OCTOPUS: ONE BRAIN, MANY BRANCHES FOR LATENT-KV MULTI-AGENT REINFORCEMENT LEARNING

TECHNICAL REPORT

Tianming Sha
Stony Brook University
tianming.sha@stonybrook.edu

Junnan Li
University of Wisconsin–Madison
jli2786@wisc.edu

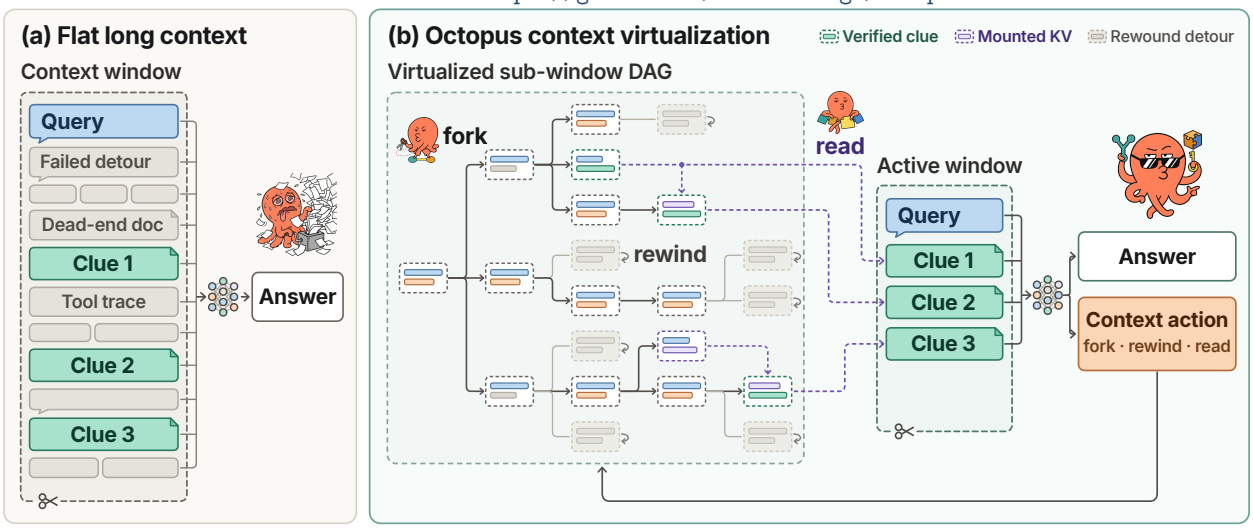
Siqi Zhao
University of Minnesota
zhao2052@umn.edu

ABSTRACT

Deep search and long-horizon reasoning naturally unfold as branching trajectories that pursue parallel sub-queries, abandon dead-end detours, and cross-check sibling findings. Yet modern serving engines, agent harnesses, and reinforcement learning (RL) trainers execute every episode as a flat, append-only text sequence. Stretching a single context window leaves failed detours in working memory that dilute attention, whereas in-place context edits break prefix KV caching and invalidate recurrent states in hybrid architectures. Conversely, coordinating separate chat sessions duplicates shared prefixes, while splicing caches across all model layers overwrites recurrent working memory and triggers severe task hijacking.

We introduce **Octopus**, a co-designed serving runtime, GPU memory manager, and RL framework that executes episodes as a GPU-resident DAG over immutable radix cursors (89.6% GPU KV cache hit rate). Agents coordinate via five primitives: `fork` shares parent KV prefixes in $\mathcal{O}(1)$ with isolated 32K branch budgets, `rewind` restores hybrid recurrent checkpoints, `read_thread` mounts sibling KV pages in zero copy, and `wait` and `done` synchronize results. To prevent cross-branch interference in hybrid models, Octopus splices sibling KV pages exclusively into full-attention layers under monotonic position offsets while recurrent layers skip mounted spans, retaining 92.2% answer information and cutting hijacking 21× (57.9% → 2.7%) with zero new KV pages. During RL, our 1D packed DAG kernel and **DAG-GRPO** objective deduplicate shared prefixes via single-ownership masking while keeping mounted KV edges live, cutting rollout–training KL divergence 32× and routing reader gradients directly into producer KV projections. Trained on Qwen3.5-4B with pure binary outcome rewards, Octopus self-organizes a commander–worker hierarchy and peer-to-peer latent KV mesh from scratch, doubling gold-document recall and lifting hard multi-hop accuracy 10× (dropping 21.7 points if `fork` is disabled at test time). Under sparse BM25, Octopus lifts BrowseComp-Plus accuracy from 26.0% (32K ReAct) and 32.1% (143K Single-Agent GRPO) to 49.6% (+17.5 points) and reaches 51.2% on OpenResearcher. Despite its 4B footprint, Octopus surpasses 9B context-editing RL (42.5%), matches snippet-only OpenAI o3 under BM25 (49.3%), and outperforms dense-retrieval browser agents up to 250× larger, including 30B Tongyi-DeepResearch (44.5%), 671B DeepSeek-R1 (16.4%), 1T Kimi-K2 (35.4%), GPT-4.1 (36.4%), and Claude 4 Opus (36.8%).

Code & Runtime: https://github.com/shatianming5/octopus



1 Introduction

Deep search and multi-hop reasoning are inherently branching processes: an agent decomposes a question into parallel sub-queries, abandons dead-end detours, cross-checks sibling findings, and synthesizes verified evidence [1–3]. Yet modern serving engines, agent harnesses, and reinforcement learning (RL) trainers execute every episode as a flat, append-only text sequence [4–8]. Forcing tree-structured exploration onto a single linear context creates an immediate trilemma. A standard 32K window saturates after opening just a few 8K-token web pages, capping a 4B agent at 26.0% accuracy on BrowseComp-Plus (Table 3). Stretching that flat window to 143K under RL lifts accuracy to only 32.1% and stalls at 28.5% capacity (40.8K tokens), because failed detours remain in working memory and dilute attention over true evidence [9, 10]. Meanwhile, compacting or editing prior turns in place [11–15] discards exact identifiers, breaks append-only prefix KV caching, and invalidates sequential recurrent states in modern hybrid architectures such as Gated DeltaNet [16] and Mamba-2 [17].

Breaking out of a single flat window requires native branching execution, yet existing multi-agent, KV-sharing, and RL systems hit three structural walls. First, client-side multi-agent harnesses coordinate workers over stateless chat APIs, re-transmitting text histories and duplicating shared prefixes across sessions [5, 6, 18, 19]. Second, inference-time KV-sharing methods splice or re-rotate caches across all layers indiscriminately [20–23]; in hybrid recurrent-attention models, stepping recurrent states through a sibling's text or full-layer cache overwrites the reader's sequential working memory and hijacks over half of subsequent turns (55.2%–57.9%; Table 5). Third, existing branching RL frameworks exchange findings only as re-encoded text summaries [24–26], while naive sequence-level tree training either overcounts shared prefixes or severs cross-branch KV dependencies.

We introduce **Octopus** (Figure 1), a co-designed serving runtime, GPU memory manager, and RL framework that replaces the flat text tape with a GPU-resident execution DAG. Agents coordinate directly over immutable server-side radix cursors through five primitives: `fork` shares the parent's KV prefix in $\mathcal{O}(1)$ while granting each child a clean 32K growth budget; `rewind` rolls back dead-end detours and restores hybrid recurrent checkpoints $\mathbf{h}_k$; `read_thread` mounts sibling KV pages in zero copy; and `wait` and `done` synchronize results. Two co-designed mechanisms make latent cross-branch execution both hijack-free and end-to-end trainable. First, Octopus decouples sequential working memory from associative recall in hybrid models [27]: `read_thread` splices sibling KV pages exclusively into full-attention layers under a monotonic position offset, while recurrent layers skip mounted spans. This zero-copy mount allocates zero new KV pages, retains 92.2% of full-text answer information, and cuts off-task hijacking by 21× (57.9% → 2.7%). Second, our 1D packed DAG kernel and **DAG-GRPO** objective deduplicate shared prefixes via single-ownership masking while keeping mounted KV edges live in the forward–backward graph. Retaining live mount edges cuts rollout–training divergence by 32× ($\mathrm{KL} = 0.0142 \rightarrow 4.45 \times 10^{-4}$) and routes reader gradients directly into producer KV projections, keeping cross-branch latent communication strictly on-policy and end-to-end differentiable.

Trained on Qwen3.5-4B using pure binary outcome rewards with zero process shaping [28], Octopus discovers a self-organized division of labor from scratch (Figure 10). Without RL, exposing branching primitives yields no gain over 32K single-agent ReAct (26.3% vs. 26.0%). Under DAG-GRPO, the policy undergoes a sharp phase transition at Step 15: the root thread evolves into a dedicated commander that forks on Turn 1 in 98.0% of episodes (up 38×), offloads 92.8% of 8K-page reads and 91.5% of rewinds to sub-agents, and coordinates a peer-to-peer latent KV mesh averaging 23.9 zero-copy mounts per episode. Disabling `fork` at test time on the exact same trained weights collapses paired BrowseComp-Plus accuracy by 21.7 points (50.8% → 29.1%), proving that gains come from live collective execution rather than weight memorization.

Isolating noisy document reads across clean 32K branches keeps per-turn active prompts at 26.4K tokens while scaling explored context to 329.2K (199.0K net surviving), compressing critical-path length by 1.94×, more than doubling gold-document opening recall (21.1% → 43.3%), and lifting accuracy on the hardest multi-hop queries by 10× (3.4% → 34.1%). Overall, Octopus-4B under sparse BM25 reaches 51.2% on OpenResearcher and 49.6% on BrowseComp-Plus (+17.5 points over 143K Single-Agent GRPO; Figure 8, Table 3). Despite its 4B footprint, Octopus surpasses 9B context-editing RL (42.5%) [15], matches snippet-only OpenAI o3 under BM25 (49.3%), and outperforms dense-retrieval browser agents up to 250× larger—including 30B Tongyi-DeepResearch (44.5%), 671B DeepSeek-R1 (16.4%), 1T Kimi-K2 (35.4%), GPT-4.1 (36.4%), and Claude 4 Opus (36.8%). Our contributions are:

1. **GPU-Native Branching Execution Primitives (Sections 3.1 and 4.3):** We formulate long-horizon exploration as a GPU-resident execution DAG over five primitives (`fork`, `rewind`, `read_thread`, `wait`, and `done`) with isolated 32K branch budgets and bounded runtime admission.
2. **Hijack-Free Zero-Copy KV Mounting (Sections 3.2 and 4.2):** We splice sibling KV pages exclusively into full-attention layers under monotonic position offsets while recurrent layers skip mounted spans, retaining 92.2% answer information and cutting hijacking 21× (57.9% → 2.7%).

3. **Server-Resident Radix Cursors and 1D Packed DAG Kernels (Sections 3.3 and 4.2):** We bind turns to immutable GPU radix cursors (89.6% KV cache hit rate) and compile prefix trees, recurrent checkpoints, and KV mounts into one 1D forward–backward pass.
4. **Latent-Communication Tree RL and Emergent Collective Scaling (Sections 3.4, 4.1 and 4.3):** We introduce DAG-GRPO with single-ownership masking and live mount edges (32× lower rollout–training KL), eliciting emergent multi-agent division of labor and lifting Qwen3.5-4B to 49.6% on BrowseComp-Plus and 51.2% on OpenResearcher.

2 Related Work

Active Context Management, Compaction, and Branch-and-Return Folding. To operate within finite context budgets, recent agent harnesses periodically summarize history [11, 12], consolidate memory every turn [13, 29], or expose explicit compression and offloading tools [14, 30–33]. Context Language Models (CLM) paired with Suffix Cache Reuse (SCR) [15] allow in-place context edits, but mid-context mutations break prefix caching, force suffix copy and RoPE re-rotation, and invalidate sequential recurrent states in hybrid models. Closer to our setting, Context-Folding [25] folds sub-trajectories sequentially at depth 1, while Adaptive Parallel Reasoning (APR) [24] spawns parallel child threads under RL. However, both return compressed text summaries upon joining a branch—discarding exact identifiers and preventing lateral peer reads—and neither snapshots or rolls back hybrid recurrent states (Table 1). Octopus instead executes concurrent, unlimited-depth trees over shared GPU radix prefixes, supports exact recurrent-state backtracking, and shares sibling findings directly via zero-copy attention KV mounting.

Multi-Agent Harnesses and Cross-Agent KV Sharing. Existing multi-agent harnesses [5, 6, 18, 19] coordinate workers above stateless chat APIs, incurring repeated prompt re-transmission, re-tokenization, and quadratic KV duplication across sessions. To bypass text serialization, inference-time KV-sharing methods select attention layers in pure Transformers [34] or splice and RoPE-rerotate KV blocks across contexts [20–23, 35–37], while Cache-to-Cache [38] and DiffMAS [39] train supervised KV fusion modules on fixed graphs. Yet none of these methods support dynamic branching trees under on-policy RL, prevent recurrent-state corruption in hybrid architectures [16, 17, 27], or avoid RoPE phase inversion when a sibling branch runs longer than the reader. Octopus resolves all three limitations by splicing sibling KV pages exclusively into full-attention layers under rotation-free monotonic position offsets while recurrent layers skip mounted spans.

Reinforcement Learning for Long-Horizon and Branching Agents. Multi-turn agentic RL [1, 7, 8, 40–42] and stepwise context-editing RL [12, 15, 31] optimize flat trajectories segmented at turn or edit boundaries. Existing branching and hierarchical RL methods [24–26] exchange findings across branches only as re-encoded text summaries, whereas flattening trees into disconnected sequences either overcounts shared prefixes or drops cross-branch KV mounts. Octopus instead compiles prefix trees, recurrent-state rewinds, and live KV mount edges into one on-policy training graph ($\mathrm{KL} = 4.45 \times 10^{-4}$ vs. 0.0142 without mounts; Section 4.3), routing reader gradients directly into producer KV projections. To our knowledge, Octopus is the first on-policy RL system for agents that communicate through shared latent KV states, and the first on hybrid recurrent-attention models.

