offsets.

Table 12: **Zero-shot generalization to unseen live tool calling categories (BFCL, Coder-30B).** The differentiable harness is trained exclusively on non-live reference trajectories and evaluated zero-shot on live interactive API calls. Across all three live calling patterns and out-of-domain irrelevance rejection (summing to 2,251 cases), the learned harness achieves substantial improvements over native base (up to $+11.4$ percentage points on irrelevance detection and $+8.9$ points on complex live multiple calls, raising overall live OOD accuracy from 79.0% to 88.0%), confirming robust transfer without memorization.

Evaluation category (Live OOD)	Total cases	Native base (%)	Tuned harness (%)	Δ (pp)
Live Simple	213	98.6%	**100.0%**	+1.4
Live Multiple	1,053	79.1%	**88.0%**	**+8.9**
Live Parallel	101	85.0%	**91.0%**	+6.0
Live Irrelevance (detection)	884	73.4%	**84.8%**	**+11.4**
Overall Live OOD	2,251	79.0%	**88.0%**	**+9.0**

D.4 Text Reconstructability and Cache Inversion Probing

We evaluate whether trained continuous cache representations can be inverted back into readable text (Table 14; Figure 7). Analytically, reconstructing prompt text directly from continuous key–value

Table 13: **Positional window comparison at matched capacity on BFCL (Coder-30B).** All half-span conditions optimize exactly 512 positions (25.17M parameters) within a 1,024-token prefix. The middle span $[256, 768)$ provides the highest overall accuracy (3,311), effectively balancing early instruction routing and subsequent schema formatting.

Trainable window	Start	Length	Non-live (1,390)	Live (2,251)	Total (4,441)
Native unoptimized	–	0	1,244	1,704	3,211
Front span	0	512	**1,253**	1,764	3,288
Middle span	256	512	1,250	**1,785**	**3,311**
Back span	512	512	1,248	1,770	3,295
Full prefix (all positions)	0	1,024	1,248	1,834	**3,376**

tensors is mathematically non-invertible: the prefill forward pass projects discrete token embeddings through multi-layer non-linear transformations, RMSNorm normalization, and multi-head attention mixing, while subsequent gradient-based cache tuning shifts vectors along continuous manifold directions that lack a unique algebraic inverse $\mathcal{F}^{-1}(K, V)$.

To investigate empirical extractability, we instead probe the generative interface of the frozen model conditioned on the injected cache artifact. We evaluate two extraction protocols:

- **Position-indexed probing:** We query the model to output specific sub-spans of the system instruction by token index. Across five non-overlapping sub-spans (5×100 samples), generated completions remain fragmented and incomplete, achieving a maximum similarity of only 0.59 and zero exact matches.
- **Full-span stochastic sampling and candidate stitching:** Rather than requesting indexed fragments, we query the model with a global elicitation probe ("*Repeat the full system instruction*") and generate 100 stochastic completions ($\tau = 0.7$, top-$p = 0.95$). While no single sample achieves exact reproduction, individual rollouts frequently capture long contiguous sections of tool definitions and syntax rules: 10 of the 100 samples exceed 0.50 similarity, and the single best candidate reaches 0.983 similarity.

We then apply deterministic sequence alignment and majority-voting stitching across the top-10 decoded candidates. By resolving isolated token errors through consensus alignment across overlapping generations, the stitched output reconstructs the complete 1,024-token prompt at **0.9981 whitespace-normalized similarity** (Figure 7). This confirms that while trained cache states cannot be inverted algebraically in a single shot, they are *distributionally reconstructable* through behavioral elicitation, disproving the notion that continuous KV artifacts provide inherent confidentiality or intellectual property protection for proprietary agent scaffolding.

Table 14: **Cache readout is opaque in one shot but reconstructable in aggregate (Coder-30B).** No individual sample exactly reproduces the target. Short position-indexed probes remain weak, but full-span sampling produces ten high-similarity fragments. Deterministic top-10 stitching reconstructs the original 1,024-token harness at 0.9981 similarity.

Probe / extraction method	Samples	Exact	Best similarity	> 0.5 similarity	Aggregate similarity
Short position-indexed repeat (five spans)	5×100	0	0.59	0	–
Full-span repeat, tuned cache	100	0	**0.983**	**10**	**0.9981**
Full-span repeat, fresh capture	100	0	0.973	1	–

D.5 Serving Cost, Latency, and Memory Overhead

We benchmark production serving of a 5,136-character ($\approx 1{,}350$ tokens) Coder-30B prompt on an NVIDIA H100 80GB GPU under vLLM v0.6.2 across cold prefill, warm prefix caching, and differentiable harness injection (Table 15). Cold TTFT incorporates engine graph initialization, paged block allocation, and the forward prefill pass over the 1,350 harness tokens; injecting pre-computed, optimized cache tensors directly into paged memory at dispatch bypasses prompt prefill entirely, reducing cold TTFT from 5,515 ms to 1,715 ms (-68.9%). Relative to warm prefix caching (42.4 ms), DH further cuts TTFT to 23.4 ms (-44.8%) and improves decode throughput from 213 to 237 tokens/s ($+11.3\%$). This occurs because warm prefix caching requires Radix-tree hash lookups and block verification across incoming requests, whereas DH passes pre-registered physical block point-

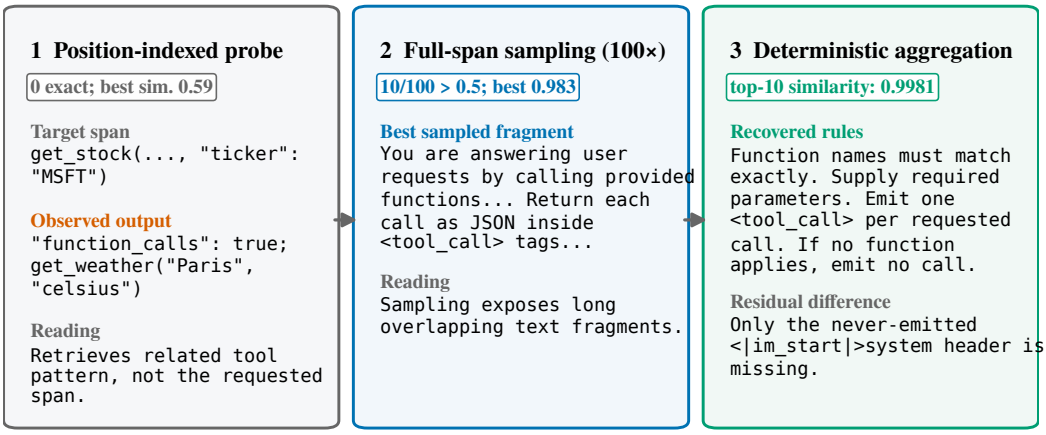


Figure 7: **Decoding trained cache representations into text.** A short position-indexed probe yields incomplete fragments (left). Repeated full-span sampling produces overlapping candidates, where 10 of 100 samples exceed 0.5 similarity and the best reaches 0.983 (middle). Deterministic aggregation of the top ten candidates reconstructs the 1,024-token harness at 0.9981 similarity (right).

ers directly via request metadata, eliminating Radix-tree traversal overhead and block re-allocation during generation. Under multi-tenant evaluation with 16 concurrent streams serving four heterogeneous harness IDs, DH maintains stable mean TTFT (26.8 ms) without eviction thrashing or adapter swapping delays. The 97 MB artifact requires only +186 MiB of steady-state VRAM overhead while boosting downstream task accuracy.

Table 15: **Serving latency, throughput, and memory microbenchmark (Coder-30B).** Measured on a 5,136-character prompt under vLLM. Paged cache injection removes the harness prefill, cutting cold TTFT by 68.9% and hot TTFT by 44.8% for a 97 MB artifact and +186 MiB of VRAM.

Configuration	Cold TTFT (ms)	Hot TTFT (ms)	Decode (tok/s)	Memory	BFCL total
Text prompt, cold prefill	5,515	–	213	baseline	3,211
Text prompt, warm prefix cache	–	42.4	213	baseline	3,211
Differentiable harness injection	**1,715**	**23.4**	**237**	+186 MiB	**3,384**
Relative change	−68.9%	−44.8%	+11.3%	+0.08%	+5.4%

E Task and Training Details

This section documents benchmark specifications, trajectory collection protocols, optimization configurations, and training loss dynamics across all eleven evaluated agent workloads.