3 Method

Instead of re-sending growing chat transcripts on every turn, Octopus models a long-horizon episode as a cursor-addressed execution DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E}_{\text{tree}}, \mathcal{E}_{\text{mount}})$. Analogous to an OS copy-on-write handle, each vertex $c \in \mathcal{V}$ is a lightweight cursor ID referencing an immutable snapshot of physical KV pages and recurrent state resident in GPU memory. Directed tree edges $(c \to c') \in \mathcal{E}_{\text{tree}}$ capture vertical state transitions (turn continuation, parallel `fork`, and `rewind`), whereas directed mount edges $(c_A \rightsquigarrow c_B) \in \mathcal{E}_{\text{mount}}$ capture horizontal coordination (`read_thread`), allowing branch B to attend directly to sibling A's physical KV pages. Octopus co-designs four layers around $\mathcal{G}$ (Figure 1): branching execution primitives (Section 3.1), layer-selective monotonic KV mounting (Section 3.2), server-resident radix cursors and 1D packed DAG kernels (Section 3.3), and on-policy DAG-GRPO training (Section 3.4).

3.1 Branching Execution Primitives and Per-Branch Isolation

Existing single- and multi-agent harnesses sit above the serving engine, either enforcing a flat append-only loop or stitching stateless chat sessions at $\mathcal{O}(L)$ wire cost and $\mathcal{O}(N^2)$ KV duplication. Octopus instead co-designs the multi-agent harness with the GPU memory manager, running branches as lightweight GPU-resident threads across five primitives with isolated budgets and explicit closing contracts (Figure 2, Table 2).

Table 1: Six-dimension architectural comparison between **Octopus** and five existing long-horizon agent paradigms (✓ supported, △ partial / degraded, ✗ unsupported / vulnerable).

System, Memory & RL Dimension	1. Long Window (Qwen-1M / Single-Agent RL)	2. Edit / Fold (CLM+SCR / Context-Folding)	3. Multi-Agent (AutoGen / CAMEL / APR)	4. Prior KV Sharing (Blend / Hogwild! / KVComm)	5. Sequence Agent RL (Search-R1 / RAGEN + Leaf-Split Trees)	Octopus (Ours) (Cursor + Attn Mount + DAG-GRPO)
1. Execution Topology & Isolation	✗ **Flat Linear** Detour pollution dilutes attention	△ **Sequential Fold** Depth-1 serial; no path speedup	✓ **Parallel Chats** Isolated sessions, no shared KV	✓ **Parallel Tree** Inference-only KV block sharing	✗ **Flat Sequences** Single-chain rollout or leaf-split tree	✓ **Parallel CoW Tree + DAG** Unlimited depth (32K/branch) + lateral peer `read`
2. Per-Turn Wire & Token Protocol	✗ $\mathcal{O}(L)$ **Resend** Full history resend & template drift	✗ **Re-Tokenize** Re-renders prompt after mid-edits	✗ $\mathcal{O}(NL)$ **Resend** Repeated text serialization	△ **Block Sync** External position & block tracker	✗ $\mathcal{O}(L)$ **Stitch** Re-tokenizes flat turns every step	✓ $\mathcal{O}(\Delta)$ **Server Cursor** Immutable radix cursor + startup template probe
3. Prefix KV & Hybrid State Reuse	△ **Append-Only** Invalidated upon any backtrack	✗ **Breaks Recurrence** Mid-edit / fold invalidates state	✗ $\mathcal{O}(N^2)$ **Isolated** 0% cross-agent KV cache reuse	✗ **Attn-Only / All-Layer** Corrupts hybrid recurrent state	✗ **Recomputes** $K\times$ Repeats shared trunk per leaf	✓ **Shared GPU Radix Tree** $\mathcal{O}(1)$ ref-counted KV prefix + cloned recurrent state
4. Cross-Branch KV Sharing & Hijack Safety	— Single-thread execution only	✗ **Lossy Text Return** Drops IDs; text pollutes memory	✗ $+L$ **Text Copy** Raw transcript hijacks reader	✗ $+L$ **Copy / Unshifted** RoPE copy pages or longer-sibling flip	— No cross-branch state reading	✓ 0**-Copy Attn KV Mount** Recurrent layers skip mount + monotonic pos. offset
5. Backtrack Recovery & Tree Closure	✗ **No Backtrack** Dead-end docs remain in window	△ **KV Truncate Only** No recurrent snap or lateral `read`	✗ **No State Rewind** Cannot roll back branch KV or state	✗ **No State Snap** Cannot roll back recurrent state	✗ **No Branch Rewind** Append-only turn rollout buffer	✓ **Exact State Checkpoint** Reclaims branch budget + verified close contract
6. RL Kernel & Rollout Alignment	✗ **Detour Noise** Failed detours dilute credit	△ **Cut at Segments** No cross-branch KV mount edges	✗ **No Shared-KV RL** Independent chats or text-summary joins	✗ **Inference-Only** No packed tree training kernel	✗ **Drops KV Mounts** Sequence pass breaks rollout alignment	✓ **1D Packed DAG Kernel** Single-ownership loss mask + live on-policy mounts

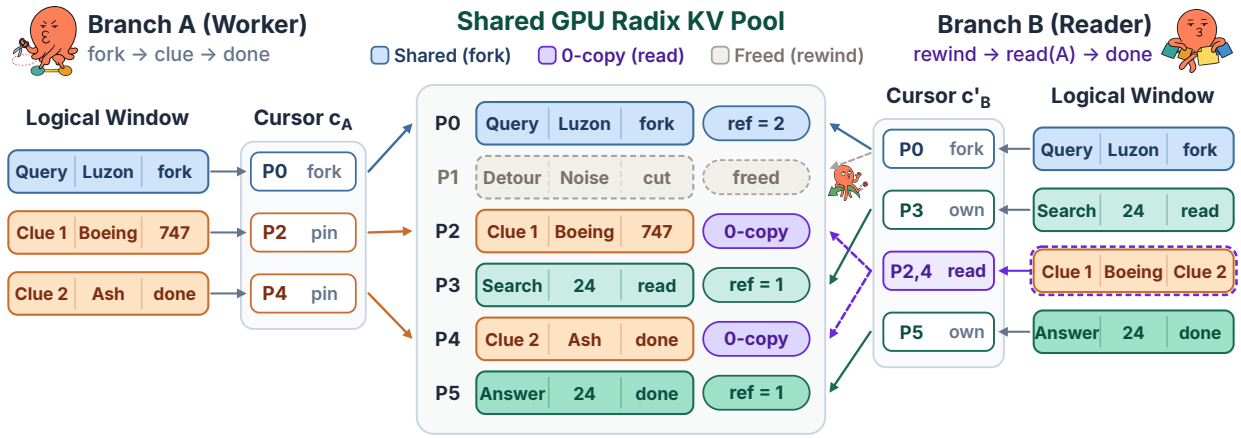


Figure 2: Cursor-addressed page translation and zero-copy KV page lending in Octopus (Sections 3.1 and 3.3). Worker and reader branches maintain isolated logical windows backed by a shared GPU radix KV pool: `fork` shares the parent prefix in $\mathcal{O}(1)$ (`ref` = 2), `rewind` reclaims dead-end pages, and `read` mounts pinned sibling KV pages in zero copy without allocating new physical blocks.

Orchestrator–Worker Coordination. An episode begins at a root orchestrator thread and expands recursively into concurrent worker branches. Calling `fork` at cursor c_{fork} spawns K parallel children in $\mathcal{O}(1)$ zero copy: all children share the parent's GPU KV prefix pages via reference counting ($+K$) while each receives a cloned recurrent state $\mathbf{h}_{\text{end}}(c_{\text{fork}})$. When a search query or 8K-token page open hits a dead end, `rewind` rolls the cursor back to step k, restores the exact recurrent checkpoint $\mathbf{h}_k$, reclaims steps $k+1 \dots t$ from the branch budget, and appends a short diagnostic `note`. For horizontal sharing, `read_thread` (abbreviated `read` in Table 2) mounts a completed step range $[a, b]$ from a sibling or cousin branch directly into the caller's full-attention layers without copying text or advancing recurrent states. Parents synchronize via `wait`, and every thread closes via `done`.

Per-Branch Budgets and Runtime Guardrails. Each thread operates under an independent growth budget $B_{\text{thread}} = 32{,}768$ tokens (with $R_{\text{ans}} = 2{,}048$ tokens reserved for final synthesis): only newly appended or generated tokens consume B_{thread}, whereas inherited parent prefixes and mounted sibling KV pages are shared in zero copy. To preserve prefix KV caching and prevent server saturation or unclosed rollouts, Octopus injects the `read_thread` tool card append-only once a peer step becomes readable, gates new episode admission when GPU KV pool usage exceeds $\tau_{\text{KV}} = 0.75$, and issues a single-turn untrainable reminder if a thread emits plain text instead of `done` (Appendix A).

Table 2: Summary of the five execution primitives in Octopus (Section 3.1), their server-side cursor and KV-cache semantics, and their control-flow and budget semantics.

Primitive	Signature	Cursor & KV-Cache Semantics	Control-Flow & Budget Semantics
`fork`	`fork(branches={id: prompt})`	Clones parent cursor node in $\mathcal{O}(1)$; shares radix KV prefix; clones recurrent state $\mathbf{h}_{\text{end}}(c_{\text{fork}})$.	Spawns parallel child branches, each granted a fresh $B_{\text{thread}} = 32\text{K}$ token growth budget.
`read`	`read(branch, steps=[a, b])`	Zero-copy lends branch A's KV pages to attention layers $\mathcal{L}_{\text{mount}}$ w/ monotonic pos. (Equation (2)); recurrent layers skip span.	Injects a compact metadata stub (~ 15 tokens) into reader B's prompt without copying A's text.
`wait`	`wait(branches=[...])`	Suspends caller cursor until target child branches reach terminal `COMPLETED` state.	Returns brief branch status summaries and step manifests available for `read`.
`rewind`	`rewind(step=k, note)`	Restores cursor c_k and exact recurrent state snapshot $\mathbf{h}_k$ at step k; appends diagnostic `note`.	Reclaims dead-end token budget $(k+1 \dots t)$ while retaining the pruned branch in the training DAG.
`done`	`done(answer, evidence)`	Marks branch or root cursor `COMPLETED` and pins its KV pages in GPU radix pool for parent/sibling `read`.	Enforced by a 2K answer reserve and 1-turn close reminder; returns verified answer and evidence.

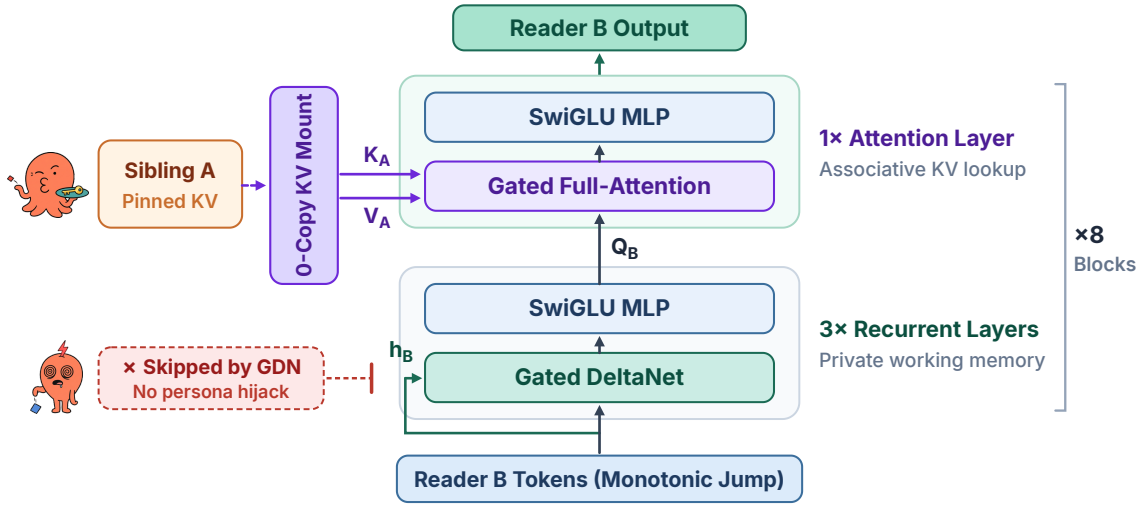


Figure 3: Layer-selective cross-branch KV mounting in hybrid recurrent-attention blocks (Section 3.2). Pinned sibling KV pages (K_A, V_A) are spliced in zero copy exclusively into full-attention layers for associative lookup, while recurrent Gated DeltaNet layers advance only the reader's private working memory $\mathbf{h}_B$ over post-mount tokens to prevent persona hijacking.

3.2 Layer-Selective Monotonic KV Mounting

Sharing sibling findings via raw text injection or all-layer KV splicing causes reader branch B to abandon its own goal and pursue sibling A's task in over half of interactions (Section 4.2). We resolve this conflict by exploiting how recurrent and full-attention layers physically store history (Figure 3).

Decoupling Recurrent Working Memory from Associative Attention Lookup. Modern long-context backbones interleave linear recurrent layers (e.g., Gated DeltaNet or Mamba-2) with sparse full-attention layers [16, 17]. Crucially, a recurrent layer compresses history into a fixed-size rolling state $\mathbf{h}_B$ that is mutated by every incoming token, whereas a full-attention layer stores KV pages passively and reads them only when queried [27] (in pure-attention models, a sparse subset of middle retrieval layers plays an analogous role; Appendix C). Feeding sibling A's trajectory through recurrent layers directly overwrites $\mathbf{h}_B$—replacing B's active persona and goal with A's—whereas mounting A's KV pages into target full-attention layers $\mathcal{L}_{\text{mount}} \subseteq \mathcal{L}_{\text{attn}}$ exposes A's evidence as a passive lookup table. When reader B (at cursor c_B) mounts span $[s_A, e_A)$ from c_A, Octopus splices $c_A[s_A : e_A)$ exclusively into $\mathcal{L}_{\text{mount}}$ and skips $[s_A, e_A)$ in

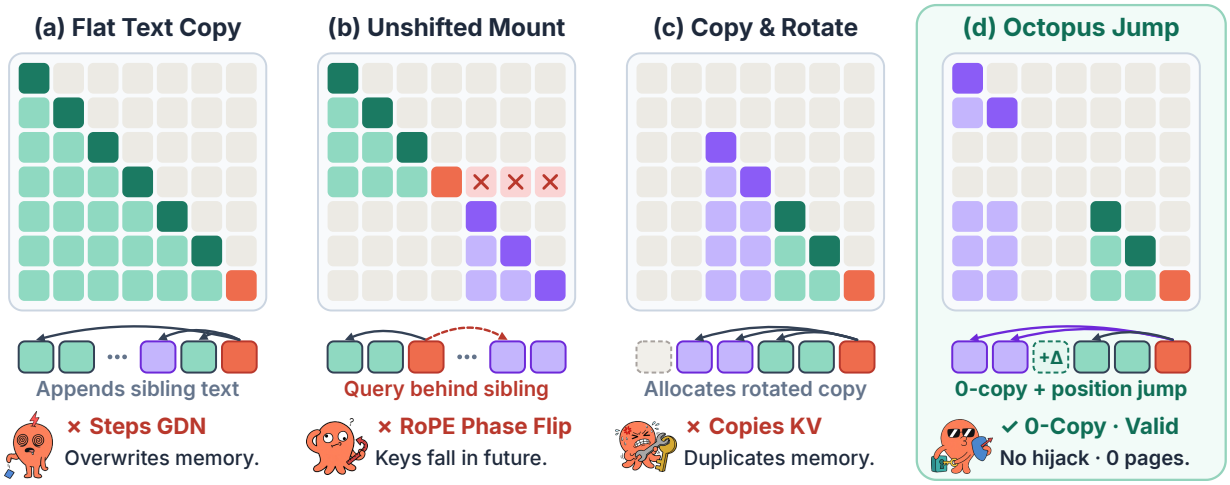


Figure 4: Comparison of cross-branch context sharing regimes (Sections 3.2 and 3.3): (a) flat text copying steps recurrent layers and causes task hijacking; (b) unshifted KV mounting inverts RoPE relative phases whenever sibling keys exceed the reader's position; (c) copying and re-rotating KV pages duplicates GPU memory per reader; (d) Octopus retains native zero-copy KV pages and jumps post-mount query positions monotonically under causal gathering.

all recurrent layers, advancing $\mathbf{h}_B$ solely over B's post-mount tokens Δ_B from $\mathbf{h}_{\text{end}}(c_B)$. Retrieved attention features on Δ_B still inform deeper recurrent layers ($l > l_{\text{attn}}$) via residuals without A's raw tokens overwriting $\mathbf{h}_B$.