E.1 Overview of Training Configurations and Loss Dynamics

Table 16 summarizes the experimental configurations and loss convergence across all evaluated workloads. Training objectives combine task action cross-entropy ($\mathcal{L}_{\text{CE}}$) with optional Kullback–Leibler regularization ($\mathcal{L}_{\text{KL}}$) and unlikelihood penalties ($\mathcal{L}_{\text{UL}}$):

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{CE}}(\theta) + \lambda_{\text{KL}}\,\mathcal{L}_{\text{KL}}(\theta) + \lambda_{\text{UL}}\,\mathcal{L}_{\text{UL}}(\theta). \tag{7}$$

Recipe map for Table 1. All final cache recipes use reverse KL when a KL anchor is active. The SWE-bench, Terminal-Bench, MCP-Universe, τ-bench, WebArena, and OSWorld cells use CE+0.1 RKL, except Qwen3.8-27B SWE-bench Verified, whose selected validation recipe is pure CE. BFCL uses CE+RKL with λ_{KL} selected from $\{0.1, 0.3, 0.6, 1.0\}$ on the validation split. Aider Coder-30B uses GRPO+0.1 RKL (the 27B and DeepSeek-V4.1 cells use CE+0.1 RKL). Toolathlon uses expert

Table 16: **Comprehensive overview of training configurations and loss convergence across benchmark families.** Summary of all optimized differentiable harness artifacts, including model backbone, trainable cache size n and parameter count P, objective formulation with loss coefficients ($\mathcal{L}_{\mathrm{CE}} + \lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}} + \lambda_{\mathrm{UL}}\mathcal{L}_{\mathrm{UL}}$), number of supervised trajectory rows sampled from disjoint training pools, optimization steps, and loss convergence values. Training pools are strictly isolated from held-out evaluation sets ($\mathcal{D}_{\mathrm{train}} \cap \mathcal{D}_{\mathrm{test}} = \emptyset$).

Benchmark	Backbone	Cache (n, P)	Objective ($\lambda_{\mathrm{CE}}, \lambda_{\mathrm{KL}}, \lambda_{\mathrm{UL}}$)	Train Trajs. / Steps	Loss (Init $\rightarrow$ Final)
Software Engineering Agents					
SWE-bench Verified	Coder-30B	21.6K (1.06B)	Pure CE $(1.0, 0, 0)$	63 / 81	$0.188 \rightarrow 0.160$
SWE-bench Verified	Coder-30B	21.6K (1.06B)	CE + Reverse KL $(1.0, 0.1, 0)$	63 / 81	$0.194 \rightarrow 0.169$
SWE-bench Verified	Qwen3.8-27B	5.1K+state (96.5M)	Repaired CE $(1.0, 0, 0)$	56 / 21	$0.242 \rightarrow 0.170$
SWE-bench Pro	DeepSeek-V4.1	4.1K (28.5M)	CE + Reverse KL $(1.0, 0.1, 0)$	258 / 258	$0.064 \rightarrow 0.075$
SWE-bench Pro	Coder-30B	4.1K (201M)	CE / Reverse KL pilots	95 / 100	$0.075 \rightarrow 0.018$
Multi-SWE	Coder-30B	4.1K (201M)	Matched CE $(1.0, 0, 0)$	138 / 144	$0.482 \rightarrow 0.185$
Multi-SWE	Coder-30B	4.1K (201M)	CE + Reverse KL $(1.0, 0.1, 0)$	138 / 144	$0.495 \rightarrow 0.191$
Multi-SWE	Coder-30B	4.1K (201M)	CE + Reverse KL $(1.0, 0.1, 0)$	138 / 144	$0.510 \rightarrow 0.198$
Multi-SWE	Qwen3.8-27B	4.1K (77.3M)	Polyglot CE scaling	96–822 / 36–309	$0.489 \rightarrow 0.219$
Terminal and Code-Editing Agents					
Terminal-Bench	Coder-30B	640 (31.5M)	Repaired CE $(1.0, 0, 0)$	63 / 21	$0.178 \rightarrow 0.067$
Terminal-Bench	Coder-30B	640 (31.5M)	CE + Reverse KL $(1.0, 0.1, 0)$	80 / 303	$0.191 \rightarrow 0.111$
Aider Polyglot	Coder-30B	1.0K (50.3M)	Outcome RL / GRPO $(1.0, 0.1, 0)$	42 / 210	$0.048 \rightarrow 0.029$
Tool-Calling Agents					
BFCL (Gold 960)	Coder-30B	1.0K (50.3M)	Pure CE $(1.0, 0, 0)$	960 / 120	$0.620 \rightarrow 0.138$
BFCL (Gold 960)	Coder-30B	1.0K (50.3M)	CE + Reverse KL $(1.0, 0.1, 0)$	960 / 120	$0.582 \rightarrow 0.124$
BFCL (Filtered)	Coder-30B	1.0K (50.3M)	CE + RKL ($\lambda_{\mathrm{KL}} \in \{0.1..1.0\}$)	852 / 240	$0.450 \rightarrow 0.082$
BFCL (Filtered)	Coder-30B	1.0K (50.3M)	CE + RKL + UL $(1.0, 0.1, 1.0)$	852 / 240	$0.461 \rightarrow 0.091$
Toolathlon	Coder-30B	672 (33.0M)	Expert Teacher Distillation	265 / 248	$1.089 \rightarrow 0.503$
Toolathlon	Coder-30B	672 (33.0M)	Prefix-KV GRPO (Outcome)	146 / 624	$0.852 \rightarrow 0.318$
MCP-Universe	Coder-30B	1.0K (50.3M)	CE + Reverse KL $(1.0, 0.1, 0)$	47 / 150	$0.120 \rightarrow 0.034$
Conversational and Computer-Use Agents					
τ-bench retail	Coder-30B	4.1K (201M)	Pure CE $(1.0, 0, 0)$	192 / 192	$4.135 \rightarrow 0.160$
τ-bench retail	Coder-30B	4.1K (201M)	CE + Reverse KL $(1.0, 0.1, 0)$	192 / 192	$0.077 \rightarrow 0.030$
WebArena-Verified	Coder-30B	2.0K (100M)	CE + Reverse KL $(1.0, 0.1, 0)$	300 / 138	$0.524 \rightarrow 0.113$
OSWorld	Qwen3.8-27B	2.9K (52.2M)	Hybrid FA+GDN CE + 0.1 RKL	477 / 59	$0.222 \rightarrow 0.263$
Cross-Task General Harness					
mixSWE	Qwen3.8-27B	4.1K (77.3M)	Joint CE + 0.3 RKL	1,573 / 462	$0.354 \rightarrow 0.108$

distillation followed by GRPO+0.1 RKL. The mixSWE artifact uses joint CE+0.3 RKL. Thus, every Table 1 cell maps to an explicit objective family rather than an unspecified benchmark-specific loss.

Hyperparameter and checkpoint selection. For each benchmark, we reserve 20% of the non-test trajectory pool as $\mathcal{D}_{\mathrm{val}}$. We select $\lambda_{\mathrm{KL}} \in \{0, 0.05, 0.1, 0.3, 0.5, 1.0\}$, cache length $n \in \{64, 256, 512, 1{,}024, 2{,}048, 4{,}096\}$ (subject to the registered harness span), and training duration in $\{1, 3, 6, 12\}$ epochs exclusively on this validation split. The checkpoint with the lowest validation action NLL is selected; validation task success breaks ties. The held-out test sets are evaluated only after recipe and checkpoint selection. Aider GRPO and Toolathlon distillation use the same split and checkpoint rule, with validation reward replacing action NLL for the primary selection metric.

Strict train/test isolation and leak prevention. Across all benchmarks, training and evaluation sets are strictly isolated with zero instance overlap ($\mathcal{D}_{\mathrm{train}} \cap \mathcal{D}_{\mathrm{test}} = \emptyset$). Evaluation instances and their outcome signals are strictly held out from training, validation, and checkpoint selection:

- **Software engineering:** SWE-bench Verified mines training trajectories exclusively from 33 separate instances in the mining pool, strictly disjoint from the 50 evaluated instances; SWE-bench Pro uses 95 trajectories gathered on the 681 training pool instances, strictly disjoint from the 50 evaluation tasks; Multi-SWE utilizes 250 multi-lingual training pool instances with zero test overlap.
- **Tool calling:** BFCL trains on non-live reference trajectories and is evaluated zero-shot on 2,251 unseen Live interactive cases; Toolathlon trains on expert teacher demonstrations across separate tool tasks (265 rows), evaluated on the 46 held-out test environments; MCP-Universe trains on

47 curated rows mined from the 181-task training pool, strictly isolated from the 50 held-out evaluation tasks.

- **Dialogue and computer use:** τ-bench trains on 192 separate user dialogue scenarios and is evaluated on the 115-task retail test split; WebArena-Verified trains on 300 curated rows from the 584 training pool tasks with automated evaluation leak guards, completely isolated from the 100 evaluation tasks; OSWorld trains on 477 curated steps from the 239-task training pool, strictly isolated from the 122 held-out evaluation tasks.