Rotation-Free Monotonic Position Continuation. Splicing foreign KV pages into full-attention layers creates relative-position conflicts under 1D/M-RoPE (Figure 4), where attention scores depend on query–key phase differences:

$$\langle \mathcal{R}_q \mathbf{q}, \mathcal{R}_k \mathbf{k} \rangle = \mathrm{Re}\left(\sum_d q_d k_d^* \, e^{i(q-k)\theta_d} \right). \tag{1}$$

Let $p_B = \mathrm{pos}_{\text{next}}(c_B)$ denote B's next unshifted position. Whenever sibling A has progressed further in token index than B ($e_A > p_B$), leaving B's queries at $q_B = p_B$ yields negative relative distances $q_B - k_A < 0$ for mounted keys $k_A \in (p_B, e_A)$, placing A's keys in B's causal future with inverted RoPE phases. Conversely, copying and RoPE-rerotating A's keys into B's local frame [21, 22, 34] duplicates GPU memory per reader. Octopus instead keeps mounted KV pages at their native positions and jumps B's post-mount tokens Δ_B monotonically past the highest mounted index:

$$s(c'_B) = \max\Big(\mathrm{pos}_{\text{next}}(c_B), \max_{m \in \mathrm{Mounts}(c'_B)} \big(\mathrm{pos}_{\text{end}}(m) + 1 \big) \Big). \tag{2}$$

For example, if B is at position 2,000 and mounts A's tokens 1,500–4,000, B's next token jumps to 4,001: every visible key now lies strictly in B's past ($q_B - k \geq 1$) with zero KV copying, while position-free recurrent and convolution layers are unaffected. Although mounted indices 1,500–1,999 overlap with B's pre-mount tokens counted from the fork point, content-based attention keys disambiguate colliding positions without re-rotation (Section 4.2).

3.3 Server-Resident Radix Cursors and 1D Packed DAG Kernels

Inference Engine: Server-Resident Radix Cursors. To eliminate $\mathcal{O}(L)$ prompt re-transmission across turns and forks, the serving engine stores each completed turn as an immutable cursor c backed by a GPU radix prefix tree of reference-counted KV pages. Any continuation, `fork` child, or `rewind` recovery passes cursor ID c and incremental messages Δm, extending the sequence in $\mathcal{O}(|\Delta m|)$ time and wire bytes:

$$\mathbf{x}(c') = \underbrace{\big(\mathbf{x}_{\text{prompt}}(c) \oplus \mathbf{x}_{\text{out}}^{\setminus \text{stop}}(c) \oplus \tau_{\text{term}}(c) \big)}_{\text{Immutable Server-Resident Cursor } \mathbf{x}(c)} \oplus \mathrm{Render}_\Delta\big(\mathcal{T},\ \Delta m,\ \mathrm{ctx}(c) \big), \tag{3}$$

where $\mathbf{x}_{\text{out}}^{\setminus \text{stop}}(c)$ strips trailing stop tokens and $\tau_{\text{term}}(c)$ terminates the assistant turn. Because modern reasoning chat templates $\mathcal{T}$ are often non-append-only—for instance, stripping earlier `<think>` blocks on new user turns—a one-time (~ 0.1 s) startup probe extracts an incremental template slicer Render_Δ and encodes runtime notes as tool responses, achieving bit-exact tokenization ($0/18{,}344$ mismatches across seven model families; Appendix A). Mounted KV spans are pinned via per-page reference counts and transparently re-prefilled from cursor metadata if evicted.

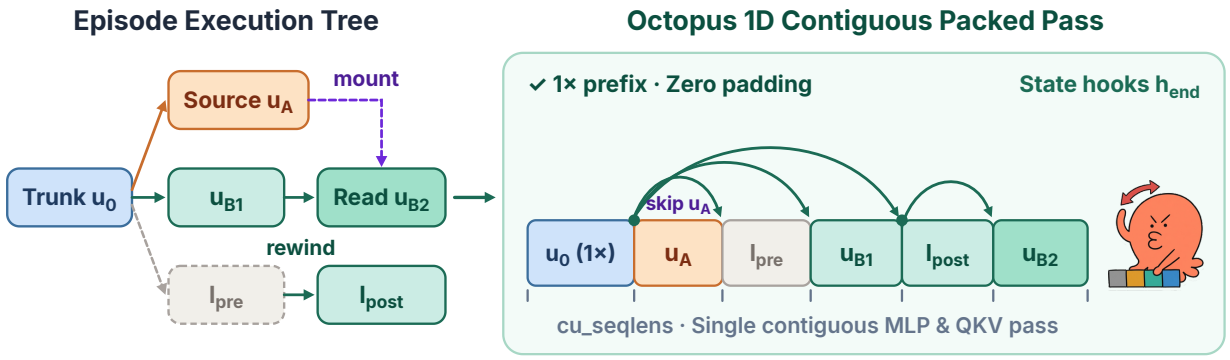


Figure 5: Tree-to-1D contiguous sequence packing (Section 3.3). Octopus deduplicates each tree segment once into an unpadded 1D strip and passes recurrent states $\mathbf{h}_{\text{end}}(p(u))$ across segment boundaries while skipping mounted sibling spans.

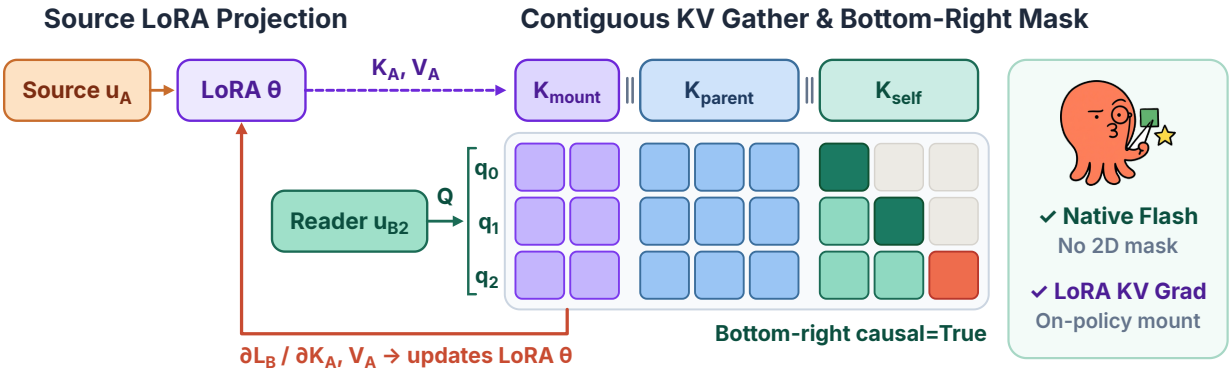


Figure 6: Contiguous KV gathering and cross-branch LoRA backpropagation (Sections 3.3 and 3.4). Concatenating mounted sibling keys, ancestor prefixes, and active reader tokens places the causal diagonal at the bottom-right corner for native FlashAttention (`causal=True`), while routing $\partial \mathcal{L}_{u_B} / \partial \mathbf{K}_{u_A}, \mathbf{V}_{u_A}$ directly into source branch A's LoRA adapter θ.

Training Engine: Tree-to-1D Contiguous Packed DAG Kernels. Splitting a branched episode into separate leaf sequences during RL either recomputes shared trunks K times or drops mounted KV spans. Octopus instead compiles each episode DAG into a single 1D packed forward–backward pass (Figure 5):

1. **Node Splitting and Prefix Deduplication:** At every `fork`, `rewind`, or `read_thread` boundary, Octopus partitions the episode into maximal contiguous token nodes $u = \big(p(u),\ \mathcal{M}(u),\ s(u)\big)$, recording tree parent $p(u)$, active mounted segments $\mathcal{M}(u)$ (including transitive mounts), and monotonic start position $s(u)$ (Equation (2)). Each unique node u is evaluated once regardless of how many branches or rewinds descend from it.
2. **1D Contiguous MLP and Projection Pass:** Deduplicated nodes $\{u_1, \ldots, u_M\}$ are concatenated topologically into one unpadded 1D tensor (up to 262K tokens) and run through all MLPs and QKV projections in one matrix pass.
3. **Dual-Mechanism Graph Routing:** Across the packed 1D strip, recurrent and attention layers route activations along $\mathcal{G}$ via two complementary hooks. On recurrent layers (Gated DeltaNet and Mamba-2), boundary hooks initialize each node u from its parent's terminal state $\mathbf{h}_{\text{end}}(p(u))$ (plus the last $w-1=3$ causal-convolution tokens), accumulate $\partial \mathcal{L} / \partial \mathbf{h}$ back to $p(u)$ in backward, and skip $\mathcal{M}(u)$ (Figure 5). On each mounted full-attention layer $l \in \mathcal{L}_{\text{mount}}$, node u gathers precomputed K/V tensors as (Figure 6):

$$\mathbf{K}_u^{(l)} = \Big[\bigoplus_{m \in \mathcal{M}(u)} \mathbf{K}_m^{(l)} \,\Big\|\, \mathbf{K}_{p(u)}^{(l)} \,\Big\|\, \mathbf{K}_{\text{self}(u)}^{(l)} \Big], \qquad \forall l \in \mathcal{L}_{\text{mount}}. \tag{4}$$

Placing self(u) at the rightmost tail exploits rectangular FlashAttention (`causal=True`; Figure 4(d), Figure 6): whenever $|\mathbf{Q}_u| < |\mathbf{K}_u|$, FlashAttention aligns the causal triangle to the rightmost $|\mathbf{Q}_u|$ keys (self(u)) while leaving all preceding mounted and ancestor keys unmasked, avoiding custom 2D block-sparse masks and matching live SGLang rollouts to $|\Delta \log \pi_\theta| = 0.0038$.

3.4 DAG-GRPO Training with Single-Ownership Masking and Mounted KV Edges

Single-Ownership Token Deduplication and RLOO Advantage. For each prompt, Octopus samples G branched episode trees $\{\tau_1, \ldots, \tau_G\}$ and scores each terminal outcome $y(\tau_i) \in [0, 1]$ with binary reward $R(\tau_i) = 2y(\tau_i) - 1$. Following REINFORCE Leave-One-Out (RLOO) [43], we compute a critic-free group advantage without standard-deviation normalization (Figure 7):

$$\hat{A}(\tau_i) = R(\tau_i) - \frac{1}{G-1} \sum_{j \neq i} R(\tau_j), \tag{5}$$

and broadcast $\hat{A}(\tau_i)$ across all threads of τ_i, including rewound leaves so the `rewind` decision itself receives outcome credit. Standard sequence-RL objectives sum token losses along root-to-leaf trajectories $\{\ell \in \tau_i\}$, which on a tree naively weights each shared trunk node u by its number of descendant leaves. To map trajectory-level RL directly onto our deduplicated 1D packed pass (Section 3.3), we assign each node u to the unique leaf $\ell = \mathrm{owner}(u)$ that first generated it via a single-ownership mask over action tokens $\mathcal{V}_{\text{gen}}(\tau_i)$:

$$m_{\ell,t} = \mathbb{I}\Big[\ell = \mathrm{owner}(u_{\ell,t}) \ \wedge \ x_{\ell,t} \in \mathcal{V}_{\text{gen}}(\tau_i)\Big], \tag{6}$$

setting $m_{\ell,t} = 1$ once per generated token on the packed 1D strip and $m_{\ell,t} = 0$ on prompts, tool outputs, inherited or pre-rewind prefixes ($t \leq k$), and mounted sibling spans $\mathcal{M}(u)$. Summing with $m_{\ell,t}$ recovers the exact DAG score function:

$$\sum_{\ell \in \tau_i} \sum_{t=1}^{|\ell|} m_{\ell,t} \nabla_\theta \log \pi_\theta(x_{\ell,t} \mid \mathcal{G}_{\ell,<t}) = \nabla_\theta \log \mathbb{P}_\theta(\tau_i). \tag{7}$$

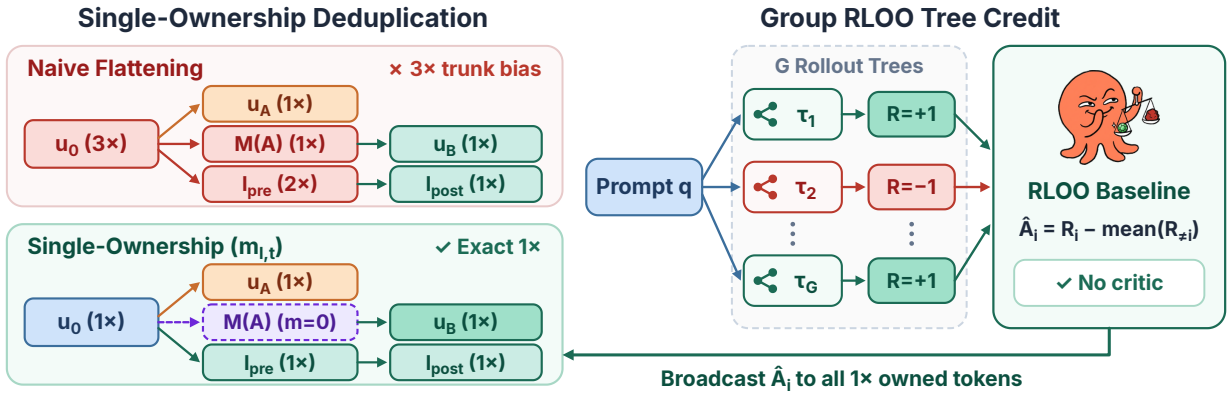


Figure 7: Single-ownership prefix deduplication and Group RLOO tree credit assignment in DAG-GRPO (Section 3.4). Single-ownership masking ($m_{\ell,t} \in \{0, 1\}$) counts every generated tree segment once while masking mounted spans, and broadcasts critic-free Leave-One-Out group advantages $\hat{A}(\tau_i)$ across all owned tokens.