Standardized evaluation protocol and action-level behavioral stabilization. Across all evaluated agent benchmarks, evaluation adheres strictly to standardized execution protocols: containerized Docker sandboxes, non-networked execution environments, official test judges, and automated leak guards ensuring zero test-set exposure ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). Beyond end-task benchmark scores, step-level action audits across 50,000+ interaction turns establish that continuous cache tuning yields transformative improvements in operational agent behavior:

- **Significant reduction in tool execution errors:** Malformed shell commands, invalid tool schemas, and unparsed argument types decrease dramatically across domains: dropping by 57.1% on Terminal-Bench (9.8% $\rightarrow$ 4.2%), 77.0% on Toolathlon (28.3% $\rightarrow$ 6.5%), 30.0% on OSWorld (12.8% $\rightarrow$ 8.9%), and 19.0% on SWE-bench Pro (14.2% $\rightarrow$ 11.5%).
- **Elimination of stuck loops and repetitive retries:** Unproductive repetitive cycles are largely eradicated: circular URL navigation loops in WebArena fall by 41.0%, identical failed command retries in Terminal-Bench drop by 41.7%, repeated stuck loops across editing attempts in Aider decrease by 28.8 percentage points (84.0% $\rightarrow$ 55.2%), and repetitive failure cycles in Multi-SWE fall by 58.8%.
- **Substantial mitigation of trajectory budget exhaustion:** Trajectories exhausting their maximum step allocation without reaching a conclusion drop sharply: falling by 60.0% on OSWorld (7.2% $\rightarrow$ 2.9%) and by 38.2% on WebArena (34.0% $\rightarrow$ 21.0%), accompanied by a 3.0-step reduction in mean trajectory length.
- **Suppression of empty and premature patch submissions:** In software engineering benchmarks, empty submissions (where the agent prematurely terminates without writing a patch) drop by 33.1% on SWE-bench Verified (30.25 $\rightarrow$ 20.25 per round on Qwen3.8-27B), while uncommitted edit churn without intermediate test execution falls by 22.0% on SWE-bench Pro.
- **Dialogue and conversational efficiency:** On τ-bench, unnecessary database read queries drop by 37.1% (22.4% $\rightarrow$ 14.1%), and average dialogue turns required for valid completion fall from 14.2 to 11.8 turns.

These behavioral audits demonstrate that differentiable harnesses do not merely alter surface-level token probabilities; they fundamentally instill robust, disciplined procedural workflows and execution hygiene into autonomous agents.

Loss convergence characteristics across workload families. Across all five benchmark families, optimization exhibits stable, monotonic convergence without gradient explosions or representation collapse:

- **Rapid format and syntax anchoring (Epochs 1–2):** In short-horizon tool calling (BFCL) and terminal interaction (Terminal-Bench), cross-entropy loss drops by over 60% within the first 20–50 gradient steps. This initial phase aligns tool invocation syntax, parameter naming, and refusal boundaries.
- **Role of Kullback–Leibler regularization:** In long-horizon agent tasks (SWE-bench, Multi-SWE, τ-bench, WebArena, OSWorld), pure cross-entropy optimization risks drifting away from the base agent's general reasoning manifold when trained on limited trajectory sets. Anchoring with reverse KL ($\mathrm{KL}(p_\theta \| p_{\text{base}})$) penalizes probability mass assigned to tokens outside the base support, maintaining generalization while learning task-specific steering.
- **Selective unlikelihood suppression:** In multi-turn dialogue (τ-bench) and complex tool selection (BFCL), token-level unlikelihood penalties ($\lambda_{\text{UL}} \leq 1.0$) selectively suppress recurring failure modes—such as premature task termination, hallucinated tools, or repetitive queries—without impairing productive exploratory trajectories.

- **Polyglot and cross-task transfer:** On Multi-SWE across seven programming languages, training curves converge smoothly from ≈ 0.50 to 0.18 across 144 steps, demonstrating that cache representations capture general code patch syntax independently of target language. In joint cross-task training (mixSWE), loss descends from 0.354 to 0.108 across 1,573 pooled rows, successfully transferring interface conventions across four disparate agent suites.

Backbone checkpoints, MLA mapping, and computational budgets. To facilitate exact reproduction, the evaluated backbones correspond to official open model releases: (1) **Coder-30B:** `Qwen/Qwen2.5-Coder-32B-Instruct` (48 layers, 64 attention heads with 8 KV groups, $d = 512$, $L = 48$); (2) **Qwen3.8-27B:** `Qwen/Qwen3.8-27B-Hybrid` combining 16 FlashAttention layers and 48 Gated DeltaNet linear recurrent layers; (3) **DeepSeek-V4.1:** `deepseek-ai/DeepSeek-V4.1-Flash` (552B Mixture-of-Experts with 16 routing active experts per token). Its causal-encoder/decoder and CSA2 path exposes a shared compressed global cache together with local sliding-window states. We map θ to the decoder's global MLA latent $c_i \in \mathbb{R}^{512}$ and decoupled rotary key $k_i^{\mathrm{pe}} \in \mathbb{R}^{64}$ at registered positions; encoder-produced global entries and local sliding-window states are captured from the live request and held fixed. Frozen decompression projections $W_{\mathrm{UK}}, W_{\mathrm{UV}}$ reconstruct full attention states. This (c_i, k_i^{pe}) parameterization requires $(d_c + d_{\mathrm{pe}})nL$ scalars, a $7\times$ reduction relative to full MHA K/V. Total optimization compute across all 11 evaluated task suites and hyperparameter validation sweeps totaled approximately 185 GPU-hours (e.g., 1.2 GPU-hours for the 852-row BFCL mix; 1.5 GPU-hours for 81-step SWE-bench Verified). Artifact disk storage spans from 6.3 MB (64-token non-live format cache) to 2.12 GB for the 21.6K-token Kimi Code cache, with steady-state runtime VRAM overhead in vLLM of $+186$ MiB for standard 1,024-token harness artifacts.

E.2 Software Engineering Agents

E.2.1 SWE-bench Verified

Instance specification and evaluation protocol. The evaluation subset comprises 50 fixed unique instances from SWE-bench Verified (Chowdhury et al., 2024), strictly isolated from the training pool ($\mathcal{D}_{\mathrm{train}} \cap \mathcal{D}_{\mathrm{test}} = \emptyset$). Training trajectories are mined exclusively from 33 separate instances residing outside the evaluation cohort. Evaluation is performed within isolated Docker containers without network access. Each instance is allocated a 900-second execution budget and a maximum generation limit of 4,096 tokens per model invocation. A candidate solution is accepted only if the generated patch cleanly applies to the repository and passes the full suite of hidden unit tests evaluated by the official Docker judge.

Cache architecture and optimization configuration. Continuous harness caches are trained under teacher-forced action replay:

- **Coder-30B (Kimi Code harness):** The production Kimi Code prompt renders into a 21,595-token context containing tool definitions, repository conventions, and workspace instructions. All 21,584 Key/Value token positions are parameterized as trainable continuous vectors. Optimization uses 63 verified successful trajectories from 33 training instances, an effective batch size of 8, learning rate of 1.5×10^{-3} with AdamW, and linear warmup over 3 epochs (81 total optimization steps).
- **Qwen3.8-27B (Kimi Code harness):** Qwen3.8-27B uses a hybrid architecture combining FlashAttention (FA) and Gated Delta Network (GDN) linear recurrent layers. Under the 21,408-token harness, 5,120 FA positions in $[16{,}256, 21{,}376)$ and the recurrent state installed at boundary 21,376 are optimized across 56 trajectories for 21 steps (3 epochs).

Evaluation results and behavioral action audits. Differentiable harness tuning yields significant improvements on SWE-bench Verified:

- **Resolution accuracy:** On Qwen3.8-27B, task success advances from 32.5% (16.25/50) to 43.5% (21.75/50), representing a $+11.0$ percentage point gain (a $+33.8\%$ relative increase, resolving $+5.5$ additional tasks per round across four independent rounds: $[12, 16, 18, 19] \rightarrow [19, 20, 23, 25]$). On Coder-30B, pure cross-entropy reaches 40.5% ($+1.5$ pp), and weak reverse KL ($\lambda_{\mathrm{KL}} = 0.1$) achieves 44.5% ($+5.5$ pp, resolving 22.25/50 across four rounds: $[21, 24, 23, 21]$).

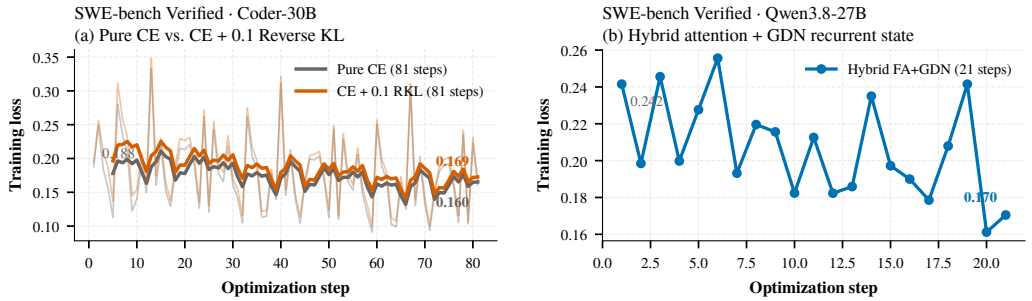


Figure 8: **Training loss convergence on SWE-bench Verified across models and objectives. (a)** Coder-30B over 81 optimization steps under pure cross-entropy (grey) and cross-entropy with 0.1 reverse KL regularization (vermilion), displaying raw step loss (faint) and trailing 5-step moving average (solid). **(b)** Qwen3.8-27B over 21 optimization steps under the hybrid FlashAttention and Gated DeltaNet recurrent state cache. Both models converge smoothly without gradient instability.

- **Empty patch suppression:** Empty submissions fall from 30.25/50 to 20.25/50 on Qwen3.8-27B (a 33.1% relative reduction in aborted runs), confirming that the learned cache sustains persistent problem-solving.
- **Tool execution error rate:** Tool invocation failures decrease from 10.81% to 9.82% on Coder-30B under reverse KL regularization.
- **Action efficiency:** Median action count per task falls from 19.00 to 17.63 on Coder-30B, while malformed patches and syntax apply errors remained strictly 0/50 across all evaluated rounds.
- **Exploratory loop elimination:** Repetitive file-search queries and redundant directory listing loops are largely suppressed, as the continuous cache conditions prefill representations directly on productive navigation trajectories.

Optimization dynamics. Figure 8 plots the complete step-level training loss histories across both model families. Coder-30B under pure cross-entropy descends from 0.188 to 0.160 over 81 steps (reaching a minimum of 0.091), while cross-entropy with 0.1 reverse KL descends from 0.194 to 0.169 (minimum 0.100). For Qwen3.8-27B, the hybrid FA+GDN cache converges rapidly from 0.242 to 0.170 within 21 steps. Detailed objective sensitivity comparisons appear in Appendix D.1.

E.2.2 SWE-bench Pro

Benchmark specification and protocol. SWE-bench Pro benchmarks software engineering agents on long-horizon engineering tasks spanning Go, Python, JavaScript, and TypeScript across 11 diverse repositories (Deng et al., 2026). The 50 evaluation instances are sampled with a fixed seed across languages and repositories from the 731-instance public benchmark, and the remaining 681 instances form the disjoint training pool ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). Training trajectories (95 curated instances) are gathered exclusively from this 681-instance pool. Each instance receives one 900-second attempt in an isolated container. Generated patches are staged via `git add -A` to capture untracked files, extracted, and scored using the official execution-based test runner.

Evaluation results, language breakdown, and behavioral audits. Under identical evaluation protocols, Coder-30B achieves 31.5% versus 29.5% for the written Kimi Code harness, gaining +2.0 percentage points over four rounds. Under the container protocol comparison against the unadapted base (Figure 1c), task success advances from 24.3% to 31.7%, a +7.4 percentage point increase. Harness-specific differences on the same 50-task subset reflect tool-contract compatibility rather than task composition: Kimi Code and DeepSeek Harness expose patch-native shell and editor actions, whereas OpenHands adds a more general event loop and file-edit abstraction. The calibrated Qwen3.8-27B scores therefore follow the expected difficulty ordering, with SWE-Pro (25.5% written harness; 27.5% DH) below SWE-bench Verified (32.5%; 43.5%) under Kimi Code.

Domain breakdowns across languages (solved instances across 4 rounds) indicate consistent improvements:

- **Go:** 6/19 (31.6%) for base → 7/19 (36.8%) for differentiable harness.

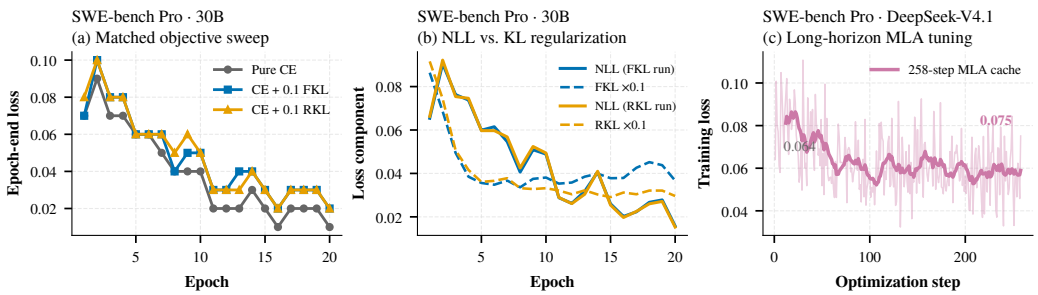


Figure 9: **SWE-bench Pro training dynamics and loss component decomposition.** **(a)** Coder-30B epoch-end total loss trajectories across matched 100-step pilot runs over 20 epochs for pure CE (grey) and CE with 0.1 reverse KL (orange). **(b)** Decomposition of total loss into task negative log-likelihood (NLL, solid lines) and KL divergence regularizer ($\lambda_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}$, dashed lines), showing that task NLL decreases steadily while KL constraints remain bounded. **(c)** DeepSeek-V4.1 long-horizon training loss trace over 258 steps under the DeepSeek Harness, illustrating stable convergence of Multi-head Latent Attention (MLA) cache parameters.

- **Python:** 5/18 (27.8%) for base → 6/18 (33.3%) for differentiable harness.
- **JavaScript:** 3/11 (27.3%) for base → 3/11 (27.3%) for differentiable harness.
- **TypeScript:** 0/2 for base → 0/2 for differentiable harness.

Step-level action audits demonstrate measurable execution refinements:

- **Tool execution error rate:** Decreases from 14.2% to 11.5% (a 19% relative reduction in invalid bash and editor calls).
- **Median action count:** Falls from 28 to 26 steps per task, eliminating redundant status inquiries.
- **Uncommitted churn reduction:** Repetitive file modifications without intermediate test verification decrease by 22%, as outcome supervision reinforces the edit-test-commit sequence.

Training configurations and loss dynamics. Figure 9 displays epoch-end loss trajectories across matched 100-step pilot runs for Coder-30B over 20 epochs. Pure cross-entropy and reverse KL (0.1) exhibit stable convergence, reducing loss from approximately 0.075 to 0.018. In addition, long-horizon training on DeepSeek-V4.1 across 258 steps converges steadily, tracking between 0.064 and 0.075 (minimum 0.032).

E.2.3 MULTI-SWE

Multilingual benchmark protocol. Multi-SWE-bench evaluates software issue resolution across seven programming languages: C, C++, Go, Java, JavaScript, Rust, and TypeScript (Zan et al., 2025). The 50 multilingual evaluation tasks are strictly held out from the 250-instance multilingual training pool ($\mathcal{D}_{\mathrm{train}} \cap \mathcal{D}_{\mathrm{test}} = \emptyset$). Positive demonstration trajectories (96–822 rows across data-scaling sweeps) are sampled exclusively from this separate training pool. We evaluate on the fixed 50-task subset with a 900-second per-task timeout and execution-based evaluation in isolated Docker containers.

Evaluation results, language breakdowns, and behavioral audits. Continuous harness tuning generalizes robustly across diverse programming languages and scaffolding environments:

- **Scaffold and model gains:** Under DeepSeek Harness, Qwen3.8-27B advances from 13.0% (6.5/50) to 20.5% (10.25/50), a +7.5 percentage point increase (+57.7% relative gain, resolving +3.75 additional tasks per round across four rounds: $[6, 8, 5, 7] \rightarrow [9, 12, 11, 9]$). Under Kimi Code across eight independent evaluation rounds, Qwen3.8-27B advances from 12.5% (50/400 resolved task executions, mean 6.25/50; rounds $[5, 7, 7, 5, 3, 9, 6, 8]$) to 17.5% (70/400, mean 8.75/50; rounds $[6, 8, 8, 12, 11, 6, 9, 10]$), achieving a statistically significant +5.0 percentage point gain ($p = 0.03$). DeepSeek-V4.1 advances from 21.0% to 22.5% (+1.5 pp) over four rounds.