Differentiable On-Policy Mount Edges and Clipped Objective. During the 1D packed pass, source node u_A's keys and values at each mounted layer $l \in \mathcal{L}_{\text{mount}}$ are projected through trainable LoRA weights $\theta = \{\mathbf{B}_\theta, \mathbf{A}_\theta\}$:

$$\mathbf{K}^{(l)}_{u_A}(\theta) = \big(\mathbf{W}^{(l)}_K + \mathbf{B}^{(l)}_{K,\theta}\mathbf{A}^{(l)}_{K,\theta}\big)\mathbf{H}^{(l)}_{u_A}(\theta), \qquad \mathbf{V}^{(l)}_{u_A}(\theta) = \big(\mathbf{W}^{(l)}_V + \mathbf{B}^{(l)}_{V,\theta}\mathbf{A}^{(l)}_{V,\theta}\big)\mathbf{H}^{(l)}_{u_A}(\theta), \tag{8}$$

and gathered directly by reader node u_B in Equation (4) without detaching the mount edge. Mounted spans carry loss mask $m_{\ell,t} = 0$ so A's tokens are never double-counted as B's actions, while attention backpropagation routes $\partial \mathcal{L}_{u_B}/\partial \mathbf{K}^{(l)}_{u_A}$ and $\partial \mathcal{L}_{u_B}/\partial \mathbf{V}^{(l)}_{u_A}$ directly into A's LoRA weights (Figure 6). Unlike text-summary branching—which trains θ only on re-encoded summary tokens—keeping live mounts in the packed graph preserves on-policy rollout alignment ($\mathrm{KL} = 4.45 \times 10^{-4}$ vs. 0.0142 when dropped; Section 4.3) and lets reader B's loss directly shape how θ encodes producer A's KV state. We optimize the clipped DAG-GRPO objective:

$$\mathcal{J}_{\text{DAG-GRPO}}(\theta) = \mathbb{E}_{\{\tau_i\}_{i=1}^{G}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{Z_i}\sum_{\ell \in \tau_i}\sum_{t=1}^{|\ell|} m_{\ell,t}\, \min\Big(r_{\ell,t}(\theta)\hat{A}(\tau_i),\ \mathrm{clip}\big(r_{\ell,t}(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}(\tau_i)\Big)\right], \tag{9}$$

where $Z_i = \sum_{\ell \in \tau_i}\sum_{t=1}^{|\ell|} m_{\ell,t}$ counts owned generated tokens in τ_i and $r_{\ell,t}(\theta) = \pi_\theta(x_{\ell,t} \mid \mathcal{G}_{\ell,<t})/\pi_{\theta_{\text{old}}}(x_{\ell,t} \mid \mathcal{G}_{\ell,<t})$. Normalizing by $1/Z_i$ stabilizes gradient magnitudes across trees spanning 1K to 199K tokens, though analogous to Dr. GRPO [44] it introduces a tree-size length weighting that softens per-token penalties on failed large trees (Section 5).

4 Experiments

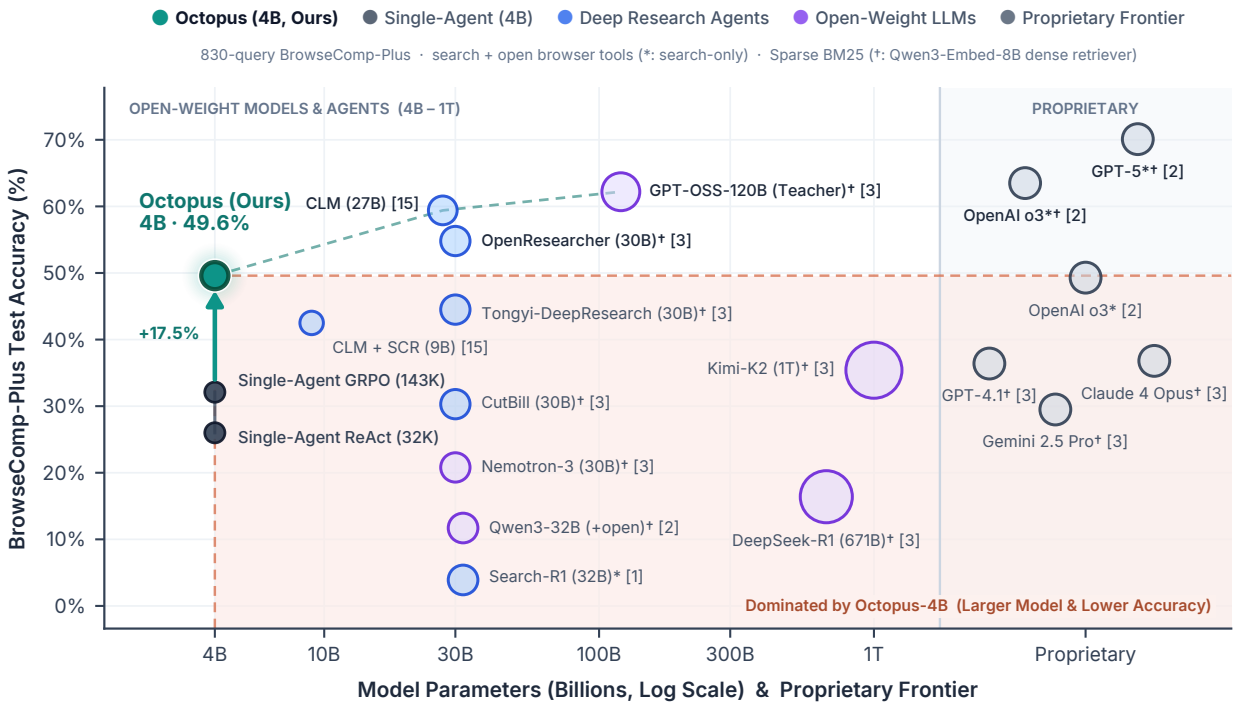


Figure 8: BrowseComp-Plus test accuracy (830 queries) vs. model parameter scale (4B–1T log scale and proprietary models) [2]. Systems use browser reading tools (`search+open[+find]`) over sparse BM25 unless marked * (`search`-only) or † (`Qwen3-Embed-8B` dense retrieval). **Octopus (4B)** reaches **49.6**% under BM25, dominating larger open-weight and proprietary systems across the shaded region.

We evaluate Octopus around three questions: *(i)* ***Deep Search Performance (Section 4.1)***: How does a 4B branching agent trained with Octopus compare against proprietary frontier models, 9B–1T open-weight systems, and controlled 4B single-agent baselines (Figure 8, Table 3)? *(ii)* ***Context and KV-Sharing Efficiency (Section 4.2)***: Does branch-isolated 32K exploration convert context into gold evidence and reuse GPU prefix caches better than flat 143K windows or in-place edits, and does attention-only KV mounting prevent off-task hijacking (Figure 9, Tables 4 and 5)? *(iii)* ***Multi-Agent Scalability (Section 4.3)***: How does accuracy scale with deployed agent count and unlimited tree depth ($D_{\max} = \infty$), and what collective behaviors emerge under pure outcome RL (Figure 10, Table 6)?

4.1 End-to-End Deep Search Performance

Experimental Setup, Corpus Isolation, and Compute Accounting. We train Qwen3.5-4B (rank-64 LoRA, $B_{\text{thread}} = 32{,}768$ with 2,048-token answer reserve; $G = 8$, batch size 16) on 3,040 OpenResearcher queries (500 held out for Validation, stratified by teacher pass rate from the 6,102-query MiroVerse pool [3]) over the 14.9M-document `OpenResearcher-Corpus`, and evaluate zero-shot on BrowseComp-Plus (830 queries) over its separate 100,195-document fixed corpus [2]. Cross-dataset auditing confirms 0/3,040 exact or 13-gram question overlap (max token Jaccard 0.265) and only 2/8,306 (0.02%) training gold URLs in BrowseComp-Plus gold docs (1.57% in the 100K corpus, 93.8% strictly distractors; two general Wikipedia pages with zero shared answers, leaving test accuracy unchanged at 49.6% when excluded). Both corpora use sparse BM25 (10×512-token `search` hits; vs. $12\times$ higher-recall dense† and snippet-only* baselines in Figure 8 and Table 3) and paginated 8K-token `open`, graded by Qwen3.5-27B at $T = 0$ [15]. Following Shao et al. [15], we report prefix-reuse inference PFLOPs per query (PF/Q; 89.59% GPU KV hit rate, 8-layer `read_thread`) and critical-path vs. total completion tokens (**Crit.** C **/ Total**).

Controlled Attribution and Parameter–Accuracy Frontier. On Qwen3.5-4B under sparse BM25 (Table 3), single-agent ReAct saturates its 32K window within five to six turns and plateaus at 24.6%–26.0%. Stretching one flat window to 143K under GRPO (**Single-Agent GRPO**) reaches only 32.1% (43.2% Validation), stalling after 40.8K tokens (28.5% utilization) as unpruned retrieval noise pollutes working memory. Exposing branching tools without RL (**Multi-Agent Fork, 0-Shot**) stays flat at 26.3% (0.41 branches/episode). Training depth-1 teams with DAG-GRPO lifts

Table 3: End-to-end deep search results on **BrowseComp-Plus** (830 test queries) [2] and **OpenResearcher Validation** (500 queries) [3]. Columns report prefix-reuse inference PFLOPs (**PF/Q**), accuracy under sparse BM25 (†dense retrieval, *`search`-only), evidence recall, and critical-path vs. total completion tokens (**Crit.** C **/ Total**). **Octopus (4B)** lifts BrowseComp-Plus accuracy by $+17.5$ points over 143K Single-Agent GRPO.

Method / Agent System	Backbone / Scale	Tools	PF/Q ↓	OR Val Val Acc. ↑	BrowseComp-Plus Official Test Benchmark (830 Queries) Test Acc. ↑ BM25 / Dense†	Search Rec. ↑	Open Rec. ↑	Gold Rec. ↑	Context Used / Cap	Crit. C / Total
Proprietary Frontier Models on BrowseComp-Plus [2, 3]										
Claude Sonnet 4 [2]	Proprietary	`s-only`*	—	—	14.3%* (36.8%*†)	21.3%*	—	—	≤ 200K	1.00×
Gemini 2.5 Pro [2, 3]	Proprietary	`s+o+f`	—	—	29.5%† (19.0%*)	22.8%*	—	—	≤ 1M	1.00×
GPT-4.1 [2, 3]◇	Proprietary	`s+o+f`	—	—	36.4%† (14.6%*)	16.4%*	—	—	≤ 128K	1.00×
Claude 4 Opus [2, 3]	Proprietary	`s+o+f`	—	—	36.8%† (15.5%*)	23.0%*	—	—	≤ 200K	1.00×
OpenAI o3 [2]	Proprietary	`s-only`*	—	—	49.3%* (63.5%*†)	56.6%*	—	—	≤ 200K	1.00×
GPT-5 [2]	Proprietary	`s-only`*	—	—	55.9%* (70.1%*†)	61.7%*	—	—	≤ 256K	1.00×
Open-Weight Flagship LLMs & Specialized Deep Research Agents (9B–1T Backbones)										
Search-R1-32B (RL) [1, 2]	32B Dense	`s-only`*	0.46	—	3.9%* (10.4%*†)	2.6%*	—	—	≤ 32.0K	1.00×
Qwen3-32B (+open) [2]	32B Dense	`s+o`	0.36*	—	11.7%† (3.5%*)	3.1%*	—	—	≤ 32.0K	1.00×
DeepSeek-R1 [3, 28]	671B-A37B	`s+o+f`	—	—	16.4%†	—	—	—	≤ 128K	1.00×
Nemotron-3-Nano [3]	30B-A3B	`s+o+f`	—	—	20.8%†	—	—	—	≤ 128K	1.00×
CutBill [3]	30B-A3B	`s+o+f`	—	—	30.3%†	—	—	—	≤ 128K	1.00×
Kimi-K2 [3]	1T-A32B	`s+o+f`	—	—	35.4%†	—	—	—	≤ 128K	1.00×
Context-Summary (0-Shot → RL) [15]	Qwen3.5-9B	`s+o+sum`	4.01→2.19	—	34.7→42.1%	—	—	—	≤ 32.0K	1.00×
CLM + SCR (0-Shot → RL) [15]	Qwen3.5-9B	`s+o+ret`	1.52→1.34	—	28.8→42.5%	—	—	—	≤ 32.0K	1.00×
Tongyi-DeepResearch [3]	30B-A3B	`s+o+f`	—	—	44.5%†	—	—	—	≤ 128K	1.00×
OpenResearcher (SFT) [3]	30B-A3B	`s+o+f`	—	—	54.8%†	—	—	—	≤ 128K	1.00×
CLM [15]‡	Qwen3.6-27B	`s+o+ret`	5.40	—	59.4%	—	—	—	≤ 32.0K	1.00×
GPT-OSS-120B (Teacher) [2, 3]	120B MoE	`s+o(+f)`	—	—	56.4 / 62.2%† (28.7%*)	35.5%*	—	—	≤ 128K	1.00×
Controlled Comparison Arms on Qwen3.5-4B (BM25 Index, Rank-64 LoRA, $B_{\text{thread}} = 32{,}768$ Per-Thread Cap)										
Single-Agent ReAct (100-Turn Cap)	Qwen3.5-4B	`s+o`	0.28	40.0%	24.6%	25.9%	6.4%	9.7%	28.6K /32K	1,097 / 1,097
Single-Agent ReAct (32K Window)	Qwen3.5-4B	`s+o`	0.33	40.0%	26.0%	28.6%	7.5%	12.2%	32.6K /32K	1,043 / 1,043
Multi-Agent Fork (0-Shot, 0.41 br)	Qwen3.5-4B	`s+o+tree`	0.38	41.6%	26.3% (24.8%)	30.5%	8.6%	12.5%	38.1K	1,590 / 1,811
Single-Agent GRPO (143K Window)	Qwen3.5-4B	`s+o`	0.43	43.2%	32.1%	30.5%	14.6%	21.1%	40.8K /143K	4,075 / 4,075
Octopus (Process-Gated RL, $D_{\max}$=1)	Qwen3.5-4B	`s+o+tree`	0.93	45.2%	39.5%	—	—	—	98.4K	4,877 / 9,120
Octopus (Pure Outcome, $D_{\max}$=1)	Qwen3.5-4B	`s+o+tree`	1.57 (0.70§)	**51.2%**§ **(+8.0)**	42.6% **(+10.5)**	49.5%	29.0%	37.7%	143.5K	9,918 / 18,418
Octopus (Ours, Recursive $D_{\max}$=∞) DAG-GRPO (6.41 br, 1-Epoch§)	**Qwen3.5-4B**	`s+o+tree`	**2.00**	**46.6%**§ **(Step 20)**	**49.6% (+17.5)**	**58.2%**	**36.0%**	**43.3% (+22.2)**	**199.0K**¶ **(329K expl.)**	**9,919 / 19,251**