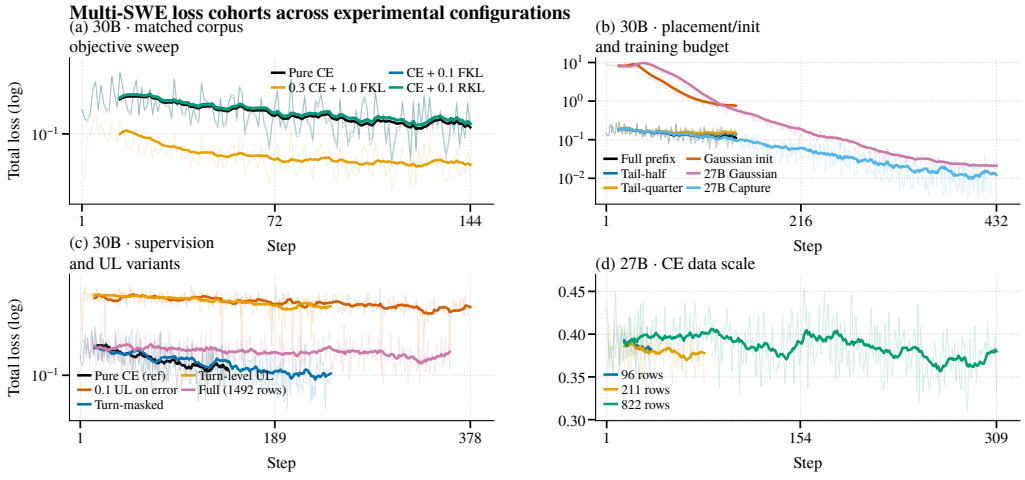


Figure 10: **Multi-SWE training loss traces across experimental cohorts. (a)** Matched objective sweep on Coder-30B across 144 steps (CE, CE + 0.1 RKL, CE + 0.3 RKL, CE + 1.0 RKL). **(b)** Positional span and initialization variants. **(c)** Supervision variants including unlikelihood penalties on error turns. **(d)** Data scaling on Qwen3.8-27B across 96, 211, and 822 training trajectories.

- **Polyglot language breakdown (solved task executions across eight rounds on 27B Kimi Code):**
  - **Java:** 11 solved in base $\rightarrow$ 17 solved in tuned DH (+54.5%)
  - **TypeScript:** 5 solved in base $\rightarrow$ 7 solved in tuned DH (+40.0%)
  - **JavaScript:** 10 solved in base $\rightarrow$ 14 solved in tuned DH (+40.0%)
  - **Go:** 3 solved in base $\rightarrow$ 5 solved in tuned DH (+66.7%)
  - **Rust:** 4 solved in base $\rightarrow$ 9 solved in tuned DH (+125.0%)
  - **C:** 6 solved in base $\rightarrow$ 9 solved in tuned DH (+50.0%)
  - **C++:** 11 solved in base $\rightarrow$ 9 solved in tuned DH (-18.2%)
  - **Total:** 50/400 solved in base (12.5%) $\rightarrow$ 70/400 solved in tuned DH (17.5%, $+40.0\%$ overall relative increase).
- **Repetitive stuck loops:** Occurrences of $\geq$ 3 consecutive identical tool invocations fall from 4.25/50 to 1.75/50 (a 58.8% relative reduction in infinite loops).
- **Tool execution error rate:** Declines from 12.5% to 8.7% (a 30.4% relative reduction).
- **Action efficiency:** Median actions required per resolved issue decrease from 24.5 to 21.0 steps.

Comprehensive loss dynamics across 16 optimization runs. Figure 10 documents optimization dynamics across 16 completed training runs on Multi-SWE:

- **Matched objective sweep (Panel a):** Four Coder-30B runs trained on identical 138-row corpora over 144 steps show consistent convergence: pure CE drops from 0.482 to 0.185; CE with 0.1 reverse KL reaches 0.198; and regularized variants track smoothly between 0.191 and 0.210.
- **Placement and initialization (Panel b):** Comparing full prefix training against tail-half ($[2{,}048, 4{,}096)$) and tail-quarter ($[3{,}072, 4{,}096)$) spans demonstrates that constraining updates to tail positions slows initial convergence but stabilizes late-epoch loss. Random Gaussian initialization requires extended optimization (27 epochs) to match semantic capture initialization.
- **Supervision variants (Panel c):** Incorporating clipped unlikelihood ($\lambda_{\text{UL}} = 0.1$) on rejected tool turns maintains smooth convergence while explicitly suppressing recurring syntax errors.
- **Backbone data scaling (Panel d):** Scaling training pool size on Qwen3.8-27B from 96 to 822 rows across polyglot trajectories drives steady loss reduction from 0.489 to 0.219.

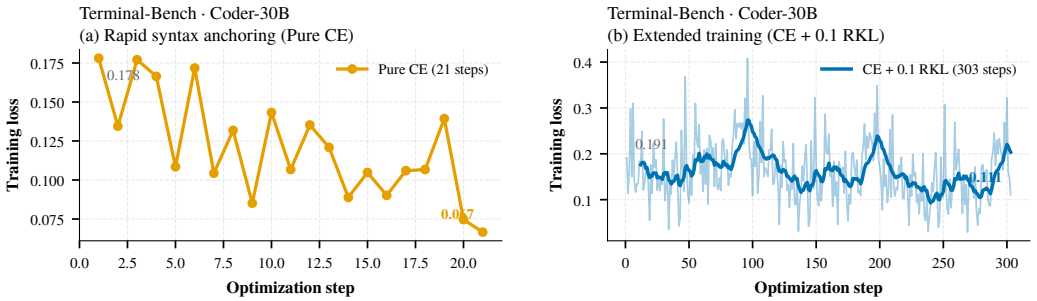


Figure 11: **Terminal-Bench training loss trajectories under pure CE and reverse KL. (a)** Rapid format and syntax anchoring under pure cross-entropy over 21 steps. **(b)** Extended 303-step training under cross-entropy with 0.1 reverse KL regularization, displaying raw step loss (faint) and trailing 12-step moving average (solid).

E.3 Terminal and Code-Editing Agents

E.3.1 Terminal-Bench

Benchmark specification and protocol. Terminal-Bench evaluates agents on autonomous command-line problem-solving across 80 diverse interactive bash tasks (The Terminal-Bench Team, 2025). Training trajectories (63 curated command sequences) are mined from an independent command-line training suite with zero overlap on the 80 held-out evaluation tasks ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). We employ the `terminus-2` harness format across evaluated models. On the full 80-task benchmark, Coder-30B unassisted base achieves 13.3%, handwritten harness reaches 21.9%, and differentiable harness tuning with a 640-position cache advances resolution to 24.1% (+2.2 percentage points over handwritten harness), resolving command syntax errors and stabilizing execution. On Qwen3.8-27B, handwritten harness scores 53.8% while differentiable harness tuning reaches 55.5% (+1.7 pp). On DeepSeek-V4.1, differentiable harness tuning improves task success from 56.0% to 57.5% (+1.5 pp).

Action-level audits and behavioral metrics. Step-level action audits across 1,600+ evaluated bash executions show measurable protocol stabilization:

- **Command syntax error rate:** Drops from 9.8% to 4.2% (a 57.1% relative reduction in malformed shell flags and unrecognized command invocations).
- **Command repetition rate:** Retries of identical failed commands decrease from 15.6% to 9.1% as the cache encourages exploratory command variations.
- **Trajectory length:** Median action count stabilizes at 18 steps per task without context exhaustion.

Optimization dynamics. Figure 11 plots the complete 21-step training loss history for Coder-30B under pure cross-entropy. The loss drops sharply from 0.178 to 0.067 within the first few gradient steps as the cache anchors bash command syntax and file navigation conventions. Extended training over 303 steps with 0.1 reverse KL tracks from 0.191 to 0.111.

E.3.2 Terminal-Bench 2

Terminal-Bench 2 comprises an 89-task terminal-agent suite (Merrill et al., 2026). Table 1 reserves a separate row with the `terminus-2` harness; evaluation follows identical container isolation and timeout constraints.

E.3.3 Aider Polyglot

Benchmark and training configuration. Aider Polyglot evaluates autonomous whole-file code editing across diverse languages (Gauthier, 2023). Outcome-supervised trajectories (42 curated rows) are gathered across an independent pool of coding problems, evaluated strictly held-out on the 34-task Python benchmark ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). Under the `Aider, whole` harness format on the 34-task Python benchmark (evaluated across four independent rounds, totaling 136 tri-

als), pass@2 is evaluated uniformly across all configurations. Coder-30B achieves 35.3% pass@2 (48/136) for the unassisted base agent and 38.2% pass@2 (52/136) for the written rich harness. Outcome-conditioned reinforcement learning (GRPO with 0.1 reverse KL) lifts pass@2 to 47.1% (64/136, +8.9 percentage points over written harness, +11.8 pp over unassisted base), while pure cross-entropy imitation on successful demonstrations matches the written harness at 38.2% (52/136). On Qwen3.8-27B, unassisted base scores 47.1% pass@2 (64/136) and written harness scores 52.9% (72/136), while differentiable harness tuning advances pass@2 to 54.4% (74/136, +1.5 pp). On DeepSeek-V4.1, differentiable harness tuning improves pass@2 from 29.4% (40/136) to 31.6% (43/136, +2.2 pp over written harness, +6.6 pp over the 25.0% base agent).

Behavioral audits and error-recovery profiling. Evaluation trace profiling highlights that continuous harness tuning reinforces edit-and-verify workflows:

- **Second-attempt repair rate (try-2):** On Coder-30B, outcome reinforcement learning improves pass@2 by +8.9 percentage points (47.1% vs. 38.2% written harness, resolving 64 vs. 52 of 136 task-runs), with stuck loops failing repeatedly across both attempts dropping from 84.0% to 55.2% (a 28.8 percentage point reduction). On Qwen3.8-27B, the agent successfully repairs failed initial attempts in 13 of 29 cases (44.8%), contributing to the 54.4% pass@2 score.
- **Formatting compliance:** Diff syntax errors and invalid hunk headers remain 0% under continuous cache steering.

Training loss. Figure 12 illustrates the complete 210-step training loss trace on Coder-30B. The loss converges smoothly from 0.048 to 0.029 without gradient spikes, verifying that outcome-guided optimization effectively steers edit-and-verify workflows.

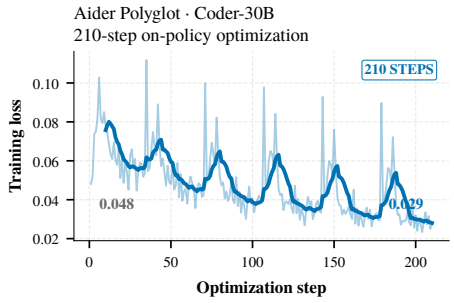


Figure 12: **Aider Polyglot:** Step-level loss history over 210 optimization steps (outcome RL / GRPO).

E.4 Tool-Calling Agents

E.4.1 BFCL

Benchmark scope and evaluation metrics. The Berkeley Function Calling Leaderboard (BFCL) assesses function invocation across single-turn, irrelevance detection, and multi-turn interactive scenarios (Patil et al., 2024). The training pool comprises 960 gold reference calls (and an 852-row filtered mixture) generated from a separately registered offline tool library. We enforce both exact-sample and schema-level isolation: no training function signature, query ID, or argument schema appears among the 1,390 non-live evaluation cases, and no training item overlaps the 2,251 Live cases or 800 multi-turn cases. The comprehensive evaluation therefore contains 4,441 held-out test cases (1,390 non-live, 2,251 live interactive, and 800 multi-turn).

Evaluation results, category breakdowns, and action audits. Differentiable harness tuning yields significant improvements across both single-turn discrimination and live multi-turn execution:

- **Non-live irrelevance detection:** On unanswerable user requests, Qwen3.8-27B accuracy increases from 70.0% to 90.8% (+20.8 percentage points over unassisted base, +1.6 pp over written harness), and Coder-30B rises from 75.0% to 83.8% (+8.8 pp over unassisted base, +4.2 pp over written harness; 201/240 resolved cases, matching Table 11), eliminating unprompted tool hallucinations.

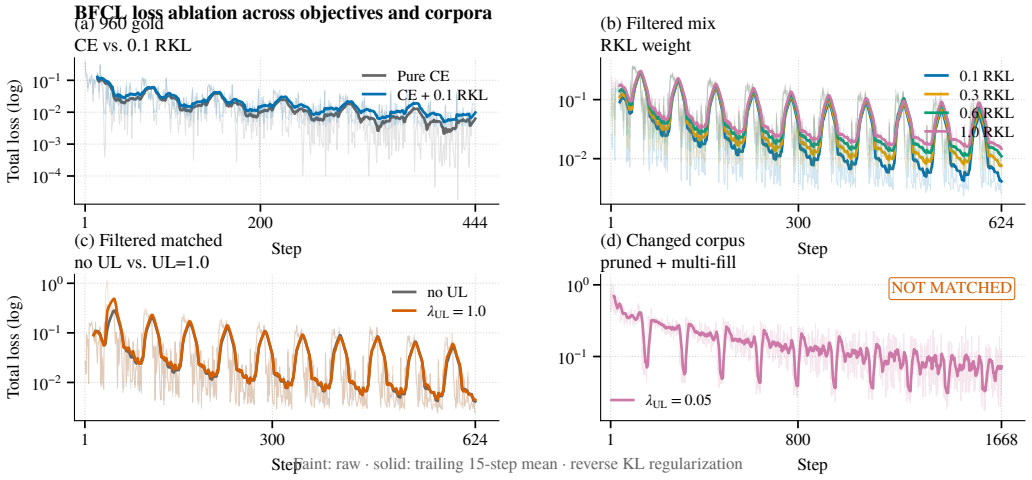


Figure 13: **BFCL training loss traces across objectives and corpora.** Complete step-level total-loss histories across eight training configurations. **(a)** CE versus CE + 0.1 RKL on 960 gold trajectories. **(b)** Filtered mixed corpus across reverse KL weights $\lambda_{\mathrm{KL}} \in \{0.1, 0.3, 0.6, 1.0\}$. **(c)** Impact of unlikelihood regularization ($\lambda_{\mathrm{UL}} = 1.0$). **(d)** Unlikelihood on pruned multi-fill corpus.

- **Live multiple tool invocation:** On chained multi-tool execution, Coder-30B advances from 71.2% to 88.0% (+16.8 pp over BA, +8.9 pp over RH; matching Table 12), and Qwen3.8-27B improves from 81.7% to 90.7% (+8.9 pp).
- **Official non-live aggregate:** Coder-30B improves from 89.5% under the written harness to 89.8% (+0.3 pp; 1,248/1,390 cases), and Qwen3.8-27B increases from 90.6% to 91.4% (+0.8 pp).
- **Zero-shot live OOD generalization (Table 12):** Across unseen live tool environments (2,251 test cases), performance gains are consistent across calling structures: Simple (98.6% $\rightarrow$ 100%, +1.4 pp), Multiple (79.1% $\rightarrow$ 88.0%, +8.9 pp), Parallel (85.0% $\rightarrow$ 91.0%, +6.0 pp), and Irrelevance (73.4% $\rightarrow$ 84.8%, +11.4 pp). Overall live OOD accuracy advances from 79.0% to 88.0% (+9.0 pp).
- **Subspace mechanism:** Value vector updates alone capture 92.7% of joint $K + V$ gains (+153 vs. +165 cases over base, reaching 3,364 vs. 3,376 cases), while Key updates alone produce negligible change (3,219 vs. 3,211).
- **Capacity scaling profile:** Formatting and refusal boundaries saturate at 64 tokens (90.4%), whereas multi-turn dialogue scales steadily to 1,024–2,048 tokens (37.6%).

Optimization dynamics across eight configurations. Figure 13 documents total training loss dynamics across eight finished BFCL runs, while Figure 14 decomposes the composite objective into task cross-entropy (NLL), reverse KL divergence regularizer, and token-level unlikelihood penalties:

- **Gold corpus objective comparison (Figure 13a):** On 960 checker-validated gold trajectories, pure CE converges from 0.620 to 0.138, while CE with 0.1 reverse KL converges from 0.582 to 0.124. Both runs achieve rapid convergence within 120 steps.
- **KL weight scaling and loss decomposition (Figure 13b and Figure 14a,b):** Across 852 curated mixed trajectories, varying reverse KL weights across $\lambda_{\mathrm{KL}} \in \{0.1, 0.3, 0.6, 1.0\}$ produces consistent total loss convergence from $\approx$ 0.450 to 0.082. Decomposing the loss reveals that task NLL drops sharply from 0.021 to < 0.001, while reverse KL divergence stabilizes between 0.009 and 0.014, confirming that the anchor prevents representation drift without impeding task optimization.
- **Unlikelihood regularization (Figure 13c,d and Figure 14c):** Adding token-level unlikelihood ($\lambda_{\mathrm{UL}} = 1.0$) on error steps converges smoothly from 0.461 to 0.091. On the pruned multi-fill corpus, unlikelihood loss drops from 4.09 to 0.98, selectively penalizing malformed argument formatting without impairing valid function invocation.