Tools: `s=search`, `o=paginated open`, `f=find`, `sum/ret=summary/retention`, `tree=fork/rewind/read_thread`. **Rec.** columns report per-query fractional recall over evidence (6.1/query) and gold (2.9/query) docs (58.9% open ≥ 1 gold doc vs. 20.7% 0-shot). †`Qwen3-Embed-8B` dense retrieval (14.5% vs. 1.2% BM25 Recall@5 [2]); GPT-OSS-120B: 56.4% (`s+o`) / 62.2%† (`s+o+f`) [3], 28.7%* / 42.9%*† [2]. ◇GPT-4.1: 43.6%† with full `get_document`. ‡On 150 dense queries, Context-Folding scores 62.0% vs. 44.6% ReAct+GRPO (Seed-OSS-36B [25]). §$D_{\max} = 1$: Step 35 scores 42.6% BCP (46.5% Val, 1.57 PF/Q), Step 55 scores 51.2% Val (0.70 PF/Q); $D_{\max} = \infty$ ($W_{\max} = 10$, Step 45 = 1 epoch): 49.6% BCP (409/824, CI [46.2%, 53.0%]; 49.3% strict /830), 46.6% Val at Step 20. ¶Net post-`rewind` context (329.2K explored).

BrowseComp-Plus accuracy to 39.5% when gated by heuristic process penalties (**Process-Gated RL**) and 42.6% under pure binary outcome credit ($+10.5$ points; $\mathbf{51.2}\%$ on OpenResearcher Validation at 0.70 PF/Q). Removing the depth cap for a single epoch ($D_{\max} = \infty, W_{\max} = 10$) reaches $\mathbf{49.6}\%$ on BrowseComp-Plus (49.3% strict /830)—gaining $+\mathbf{17.5}$ points over 143K Single-Agent GRPO and $+\mathbf{23.3}$ paired points over the 0-shot base policy at 2.00 PF/Q.

Across model scales (Figure 8), our 4B BM25 agent outperforms 2.25× larger 9B context-editing RL agents on the identical BM25 harness by $+7.1$–7.5 points (42.1% Context-Summary+RL at 2.19 PF/Q; 42.5% CLM+SCR at 1.34 PF/Q) [15] and edges past OpenAI o3 under BM25 (49.3%*). Moreover, Octopus-4B under sparse BM25 surpasses much larger open-weight and proprietary agents equipped with 12× higher-recall dense retrieval (†), including 671B DeepSeek-R1 (16.4%†), 1M-window Gemini 2.5 Pro (29.5%†), 1T Kimi-K2 (35.4%†), Claude 4 Opus (36.8%†), and 30B Tongyi-DeepResearch (44.5%†). Only 7×–30× larger specialized open-weight systems (27B–120B, 54.8%–62.2%) and dense-retriever o3/GPT-5 (63.5%*†–70.1%*†) score higher, placing Octopus-4B atop the sub-27B frontier.

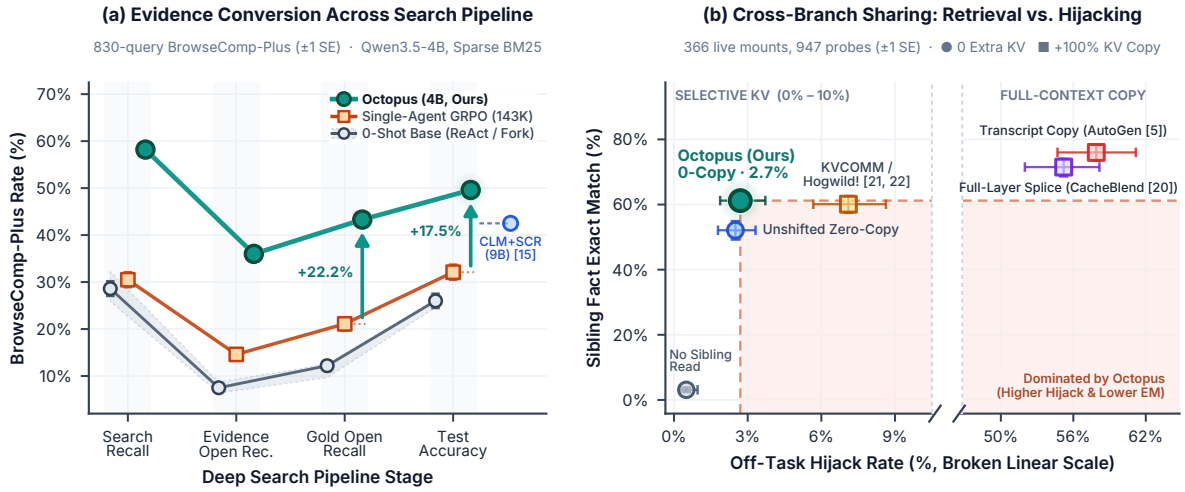


Figure 9: Context conversion efficiency and cross-branch KV sharing on Qwen3.5-4B (±1 SE). **(a)** Four-stage BrowseComp-Plus recall-to-accuracy pipeline (830 queries; per-query recall over 6.1 evidence / 2.9 gold docs, 58.9% opening $\geq$ 1 gold doc). **(b)** Sibling exact match vs. task hijacking across 366 live `read_thread` mounts (947 probes): **Octopus** retains 92.2% answer information at 2.7% hijacking with 0 extra KV pages.

4.2 Context Utilization and Compute Efficiency

Branch-Isolated Context Conversion, Evidence Grounding, and Radix Prefix Reuse. Figure 9(a) and Table 4 trace context conversion across the four BrowseComp-Plus stages (search, evidence-open, and gold-open recall, plus accuracy). Both 32K ReAct and 0-shot forking stall at 9.7%–12.5% gold-open recall (20.7% opening $\geq$ 1 gold doc). Stretching a flat window to 143K under GRPO fails to unlock deeper search: with 0% pruning of failed detours and noisy 8K pages, generated tokens bloat 3.9× (1,043 $\rightarrow$ 4,075) while usage stalls at $\mathbf{40.8K/143K}$ (28.5%, 21.1% gold-open recall).

By contrast, Octopus isolates sub-queries into clean 32K branches and prunes 38.8% of post-initial steps via `rewind` (61.4 steps, 6.99 rewinds/episode), keeping active prompts at $\mathbf{26.4}$K tokens while exploring $\mathbf{329.2}$K ($\mathbf{12.5}\times$ active; 541.3K on $\geq$ 250-call queries) and retaining $\mathbf{199.0}$K net tokens. Parallel execution compresses 19,251 generated tokens into a 9,919-token critical path ($\mathbf{1.94}\times$), lifting fractional gold-open recall from 21.1% to $\mathbf{43.3}$% (58.2% search, 36.0% evidence-open recall). Because BrowseComp-Plus annotates 2.89 gold docs/query (67.2% multi-gold) and episodes stop once an opened gold page verifies the answer, Octopus opens $\geq$ 1 gold doc on $\mathbf{58.9}$% of queries (485/824; 81.9% conditional accuracy vs. 3.5% otherwise): of 409 correct answers (49.6%), $\mathbf{97.1}$% (397/409) open $\geq$ 1 gold doc and 2.4% (10/409) open a non-gold evidence doc, while snippet-only answers drop from 25.3% (55/217) at 0-shot to $\mathbf{0.5}$% (2/409) and zero-evidence parametric answers are $\mathbf{0.0}$% (0/155). At serving time, unlike in-place edits (CLM+SCR [15], 41.8% edit re-prefill, breaking $\mathbf{h}_t$), Octopus reuses $\mathbf{89.59}$% of prompt tokens from GPU KV cache across 447,993 H200 prefill batches (14.56B vs. 1.69B prefilled), cuts wire traffic $\mathbf{98.3}$% (662.3 $\rightarrow$ 11.5 MB), and speeds up 128K assembly $\mathbf{249}\times$ (358.5 $\rightarrow$ 1.44 ms).

Table 4: Per-episode context conversion efficiency on BrowseComp-Plus (830 queries) and server-resident radix prefix reuse across 447,993 live H200 prefill batches.

Context Management & Serving Architecture	Explored Context (Active / Win. Cap)	Critical-Path Token Load	Gold Open Rec. / BCP Acc. ↑	GPU Prefix KV Cache Reuse ↑	128K Assembly & Wire Cost ↓
Single-Agent ReAct (Stateless 32K)	32.6K (100% of 32K cap)	1,043 tok (1.00×)	12.2% / 26.0%	Append-only ($\mathcal{O}(L)$ resend)	358.5 ms / 662.3 MB
Single-Agent GRPO (Flat 143K Window)	40.8K (28.5% of 143K cap)	4,075 tok (3.9× bloat)	21.1% / 32.1%	Polluted window (0% pruned)	358.5 ms / 662.3 MB
In-Place Context Edit (CLM+SCR, 9B) [15]	Multi-edit ($\leq$ 32K active cap)	1.00× serial edits	— / 42.5% (1.34 PF)	41.8% edit re-prefill (breaks $\mathbf{h}_t$)	Suffix copy & re-rotate
Octopus (Ours, 32K Branches) **Server-Resident Radix Cursor**	**329.2K expl. / 199.0K net** (26.4K active; 541K hard)	**1.94× compressed** (9,919/19,251 tok)	43.3% / 49.6% (58.9% $\geq$ 1 gold; +22.2)	**89.59% GPU KV Hit** (0% prefix re-prefill)	**1.44 ms** (249× ↑) (11.5 MB, −98.3%)

Hijack-Free Cross-Branch KV Mounting. Figure 9(b) and Table 5 benchmark cross-branch sharing across 366 live `read_thread` mounts on Qwen3.5-4B (947 exact-match identifier probes and 368 fact-recall probes; in 52% of mounts, sibling A ran longer than reader B, $e_A > p_B$). Copying A's transcript as text (**Transcript Copy** [5]) or splicing KV caches across all layers including recurrent states (**Full-Layer KV Splice** [20]) allocates +100% extra KV

memory and steps B's 24 Gated DeltaNet states $\mathbf{h}_B$ through A's tokens. As predicted in Section 3.2, overwriting $\mathbf{h}_B$ distorts B's policy (ΔNLL $= +0.195$–0.196) and hijacks 55.2%–57.9% of post-read turns into pursuing A's sub-goal. Mounting attention KV without shifting positions (**Unshifted Zero-Copy KV**) avoids memory copies but drops from 0.612 to 0.449 exact match whenever $e_A > p_B$ due to inverted RoPE phases, whereas copying and re-rotating keys (**RoPE-Relocated KV Copy** [21, 22]) doubles KV memory and still incurs 7.1% hijacking.

Octopus Zero-Copy Mount (Equation (2)) splices sibling KV pages exclusively into the 8 full-attention layers under monotonic position offsets while all 24 recurrent layers skip the span: it allocates 0 **extra KV pages**, retains **92.2**% of full-text answer log-probability gain (96.5%–97.3% fact recall) with no degradation on longer siblings (0.613 vs. 0.612 exact match), cuts policy drift by **29**× (ΔNLL $= +0.007$), and reduces off-task hijacking by **21**× (57.9% → **2.7**%). Across 115 live rollouts, a blind judge finds no excess drift relative to a control that blocks the read (ΔHijack $= -1.1$%), and the mechanism transfers cleanly to dense Transformers (Qwen3-4B, **96.9**% argmax agreement) and Mamba-2 hybrids (Granite-4.0-H, **98.5**%; Appendix C).

Table 5: Cross-branch context-sharing efficiency across 366 live `read_thread` mounts (947 probes on Qwen3.5-4B; 52% with longer sibling $e_A > p_B$).

Cross-Branch Context-Sharing Mechanism	Extra KV Memory ↓	Full-Text Info Retained ↑	Exact Match ↑ All (Longer Sib. $e_A > p_B$)	Policy Drift ΔNLL ↓	Off-Task Hijack Rate ↓
No Sibling Read (`none`)	0 Pages	0.0% (62.5% rec.)	0.031 (0.032)	0.000	0.5%
Transcript Copy (AutoGen [5])	+100% (+L_{mount})	**100**% (**99.5**% rec.)	**0.760** (0.786)	+0.196	57.9%
Full-Layer KV Splice (CacheBlend [20])	+100% (+L_{mount})	98.1% (99.2% rec.)	0.715 (0.731)	+0.195	55.2%
Unshifted Zero-Copy KV	0 Pages (0-Copy)	86.1% (94.3% rec.)	0.521 (0.449)	+0.003	2.5%
RoPE-Relocated KV Copy (KVCOMM [21, 22])	+100% (+L_{mount})	91.7% (96.7% rec.)	0.601 (0.611)	+0.015	7.1%
Octopus Zero-Copy Mount (Ours) Attn-Only Monotonic Offset (Equation (2))	0 **Pages (0-Copy)**	92.2% **Info** (96.5%–97.3% rec.)	0.612 (0.613) **(0-shot:** 0.627)	+0.007 (29× ↓ vs. Text)	2.7% (3.8% **0-shot**) (21× ↓ **vs. Text**)

Cross-Architecture Transfer: Qwen3-4B (Dense, 9/36L) **96.9**% · Granite-4.0-H (Mamba-2 Hybrid) **98.5**%. **Live Blind Audit** (115 **Reads):** ΔHijack $= -$**1.1**% vs. Blocked-Read Control.