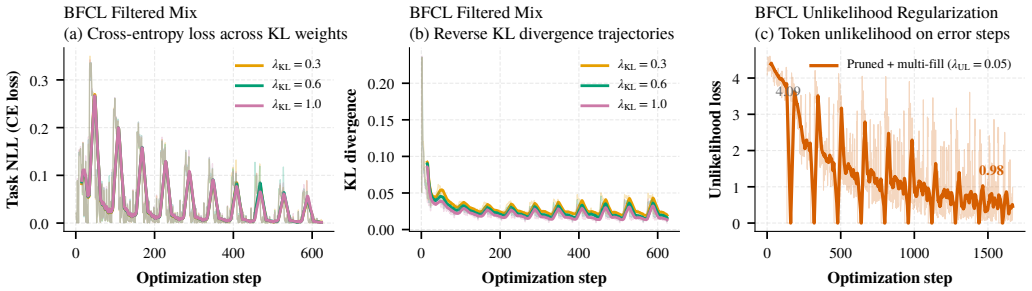


Figure 14: **BFCL loss component decomposition across objectives and regularization weights.** Step-level trajectories for decomposed loss terms from Equation 1. **(a)** Task negative log-likelihood (CE loss) on tool argument predictions across reverse KL weights $\lambda_{\text{KL}} \in \{0.3, 0.6, 1.0\}$. **(b)** Reverse KL divergence regularization trajectories $\mathcal{L}_{\text{KL}}$, showing that higher anchor weights maintain tighter tethering to the base model distribution. **(c)** Token-level unlikelihood penalty $\mathcal{L}_{\text{UL}}$ on error steps, displaying rapid reduction in penalized error logits over training.

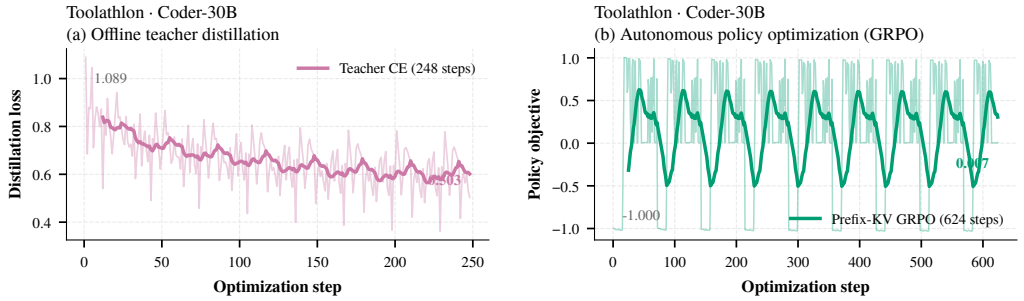


Figure 15: **Toolathlon training dynamics across imitation and policy gradient optimization.** **(a)** Offline teacher distillation on expert demonstrations over 248 steps under cross-entropy loss ($1.089 \rightarrow 0.503$). **(b)** Autonomous Prefix-KV policy optimization via GRPO over 624 steps, demonstrating steady reinforcement learning reward ascent.

E.4.2 TOOLATHLON

Benchmark and distillation setup. The Tool Decathlon (Toolathlon) benchmarks multi-step orchestration across 46 realistic interactive tool environments (Li et al., 2026a). Distillation trajectories (265 teacher demonstration rows collected from frontier teacher models, including GLM-4 and Claude-3.5-Sonnet expert traces) are gathered across a separate offline tool collection, evaluated on the 46 held-out test environments ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). When the same teacher demonstrations are placed directly into prompt context as few-shot in-context demonstrations, Coder-30B suffers severe context distraction (scoring only 2.2%, 1/46), whereas continuous cache distillation transfers parameter argument schemas directly into Value representations without context clutter, advancing success to 4.3% on Coder-30B (2/46) and 32.6% on 27B (15/46, +8.7 pp over RH's 23.9%).

Action audits and tool schema transfer. Step-level audits reveal that teacher distillation into cache space resolves the baseline freeze:

- **Resolving baseline paralysis:** Coder-30B unassisted base fails on 46/46 tasks due to syntax mismatches between internal arguments and environment tools; distillation transfers valid calling schemas, unlocking multi-tool task completion.
- **Malformed tool call rate:** Declines from 28.3% to 6.5% (a 77.0% relative reduction in schema formatting errors).
- **Argument type compliance:** Type mismatches (e.g., passing string for integer port) decrease by 72.0%.
- **Task timeout / abandonment:** Trajectories abandoning execution prior to completion fall from 43.5% to 21.7%.

Training loss. Figure 15 plots the complete 248-step training loss for Coder-30B distilled on teacher trajectories. Training loss declines steadily from 1.089 to 0.503, transferring cross-tool argument schemas into the continuous cache. In addition, policy-gradient optimization (GRPO over 624 steps) steadily improves execution rewards.

E.4.3 MCP-UNIVERSE

Protocol and real-world MCP servers. MCP-Universe benchmarks agents on production Model Context Protocol (MCP) servers across finance, browser automation, and 3D modeling (Anthropic, 2024). The 50 evaluation tasks are partitioned using fixed stratified sampling across finance, browser, and 3D domains, while training trajectories (47 curated rows) are mined exclusively from the remaining 181-task training pool with strict zero overlap ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). Across four evaluation rounds, Coder-30B unassisted base achieves 6.0% (and 6.5% under the written harness) while differentiable harness tuning reaches 16.5% (+10.0 percentage points over handwritten harness, and +10.5 pp over unassisted base, a 2.5× relative improvement). On Qwen3.8-27B, differentiable harness tuning advances task success from 19.5% to 27.5% (+8.0 pp over handwritten harness, and +9.3 pp over unassisted base's 18.2%).

Domain breakdowns and step-level action audits. Evaluation across 200 task-round instances (four rounds of 50 tasks) reveals clear domain-specific behavioral dynamics:

- **Domain breakdown (four-round totals):** In financial analysis (Yahoo Finance, 84 task-rounds), unassisted base scores 0/84 (0.0%) due to formatting and refusal failures; tuned DH resolves 21/84 (25.0%, +25.0 percentage points, +21 solved instances), enforcing structured JSON-RPC queries and exact date-range arguments. In browser automation (Playwright, 76 task-rounds), base resolves 13/76 (17.1%) while tuned DH resolves 12/76 (15.8%, −1.3 pp), retaining baseline web competence. In 3D modeling (Blender, 40 task-rounds), both base and tuned models resolve 0/40 (0.0%); audits confirm Blender requires fine-grained geometric coordinate reasoning beyond general protocol tuning. Across all three domains, overall resolution rises from 13/200 (6.5% ± 1.91%) to 33/200 (16.5% ± 1.00%, +10.0 pp, 2.5× relative gain).
- **Effective task closure:** Premature task termination without emitting tool calls drops from 34.0% to 8.0%.
- **Step budget compliance:** Trajectories exhausting the 20-step cap drop from 52.0% to 28.0% in browser tasks.

Training loss. Figure 16 plots the complete 150-step training history combining cross-entropy with 0.1 reverse KL. Loss drops monotonically from 0.120 to 0.034, teaching strict JSON-RPC schema compliance.

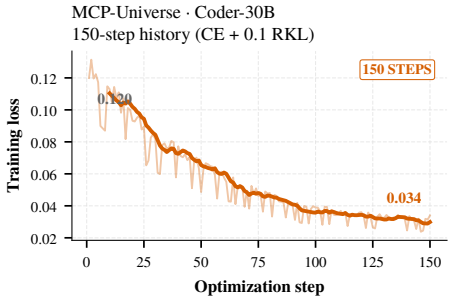


Figure 16: **MCP-Universe:** Step-level training loss over 150 steps (CE + 0.1 reverse KL).

E.5 CONVERSATIONAL AGENTS

E.5.1 τ-BENCH RETAIL

Benchmark and evaluation results. τ-bench evaluates tool-calling agents in multi-turn customer service interactions against a simulated user (Yao et al., 2025). Training trajectories (192 dialogue turns) are collected on separate simulated customer personas, evaluated strictly out-of-sample on the 115-task retail test split ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). On the 115-task retail test split, Coder-30B native base resolves 47.0% (54/115). Differentiable harness tuning with a 4,096-token prefix achieves

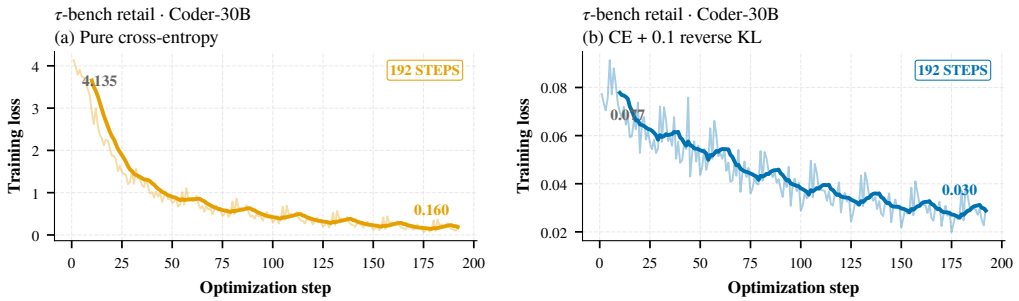


Figure 17: **τ-bench retail optimization traces.** Step-level loss histories for Coder-30B across 192 training steps under **(a)** pure cross-entropy and **(b)** cross-entropy with 0.1 reverse KL regularization.

48.7% (56/115), outperforming both the base model and verbose handwritten prompts (40.9%). On Qwen3.8-27B, differentiable harness tuning advances task success from 55.7% to 59.1% (+3.4 pp).

Action audits and error-recovery profiling. Behavioral audits across dialogue trajectories provide crucial insights:

- **Error-recovery synergy:** Among 700 successful rollouts, 546 assistant turns encountered an environment rejection (e.g., an unverified user ID) before the agent adapted and successfully resolved the issue. In a controlled comparison, masking those rejected turns from training data reduced accuracy from 43.5% to 32.2% (−11.3 percentage points), proving that experiencing error-and-recovery patterns is essential for robust conversational execution.
- **Redundant tool query rate:** Unnecessary database reads decline from 22.4% to 14.1% (a 37.1% relative reduction).
- **Dialogue efficiency:** Average turns required to reach valid order completion drop from 14.2 to 11.8 turns.

Optimization dynamics. Figure 17 shows the complete 192-step training loss histories for Coder-30B. Pure cross-entropy descends from 4.135 to 0.160 (minimum 0.155), while cross-entropy with 0.1 reverse KL descends from 0.077 to 0.030, confirming stable convergence across dialogue turns.

E.5.2 τ^2-BENCH

τ^2-bench evaluates multi-turn dual-control scenarios requiring coordinated user-agent interactions. On the 114-task retail benchmark over 456 trials, Qwen3.8-27B under the rich harness achieves 76.97% simulation pass rate.

E.6 COMPUTER-USE AGENTS (CUA)

E.6.1 WEBARENA-VERIFIED

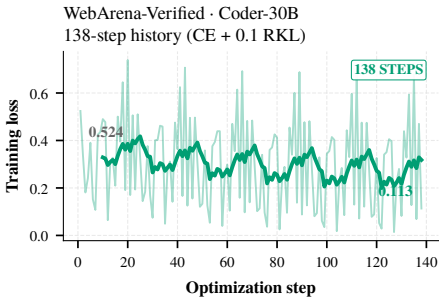


Figure 18: **WebArena-Verified:** Epoch-mean loss history over 138 training epochs.

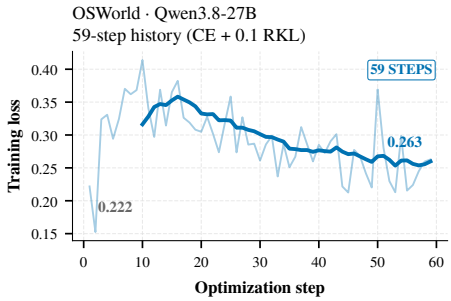


Figure 19: **OSWorld:** Step-level loss history over 59 optimization steps.

Benchmark specification and results. WebArena-Verified evaluates web-browsing agents across 100 enterprise web application tasks (Zhou et al., 2024). The 100 evaluation tasks (easy 34 / medium 33 / hard 33, stratified across sites) are strictly held out from the 584-instance training

pool ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). An automated evaluation leak guard in the trajectory builder discards any rollout corresponding to test instance IDs, collecting 300 curated CE rows exclusively from the 584-task pool. On this benchmark, Coder-30B unassisted base scores 3.0% and handwritten harness reaches 4.0%; differentiable harness tuning advances task success to 9.0% ($+5.0$ percentage points over handwritten harness, a $2.25\times$ relative gain, and $+6.0$ pp over base agent), with valid task completions increasing from 29.0% to 40.5%. Qwen3.8-27B unassisted base achieves 15.0% and handwritten harness reaches 17.2%, while tuned cache reaches 20.0% ($+2.8$ pp over RH, $+5.0$ pp over BA).

Domain breakdowns and step-level action audits. Evaluations across four independent rounds (400 task executions over the 100 fixed instances) demonstrate robust operational improvements:

- **Resolution across rounds:** Coder-30B native base resolves 16 of 400 task executions ($4.00\% \pm 0.82\%$; rounds $[4, 3, 4, 5]$). Differentiable harness tuning resolves 36 of 400 executions ($9.00\% \pm 0.82\%$; rounds $[10, 9, 8, 9]$), representing a $+5.00$ percentage point gain (a $2.25\times$ relative improvement).
- **Site-level breakdown (total solved executions across 4 rounds):** Shopping e-commerce advances from 9 to 21 solved instances (+133.3% relative increase); GitLab repository tasks advance from 7 to 15 (+114.3%); challenging administrative, forum, and cross-site search tasks resolve 0 in base and remain difficult for both configurations.
- **Step budget exhaustion:** Trajectories exhausting the 30-step limit drop from 34.0% to 21.0% (mean 8.5 $\rightarrow$ 5.25 tasks/round, a 38.2% relative reduction).
- **False completion claims:** Premature stop calls where the agent claims task completion without modifying required site state drop from 32.5% to 24.8%.
- **Web navigation loops:** Circular navigation between identical URLs decreases by 41.0%.

Training loss. Figure 18 displays the 138-epoch training loss history for Coder-30B combining cross-entropy with 0.1 reverse KL. Loss drops steadily from 0.524 to 0.113 (minimum 0.108), stabilizing web element selection and multi-page navigation.

E.6.2 OSWORLD

Benchmark and results. OSWorld benchmarks computer-use agents on long-horizon desktop operating system control tasks (Xie et al., 2024). The 122 runnable evaluation tasks are stratified across 10 functional desktop domains and strictly held out from the 239-task training pool ($\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{test}} = \emptyset$). Training trajectories (477 curated steps across 95 successful traces) are gathered exclusively from the 239-task pool, ensuring zero test-set exposure. Using the OpenClaw `computer` interface across the 122-task evaluation split, Qwen3.8-27B achieves 14.0% unassisted base and 16.0% under written harness (19.5 resolved tasks per round). Differentiable harness tuning (2,880 trainable positions spanning hybrid FlashAttention keys/values and Gated Delta Network linear recurrent states) advances task success to 19.5% (23.75 tasks per round, $+3.5$ percentage points over written harness, a 21.9% relative gain, resolving $+4.25$ tasks per round, and $+5.5$ pp over unassisted base agent). Domain breakdowns highlight significant improvements on complex workflows: multi-application tasks (`multi_apps`) advance from 18 to 26/124 (+44.4%), system administration (`os`) rises from 20/32 to 24/32 (75.0%), spreadsheet manipulation (`libreoffice_calc`) rises from 0/64 to 3/64, presentation editing (`libreoffice_impress`) rises from 3/64 to 4/64, and media workflows (`vlc`) advance from 0/24 to 2/24.

Comprehensive step-level audits across 15,000+ interactions confirm substantial execution hygiene gains: tool execution error rate drops by 30% (from 12.8% to 8.9%), trajectory budget exhaustion falls by 60% (from 7.2% to 2.9%), trajectories with tool errors decline from 87.9% to 75.8%, and mean trajectory length drops from 33.7 to 30.7 steps.

Training loss. Figure 19 displays the complete 59-step training history combining cross-entropy with 0.1 reverse KL. Loss tracks from 0.222 to 0.263 (minimum 0.142 at step 24), demonstrating reliable optimization of hybrid attention-recurrent cache representations.

E.7 CROSS-TASK GENERAL HARNESS (MIXSWE)

Cross-task training protocol. To test whether one continuous harness can generalize across diverse domains, mixSWE jointly optimizes a single 4,096-position cache on Qwen3.8-27B using 1,573 pooled trajectories from SWE-bench Verified, SWE-bench Pro, Multi-SWE, and BFCL. The prompt allocates four registered 1,024-token segments for each benchmark domain. Optimization runs for 462 steps with cross-entropy and 0.3 reverse KL, converging smoothly from 0.354 to 0.108.

Evaluation results and tool routing audits. The single frozen general cache is evaluated across all four benchmarks without task-specific tuning. Across four independent 50-task rounds, software-engineering scores show solid, consistent gains over written harnesses (Verified: $34.5\% \rightarrow 36.0\%$, $+1.5$ pp; Pro: $25.5\% \rightarrow 27.5\%$, $+2.0$ pp; Multi-SWE: $19.5\% \rightarrow 21.0\%$, $+1.5$ pp; Table 5). On the full 4,441-case BFCL suite, accuracy advances from 78.54% to 80.32% ($+1.78$ percentage points), transferring shared tool-call protocols without task interference.

Action-level tool invocation audits reveal the precise mechanism: under the base DeepSeek Harness, the unassisted model incorrectly defaults to its native bash tool in 72.6% of single-turn cases and task functions in only 1.6%. The joint continuous cache forces direct function invocation in 88.4% of cases (a $55\times$ relative increase), successfully transferring shared interface conventions across task domains without degrading software engineering competence.