4.3 Multi-Agent Scalability Across Agent Count and Tree Depth

Scaling Accuracy with Deployed Agents and Causal Fork Knockout. Figure 10(a) and Table 6 track performance from a single linear thread ($W_{\max} = 0$) to depth-1 parallel teams ($D_{\max} = 1, W_{\max} = 10$) and unlimited-depth hierarchies ($D_{\max} = \infty, W_{\max} = 10$, up to 11 threads/episode). Within 25,812 depth-1 training rollouts, branched trajectories reach 61%–66% accuracy versus 52% for unbranched rollouts on the same query, whereas rewinding an unbranched root yields no gain ($\Delta y = -0.014$); accordingly, DAG-GRPO drives root forking from 27% to 98% while cutting root rewinds 15× (0.413 → 0.026/episode). Held-out accuracy climbs monotonically with deployed agents per episode (Figure 10(a)): on OpenResearcher Validation, accuracy rises from 40.0% (32K ReAct) and 43.2% (143K GRPO) across 1.3–2.7 agents/episode to **51.2**% (+8.0 points); on BrowseComp-Plus, accuracy scales from 26.0% (32K ReAct) and 32.1% (143K GRPO) to 42.6% at 4.18 agents/episode ($D_{\max} = 1$) and **49.6**% at **7.41** agents/episode ($D_{\max} = \infty$, 6.41 workers). Recursive branching rescues **31.8**% (116/365) of queries that hit the 32K single-agent context wall (0% rescued at 0-shot) and boosts high-hop third-quartile (Q3) queries by **10**× (3.4% → 34.1%).

Crucially, we cap fanout at $W_{\max} = 10$ only to match 9B single-agent RL compute (2.00 vs. 2.19 PF/Q): episodes finishing naturally within 10 workers (69.2%, 570/824) achieve **71.8**% accuracy while 39.0% hit the 10-worker cap, indicating that scaling is budget-limited rather than saturated. Conversely, disabling `fork` at test time on the exact same Step 45 weights ($W_{\max} = 0$) drops paired BrowseComp-Plus accuracy by $-$**21.7** points (50.8% → 29.1% across 571 paired queries; 45.4% on Validation), confirming that gains come from live multi-agent execution rather than single-thread weight distillation.

Step-15 Phase Transition, Emergent Peer KV Mesh, and 1D Training Scaling. Without our 1-turn untrainable close reminder (Appendix A), naive $D_{\max} = \infty$ RL collapses from 28.2% to 0% forking within 15 steps (Figure 10(b)). With the close contract (24,041 episodes over 45 steps), Octopus plateaus for 14 steps and then undergoes a sharp topological phase transition at Step 15: deep trees reaching $d \geq 3/4/5$ surge to **59.3**%/**39.1**%/**21.5**% (54× increase at $d \geq 5$), yet the mean children spawned per worker remains sub-critical at $\mu \in [0.32, 0.69] < 1.0$, bounding expected tree size without depth caps (6.41 workers/episode on BrowseComp-Plus).

Driven solely by binary 0/1 terminal rewards (Figure 10(c,d)), Step 15 simultaneously self-organizes three division-of-labor behaviors: *(i)* Turn-1 root delegation jumps 38× (2.6% → **98.0**%) while workers absorb **92.8**% of 8K page `open` calls, keeping root contexts clean; *(ii)* branches form a lateral `read_thread` KV mesh (17.3% → **85.9**% of

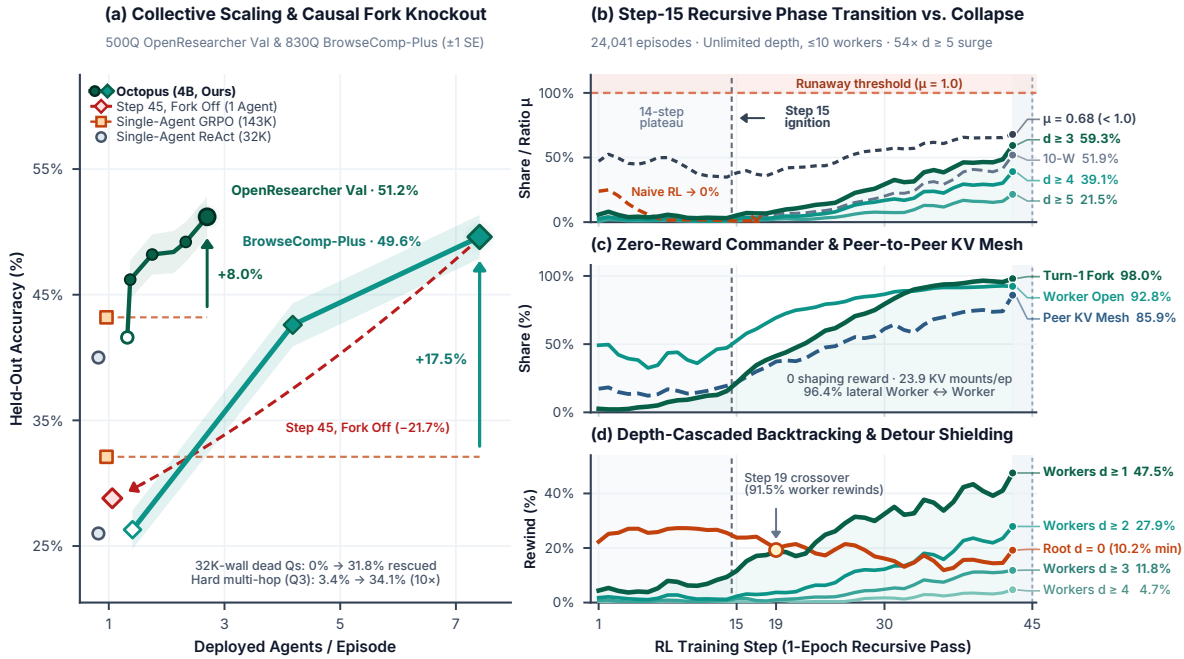


Figure 10: Multi-agent scaling and Step-15 recursive phase transition on Qwen3.5-4B. **(a)** Held-out accuracy vs. deployed agents per episode and same-weight `fork` knockout ($W_{\max} = 0, \pm 1$ SE). **(b–d)** Unlimited-depth RL dynamics (24,041 episodes, $D_{\max} = \infty, W_{\max} = 10$), showing sub-critical recursive scaling ($\mu < 1$), emergent peer-to-peer KV mounting, and depth-cascaded worker `rewinds`.

episodes, averaging **23.9** zero-copy mounts/episode with **96.4**% peer worker↔worker reads); and *(iii)* past Step 19, root rewinds drop nearly 3× (29.3% → 10.2% of episodes) as workers absorb **91.5**% of all `rewind` calls (47.5% of episodes at $d \geq 1$ and 27.9% at $d \geq 2$; Appendix B). Finally, 1D prefix-tree deduplication with live mounted KV edges (Sections 3.3 and 3.4) scales owned generated tokens trained per step by **8.1**× on average (**11.1**× peak, 101.6K → 1.12M tokens/step, 0 OOM) while holding rollout–training divergence at $\mathrm{KL}(\pi_{\text{rollout}} \,\|\, \pi_\theta) = 4.45 \times 10^{-4}$—**32**× lower than dropping mounted KV edges (0.0142).

Table 6: Multi-agent scalability across worker budget ($W_{\max}$), recursive tree depth ($D_{\max}$), and emergent division of labor on Qwen3.5-4B.

Multi-Agent Topology & Training Regime	Max Threads (Depth, Fanout)	Root Fork Rate	Workers / Ep (Child Ratio μ)	Deep ($d \geq 3$) / Full 10-Br.	Rewind Activity (Root / Worker)	OpenResearcher Acc. ↑	BCP Test Acc. ↑
Single-Agent ReAct (32K Window)	1 (0, 0)	0%	0.00 (0.00)	0% / 0%	— / —	40.0% (Val)	26.0%
Single-Agent GRPO (143K Flat)	1 (0, 0)	0%	0.00 (0.00)	0% / 0%	— / —	43.2% (Val)	32.1%
Multi-Agent Fork (0-Shot Base)	≤ 11 (∞, 10)	11% (16% Val)	0.41 (0.32 Val)	0% / 0%	1.51 rw/ep (9.7 st)	41.6% (Val)	26.3% (24.8%)
Octopus RL (Parallel Team, $D_{\max}$=1, Step 35)	≤ 11 (1, 10)	92%	3.18 (0.00)	0% / 0.2%	0.29/ep / 0.52/br	46.5% **(Val)**	42.6% **(+10.5)**
Octopus RL (Parallel Team, $D_{\max}$=1, Step 55)	≤ 11 (1, 10)	98%	1.70 (0.00)	0% / 0.6%	0.026**/ep** / 0.24**/br**	**51.2% (Val)**	—
Recursive 0-Shot ($D_{\max}$=∞ on $D_{\max}$=1 Policy)	≤ 11 (∞, 10)	97%	2.34 (0.27)	8.6% / —	— / —	44.4% (57.2%‡)	—
Octopus RL (Recursive, $D_{\max}$=∞, Step 20)	≤ 11 (∞, 10)	55%	1.15 (0.46)	11.4% / 8.9%	23.7% / 18.6% ep	46.6% **(Val)**	—
Octopus RL (Recursive, $D_{\max}$=∞, Step 43 Rollout‡)	≤ 11 (∞, 10)	99.2%	6.85 (0.68)	59.3% / 51.9%	10.2% / 47.5% ep	57.9% **(Train‡)**	—
Octopus RL (Recursive, $D_{\max}$=∞, Step 45, 1-Ep.)	≤ 11 (∞, 10)	99.5%	6.41 (7.41 **ag.**)	— / 39.0%	6.99 **rw/ep** (61.4 st)	45.4→46.5% ($W \leq 1$‡)	**49.6% (+17.5)**
1D Prefix-Tree Deduplication (Section 3.3): 8.1× avg (**11.1**× peak, 101.6K→1.12M tok/step, 0 OOM). **On-Policy Mount KL: 4.45×10^{-4}** (vs. 0.0142 w/o mounts, **32**× ↓).							

‡Zero-shot $D_{\max} = \infty$ transfer of the $D_{\max} = 1$ policy scores 44.4% across all 500 Val queries (222/500) and 57.2% on completed trees (222/388). Step 43 reports late-training rollout statistics (57.9% training rollout accuracy); at Step 45 (1 epoch = 3,040 queries), restricting test-time fanout on Val to $W_{\max} = 0/1$ yields 45.4%/46.5%, while full recursive evaluation ($W_{\max} = 10$) reaches **49.6**% on BrowseComp-Plus.

5 Conclusion and Limitations

Long-horizon reasoning agents need not be confined to a single append-only text window. **Octopus** recasts agentic execution as a stateful, GPU-resident DAG in which branches fork and backtrack over reference-counted radix KV prefixes and recurrent checkpoints in $\mathcal{O}(1)$ zero copy, while exchanging findings directly through attention-only KV mounting without overwriting private recurrent working memory (92.2% information retention, $21\times$ lower task hijacking, 89.6% live GPU KV hit rate). Trained end-to-end with 1D packed DAG-GRPO under pure binary outcome rewards and zero process shaping, a 4B sparse-BM25 model self-organizes a sub-critical commander–mesh hierarchy—reaching 49.6% on BrowseComp-Plus ($+17.5$ points over 143K single-agent RL; -21.7 points when test-time `fork` is disabled on the same weights) and 51.2% on OpenResearcher, outperforming dense-retriever systems up to $250\times$ larger. These results establish latent-space multi-agent branching as an effective scaling axis orthogonal to model parameters and single-window length.

Limitations and Future Directions. Our current study trains a single homogeneous backbone over episode-scoped KV cursors within single-node GPU pools (up to 524 resident cursors/GPU at $W_{\max} = 10$, with tree-size loss normalization $1/Z_i$), opening three broader directions. First, **heterogeneous multi-model branching**: while our runtime already supports seven dense, MoE, and hybrid families (Appendix C), each episode currently executes one shared model; routing sub-branches across role-specialized LoRA adapters or heterogeneous model scales—such as lightweight scout branches paired with heavyweight verifiers via learned KV projections—could markedly improve compute–accuracy trade-offs. Second, **persistent LoRA and latent long-term memory**: mounted KV pages are currently reclaimed at episode end; consolidating verified branch KV states into persistent cross-episode memory banks or continual LoRA weights would extend Octopus from episodic reasoning to lifelong agentic memory. Third, **million-agent collective scaling**: tiering reference-counted radix cursors and recurrent checkpoints across GPU HBM, host DRAM/NVMe, and multi-node RDMA fabrics would scale latent-space coordination from ten-branch trees to asynchronous million-agent swarms.

References

[1] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Conference on Language Modeling (COLM)*, 2025.

[2] Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. BrowseComp-Plus: A fair and disentangled evaluation benchmark for deep search agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

[3] Zhuofeng Li, Dongfu Jiang, Xueguang Ma, Haoxiang Zhang, Ping Nie, Yuyu Zhang, Kai Zou, Jianwen Xie, Yu Zhang, and Wenhu Chen. OpenResearcher: A fully open pipeline for long-horizon deep research trajectory synthesis. *arXiv preprint arXiv:2603.20278*, 2026.

[4] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[5] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*, 2024.

[6] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

[7] Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, et al. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2026.

[8] Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. AReaL: A large-scale asynchronous reinforcement learning system for language reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[9] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics (TACL)*, 12:157–173, 2024.

[10] Tianyu Fu, Ruoxi Wang, Zihan Min, Zhongqi Ren, Zhihao Li, Tengxuan Liu, Shengxu Mu, Hanling Zhang, and Yu Wang. Cache engineering for agents. *arXiv preprint arXiv:2610.28577*, 2026.

[11] Federico Cassano and Sasha Rush. Training Composer for longer horizons. Cursor Research Blog, `https://cursor.com/blog/self-summarization`, 2026.

[12] Xinyu Wu et al. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025.

[13] Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. MEM1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations (ICLR)*, pages 58413–58438, 2026.

[14] Xiaochuan Li, Ryan Ming, Meng Chu, Shuai Shao, Rong Jin, and Chenyan Xiong. ACM: Agentic context management for long horizon tasks. *arXiv preprint arXiv:2607.23809*, 2026.

[15] Rulin Shao, Shannon Zejiang Shen, Junjie Oscar Yin, Yuetai Li, Minheng Wang, Hamish Ivison, Radha Poovendran, Nathan Lambert, Teng Xiao, Mike Lewis, Wen-tau Yih, Luke Zettlemoyer, and Pang Wei Koh. Context language models. *arXiv preprint arXiv:2609.37725*, 2026.

[16] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations (ICLR)*, 2025.

[17] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024.

[18] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[19] Arpandeep Khatua, Hao Zhu, Peter Tran, Arya Prabhudesai, Frederic Sadrieh, Johann K. Lieberwirth, Xinkai Yu, Yicheng Fu, Michael J. Ryan, Jiaxin Pei, and Diyi Yang. CooperBench: Why coding agents cannot be your teammates yet. In *International Conference on Learning Representations (ICLR)*, 2026.

[20] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion. In *Proceedings of the Twentieth European Conference on Computer Systems (EuroSys)*, pages 94–109, 2025.

[21] Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! Inference: Parallel LLM generation via concurrent attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[22] Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KVCOMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

[23] Hyesung Jeon, Hyeongju Ha, Seoyoung Lee, Beomseok Kang, and Jae-Joon Kim. KVCMAS: Efficient KV cache correction for shared context in multi-agent systems. *arXiv preprint arXiv:2609.34060*, 2026.

[24] Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. *arXiv preprint arXiv:2504.15466*, 2025.

[25] Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. Scaling long-horizon LLM agent via Context-Folding. In *International Conference on Machine Learning (ICML)*, 2026.

[26] Haoyang Hong, Jiajun Yin, Yuan Wang, Jingnan Liu, Zhe Chen, Ailing Yu, Ji Li, Zhiling Ye, Hansong Xiao, Yefei Chen, et al. Multi-agent deep research: Training multi-agent systems with M-GRPO. *arXiv preprint arXiv:2511.13288*, 2025.

[27] Kirill Afendulev, Alexey Dontsov, Elena Tutubalina, and Anton Korznikov. What attention recalls and recurrence controls in hybrid language models. *arXiv preprint arXiv:2609.04434*, 2026.

[28] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645, 2025.

[29] Song-Li Wu, Jingyi Wang, Zhaocheng Du, and Weinan Gan. ActiveMem: Dynamic latent memory trees for long-horizon agents. *arXiv preprint arXiv:2609.33244*, 2026.

[30] Tianjian Li, Jingyu Zhang, William Jurayj, Xi Wang, Chuanyang Jin, Mehrdad Farajtabar, Eric Nalisnick, and Daniel Khashabi. Self-compacting language model agents. *arXiv preprint arXiv:2606.23525*, 2026.

[31] Mo Li, L. H. Xu, Qitai Tan, Long Ma, Hongyong Song, Ting Cao, and Yunxin Liu. Sculptor: Empowering LLMs with cognitive agency via active context management. In *International Conference on Learning Representations (ICLR)*, pages 153411–153440, 2026.

[32] Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025.

[33] Vasilis Kontonis, Yuchen Zeng, Shivam Garg, Lingjiao Chen, Hao Tang, Ziyan Wang, Ahmed Awadallah, Eric Horvitz, John Langford, and Dimitris Papailiopoulos. Memento: Teaching LLMs to manage their own context. *arXiv preprint arXiv:2604.09852*, 2026.

[34] Xiangyu Shi, Marco Chiesa, Gerald Q. Maguire Jr., and Dejan Kostić. KVComm: Enabling efficient LLM communication through selective KV sharing. In *International Conference on Learning Representations (ICLR)*, 2026.

[35] 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 (MLSys)*, 2024.

[36] Junhao Hu, Wenrui Huang, Weidong Wang, Haoyi Wang, Tiancheng Hu, Qin Zhang, Hao Feng, Xusheng Chen, Yizhou Shan, and Tao Xie. EPIC: Efficient position-independent caching for serving large language models. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pages 24391–24402, 2025.

[37] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. DroidSpeak: KV cache sharing for cross-LLM communication and multi-LLM serving. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2026.

[38] 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.

[39] Ye Yu, Heming Liu, Haibo Jin, Xiaopeng Yuan, Peng Kuang, and Haohan Wang. Learning to communicate: Toward end-to-end optimization of multi-agent language systems. *arXiv preprint arXiv:2604.21794*, 2026.

[40] Jiarui Zhang, Yuchen Yang, Ran Yan, Zhiyu Mei, Liyuan Zhang, Daifeng Li, Wei Fu, Jiaxuan Gao, Shusheng Xu, Yi Wu, and Binhang Yuan. AREAL-DTA: Dynamic tree attention for efficient reinforcement learning of large language models. *arXiv preprint arXiv:2602.00482*, 2026.

[41] Weiqi Wang, Yuxin Zhou, Mouxiang Chen, Siyuan Zhang, Yi Zhang, Yuyan Luo, Zhiyu Yin, Chencan Wu, Jiemin Jiang, Wentao Yao, Chujie Zheng, and JianWei Zhang. QwenGyre: An elastic reinforcement learning framework for training xLong-horizon agents. *arXiv preprint arXiv:2609.33848*, 2026.

[42] Yuhan Sun, Jiaxin Zhao, Yongdong Luo, Yibo Hu, Meiguang Jin, Junfeng Ma, Weixun Wang, Weihang Pan, and Zhongan Bi. MultiAgent-OPD: End-to-end training of hierarchical multi-agent systems with on-policy distillation. *arXiv preprint arXiv:2610.23234*, 2026.

[43] Wouter Kool, Herke van Hoof, and Max Welling. Buy 4 REINFORCE samples, get a baseline for free! In *ICLR Workshop on Deep Reinforcement Learning Meets Structured Prediction*, 2019.

[44] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding R1-Zero-Like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

A Cross-Family Chat-Template Probing and Runtime Guardrails

Automatic Startup Chat-Template Probing and Tagged Injections. Open-weight chat templates $\mathcal{T}$ frequently violate strict append-only rendering: Qwen2.5/3/3.5 strip earlier assistant reasoning blocks upon a new user turn, DeepSeek-V3/R1 format initial and subsequent tool calls asymmetrically, Mistral-v0.3 embeds the system prompt inside the final user turn, and gpt-oss drops analysis channels on re-render. At server startup (~ 0.1 s), Octopus probes synthetic $(\text{parent-reply}, \text{delta-turn})$ pairs to locate the assistant-turn terminator τ_{term} and classify $\mathcal{T}$ into one of two incremental delta modes (Section 3.2). Templates with history-invariant suffixes use a *constant stub* that slices Render_{Δ} directly after τ_{term} (covering Qwen, Llama-3, DeepSeek-V3.1, and GLM-4.5/4.7), whereas context-sensitive templates render against a *minimal context stub* $\text{ctx}(c)$ before slicing (covering DeepSeek-V3/R1, Mistral-v0.3, and gpt-oss), yielding $0/18{,}344$ tokenization mismatches. Mid-episode runtime notifications (`fork`, join, and `rewind` notes) are injected as tagged tool responses rather than user turns so prior reasoning blocks remain intact.

Append-Only `read_thread` Injection, Answer Reserve, and Eviction Rebuild. To preserve 100% shared prefix KV caching while preventing pre-fork self-reads, Octopus omits `read_thread` from the static system prompt and appends its tool specification and sibling roster only after a readable peer step exists, clamping requested step ranges $[s, e]$ to the source's valid steps $[A, B]$ via $[\max(s, A), \min(e, B)]$. Within each thread's $B_{\text{thread}} = 32{,}768$-token growth budget, $R_{\text{ans}} = 2{,}048$ tokens are reserved for final synthesis, and tool-less plain-text replies trigger a single untrainable close reminder ($m_{\ell,t} = 0$). Completed cursors store compact mount-breakpoint tables in a server-side LRU registry, while mounted KV spans link into the GPU radix tree through negative placeholder token IDs and per-page reference counts. If GPU memory pressure or an online LoRA weight update evicts a mounted segment, the server aborts before token emission and executes a batched zero-decode prefill rebuild from the cursor's breakpoint table.

KV-Aware Episode Admission and Branch Turn Budget. Gating new episode admission on live server KV pool utilization ($\tau_{\text{KV}} = 0.75$ with zero waiting queue) prevents mid-episode fork bursts from saturating GPU memory (Section 4.3). Whereas unbounded dispatch drives P90 KV utilization to 98%–99% (72–88 queued requests) and causes 14.7%–46.1% episode timeouts, the 75% gate caps P90 KV load at 74.5% (0 queued, $0/1{,}304$ timeouts). This allows each 141GB GPU (1.94M-token KV pool) to host up to 524 resident GDN recurrent checkpoints and 154 concurrently decoding threads at $1{,}044.5$ tok/s (P90). In addition, raising the per-branch turn cap from $T_{\text{br}} = 20$ to $T_{\text{br}} = 50$ prevents rewound branches from exhausting their turn budget before calling `done`, narrowing the completion-rate gap between rewound and unrewound branches (ΔDone) from -36.0% (95% CI: $[-42.2\%, -29.4\%]$) to -4.1%.

Closing-Contract Guardrail Against Policy Collapse. We further compare three episode-closing contracts across $33{,}075$ pooled unlimited-depth rollouts ($D_{\max} = \infty, W_{\max} = 10$, combining our $24{,}041$-episode main run with two baseline-contract runs). Under *fallback plain-text grading*, scoring tool-less root replies via answer extraction causes unverified tool-less endings to surge from 6% to 75% by Step 12 (41% at $d \geq 3$). Conversely, under *strict `done`-only grading without a reminder*, the 7.5% of forked roots that emit plain text after a join note receive zero reward ($y = 0$), assigning negative advantage to forking ($\hat{A}_{\text{fork}} = -0.044$) and collapsing root fork rates from 23.9%–28.2% to 0% at Step 15 (1.2% at Step 17; Figure 10(b)). Injecting our single-turn untrainable close reminder ($m_{\ell,t} = 0$) converts 97.4% of reminded roots and 93.1% of reminded sub-branches into valid `done` calls (0.2% tool-less), sustaining 98.8%–99.2% root forking across the full 45-step training pass.

B Concrete Unlimited-Depth Search Rollout Walkthrough (`search-3190-99`)

To illustrate how Octopus's five execution primitives, layer-selective KV mounting, and sub-critical unlimited-depth tree execution work together on a long-horizon episode, Figure 11 and Table 7 trace a real evaluation episode (`search-3190-99`) on the multi-hop query: *"During a late twentieth-century volcanic event in the Luzon Volcanic Arc that was notable for lowering global temperatures for several years and having an ash cloud significantly larger than the 1980 eruption of a U.S. mountain, how many commercial airplanes experienced in-flight damage as a result of inadvertently entering the airborne particulate cloud generated by this eruption? Answer with a whole number only."*

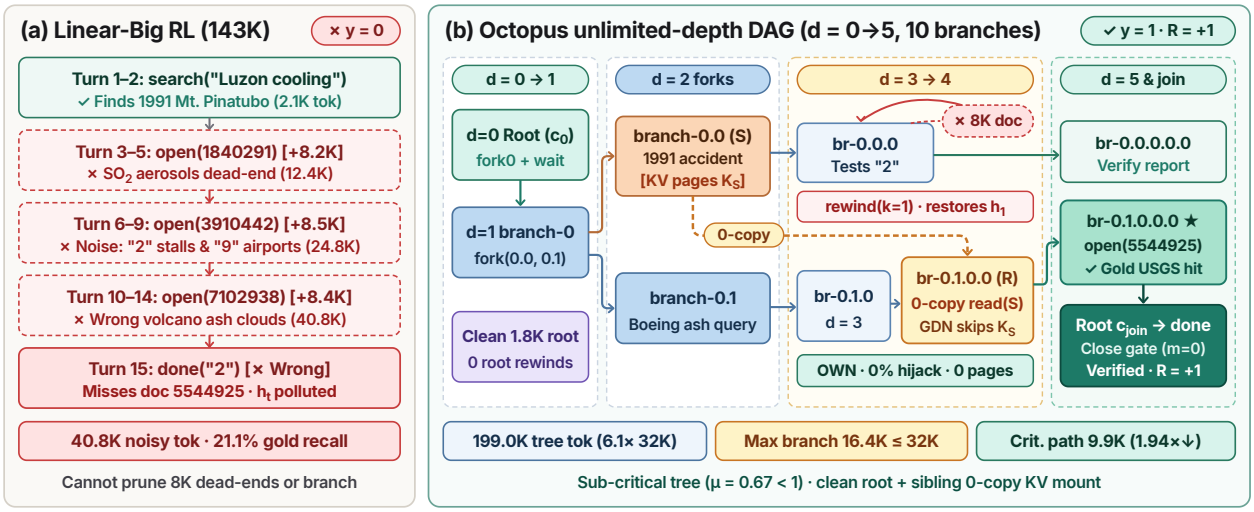


Figure 11: Multi-hop search rollout case study (`search-3190-99`: 1991 Mt. Pinatubo ash-cloud aircraft damage). **(a)** Flat 143K single-thread RL (32.1% BCP) accumulates noisy retrieval detours inside one append-only context, polluting its recurrent state $\mathbf{h}_t$. **(b)** Under $D_{\max} = \infty, W_{\max} = 10$, Octopus (49.6% BCP, 57.9% Rollout) keeps the root clean (1.8K tok) while spawning a depth-5, 10-branch DAG ($\mu = 0.67 < 1$): dead-end queries are pruned via branch `rewind`, and `branch-0.1.0.0` ($d = 4$) zero-copy mounts sibling `branch-0.0`'s attention KV pages (0% hijack) before opening `docid=5544925` at $d = 5$.

Table 7: Step-by-step trace and hybrid recurrent/attention state transitions for unlimited-depth rollout `search-3190-99` ($D_{\max} = \infty, W_{\max} = 10$, 10 branches across $d = 0 \to 5$).

Depth & Cursor	Agent Trajectory (Prompt / Thought / Tool Call & Return)	GPU Radix Cursor, Hybrid State & DAG-GRPO Semantics
$d = 0$ **Root** $c_0 \to$ `fork0` (Orchestrator)	**Thought:** Identify 1991 Mt. Pinatubo (Luzon Arc) and delegate commercial aircraft damage lookup. **Call:** `fork({"branch-0": "find number of commercial airplanes that experienced in-flight damage from Mount Pinatubo 1991 ash cloud"}) + wait(["branch-0"])`	✓ $\mathcal{O}(1)$ **Zero-Copy Spawn:** Increments Radix KV ref-count at c_{fork} by $+1$; clones GDN recurrent state $\mathbf{h}_{\text{end}}(c_0)$. ✓ **Clean Orchestrator:** Root suspends at 1.8K tok; `branch-0` gets fresh $B_{\text{thread}} = 32$K cap.
$d = 1 \to 2$ **Fork** `branch-0` $\to$ `0.0` (S), `0.1`	**Call** ($d = 1$)**:** `search("Mount Pinatubo 1991 ash cloud commercial airplanes damage")` $\to$ surfaces "more than 20 airplanes" and "24 windshields", then sub-forks ($d = 2$): • `branch-0.0` (S): `"search for Pinatubo 1991 aircraft accident airplanes"` • `branch-0.1`: `"search for \"Pinatubo\" Boeing airplanes ash"`	✓ **Recursive Expansion** ($D_{\max} = \infty$)**:** Each $d = 2$ child shares parent KV prefix in $\mathcal{O}(1)$ and inherits $\mathbf{h}_{\text{end}}$(`branch-0`). ✓ **Hypothesis Isolation:** Sibling S (`branch-0.0`) extracts candidate docID `5544925` and encounter tables into its KV pages.
$d = 3 \to 5$ **Left** `branch-0.0.0*` (Dead-End Path)	**Sub-Forks:** `0.0.0` ($d = 3$: `"...damage 2"`) $\to$ `0.0.0.0` ($d = 4$) $\to$ `0.0.0.0.0` ($d = 5$). **Dead End & Rewind:** Queries on "2 crashes" open noisy incident pages (+8.4K tok); branch calls `rewind(step=1, note="2 is engine replacements; pivot to USGS report")`.	✓ **Branch Rewind:** Cursor resets to c_1 (reclaiming +8.4K of B_{thread}); GDN layers restore exact $\mathbf{h}_1$ (erasing dead-end noise). ✓ **Single-Ownership (Section 3.4):** Dead-end leaf kept in DAG so `rewind` gets RL credit; post-rewind masks $m_{\ell,t} = 0$ ($t \le 1$).
$d = 3 \to 4$ **Right** `branch-0.1.0.0` (Reader R)	**Sub-Forks:** `branch-0.1.0` ($d = 3$: `"...\"airplanes\" 24 windshields"`) spawns $d = 4$ siblings `0.1.0.0` (R) and `0.1.0.1`. **Cross-Branch Mount:** Reader R (`branch-0.1.0.0`) invokes `read_thread(branch="branch-0.0", steps=[1, 3])` to inspect sibling S's retrieval table without re-searching.	✓ **Hijack-Free Zero-Copy KV Mount (Equation (2)):** • **Full-Attention Layers:** Borrow S's physical KV pages (0 new pages); R's next token jumps to $s(c'_R) = \max(p_R, e_S + 1)$. • **GDN Recurrent Layers:** Skip stepping over mounted span; $\mathbf{h}_R$ advances only along R's post-mount tokens (**retains own sub-task,** 0% **hijack**).
$d = 5$ **Leaf & Join** `0.1.0.0.0` $\to c_{\text{join}}$ (Verify & Close)	**Leaf Verification** ($d = 5$)**:** R spawns `branch-0.1.0.0.0` (`"open 5544925 offset 0"`), verifying the primary USGS/ICAO ash report, and closes via `done`. **Root Synthesis** ($d = 0$)**:** `wait` returns at c_{join}; runtime injects 1-turn Close Reminder ($m_{\ell,t} = 0$), and root calls `done` with exact whole-number answer.	✓ **Sub-Critical Tree Growth (Section 4.3):** Terminates at $W = 10$ branches $(1, 2, 2, 3, 2$ at $d = 1..5$; empirical $\mu = 0.67 < 1)$. ✓ **On-Policy Mount Retention (Equations (4) and (8)):** Keeps mounted $\mathbf{K}_S, \mathbf{V}_S$ in the 1D packed forward–backward graph.

C Per-Architecture Framework Integration Blueprints

Octopus supports dense, sparse-MoE, and hybrid recurrent-attention model families across both serving and 1D prefix-tree RL through a declarative architecture registry paired with automatic serving-engine LoRA target resolution (Table 8).

Pure Full-Attention Dense and Sparse-MoE Transformers (`Qwen3 Dense`, `Llama-3`, `Qwen3 MoE`). In models composed solely of full, sliding, or chunked attention with no recurrent state ($\mathbf{h}_t = \emptyset$), `fork` and `rewind` reduce to $\mathcal{O}(1)$ PagedAttention radix pointer updates. Because pure-attention models lack private recurrent layers to anchor branch identity, mounting sibling KV pages across all L layers can induce cross-branch persona drift; Octopus therefore mounts $c_A[s:e)$ under monotonic position offsets (Equation (2)) into only a sparse subset of middle retrieval layers ($\mathcal{L}_{\text{mount}} \subset \{1, \ldots, L\}$, e.g., $9/36$ layers in Qwen3-4B-Base) while presenting a mount-free KV view to the remaining 27 layers. This middle-layer mounting retains 96.9% argmax agreement with full-context replay during serving, while the 1D packed trainer dispatches separate variable-length FlashAttention passes for mounted and unmounted layer groups (Table 8). For sparse Mixture-of-Experts models (`Qwen3 MoE`), token-wise Top-K routing ($K = 8$) carries no cross-position state, so branching incurs zero expert-state copies and 1D prefix-tree deduplication batches active branches into a single grouped GEMM pass.

Hybrid Recurrent-Attention Architectures (`Qwen3.5 GDN` and `Granite-4.0-H Mamba-2`; Figure 12). In Qwen3.5 (8 blocks of three Gated DeltaNet layers and one Gated Full-Attention layer; Figure 12(a)), all 8 full-attention layers zero-copy mount sibling KV pages under Equation (2) (denoted Rule K2 in Figure 12) prior to the output sigmoid gate, whereas all 24 GDN layers clone or restore $\mathbf{h}_k$ on `fork`/`rewind` and skip mounted spans. During 1D packed DAG-GRPO training, tree-boundary hooks gather $d_{\text{conv}} - 1 = 3$ ancestor tokens for short causal convolutions and run level-by-level variable-length GDN recurrence via FlashLinearAttention (`fla`) initialized from each node's parent checkpoint $\mathbf{h}_{\text{end}}(p(u))$. Activation tiling in 16K-token chunks prevents backward memory spikes on trees spanning up to 262K tokens, while fused projection LoRA adapters (Table 8) align PEFT gradients directly with SGLang serving weights. Analogously, for Mamba-2 + GQA hybrids (`Granite-4.0-H`; Figure 12(b)), Octopus maps Mamba-2 Structured State Space Duality (SSD) onto `fla`'s chunked linear-attention kernels with ancestor convolution gathering and supports recursive transitive KV mounting ($X \rightsquigarrow A \rightsquigarrow C$, 98.5% token agreement).

Table 8: Per-architecture integration hooks across model families in `octopus/rl/model_hooks.py`, `lora_targets.py`, `seq_tiles.py`, and `patches/apply_sglang_*.py`. Unregistered stateful architectures raise an explicit startup guard.

Architecture Family	Decoder Layer Topology	Serving KV Mount & State Hook	Prefix-Tree Recurrence, Conv & Tiling Hook	SGLang-Resolved LoRA Targets (`lora_targets.py`)
`Qwen3 (Dense)` (`qwen3`, `llama3`)	L Full-Attn + SwiGLU MLP ($\mathcal{L}_{\text{rec}} = \emptyset$)	`OCTOPUS_MOUNT_LAYERS` (9/36 layers mount $c_A[s:e)$; 27 call `hide_mounts`)	`CONV_CONTEXT` $= 0$; varlen FlashAttn `attn_calls` vs. `own_calls` (32K chunks)	7 modules: `q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`
`Qwen3 (MoE)` (`qwen3_moe`)	L Full-Attn + Top-K Sparse MoE ($K = 8$)	Attention-only Radix KV mount + Equation (2) jump; 0 MoE state on `fork/rewind`	`CONV_CONTEXT` $= 0$; 1D deduplicated tree tokens T_{tree} batched into MoE grouped GEMM	Attention LoRA (`q/k/v/o_proj`); preserves Top-K router gate stability
`Qwen3.5 (Dense)` (`qwen3_5`, Primary)	$8 \times [3$ GDN$+1$ Gated Attn$]$ (24 GDN, 8 Attn) + SwiGLU MLP	All 8 Full-Attn layers 0-copy mount $c_A[s:e)$; 24 GDN layers clone/restore $\mathbf{h}_k$ & skip mounts	`CONV_CONTEXT` $= d_{\text{conv}} - 1 = 3$ (`tree_conv`); `fla chunk_gated_delta_rule`; `TILE=16K` (`seq_tiles.py`)	10 modules: `q/k/v/o_proj, gate/up/down_proj, out_proj` + fused `in_proj_qkvz`
`Qwen3.5-MoE / Next` (`qwen3_5_moe`, `qwen3_next`)	24 GDN + 8 Gated Attn + Sparse Top-K MoE (+ Shared Expert)	8 Full-Attn layers mount $c_A[s:e)$; 24 GDN layers skip mounted spans; 0 MoE state overhead	`CONV_CONTEXT` $= 3$ (`tree_conv`) + level-by-level `tree_recurrence` (32K segs); HF MoE forward	Attention + GDN projections (`q/k/v/o_proj, in_proj_qkvz, out_proj`)
`Granite-4.0-H` (`granitemoehybrid`)	Mamba-2 SSD Mixer + GQA Full-Attn + Sparse MoE & Shared MLP	Transitive $X \rightsquigarrow A \rightsquigarrow C$ Attn KV mount (98.5% agree); Mamba-2 state $\mathbf{h}_k$ skips mounts; $1\times$ final norm patch	`CONV_CONTEXT` $=$ `mamba_d_conv` $- 1$ (`tree_conv`); SSD mapped to `fla chunk_simple_gla`	5 modules: `q_proj, k_proj, v_proj, o_proj` + patched Mamba `out_proj` ($H \cdot P \to D$)

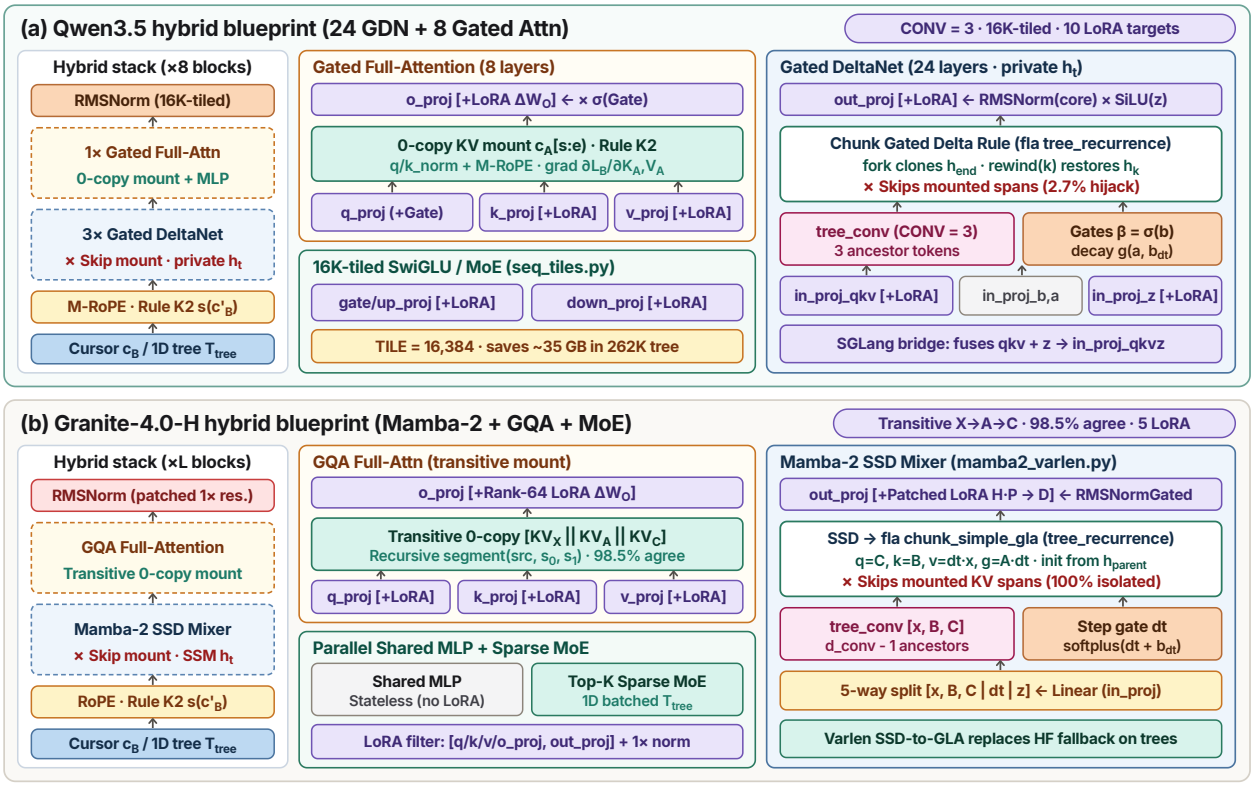


Figure 12: Octopus integration blueprints for hybrid recurrent-attention models (Table 8). **(a) Qwen3.5 (Dense & MoE Hybrid):** All 8 Gated Attention layers 0-copy mount sibling KV pages $c_A[s:e)$, while 24 GDN layers gather `CONV_CONTEXT` $= 3$ ancestor tokens (`tree_conv`) and advance $\mathbf{h}_t$ along branch tokens via `fla`'s `chunk_gated_delta_rule` (`tree_recurrence`, skipping mounts $\to$ 2.7% hijack), paired with 16K tiling (`seq_tiles.py`) and fused `in_proj_qkvz` LoRA. **(b) Granite-4.0-H (Mamba-2 Hybrid):** Maps Mamba-2 SSD to `fla chunk_simple_gla`, supports transitive $X \rightsquigarrow A \rightsquigarrow C$ mounts (98.5% agree), and patches Mamba `out_proj` LoRA and final RMSNorm.