NEVER OPTIMIZE YOUR MEMORY-COMPRESSION HARNESS UNDER TRAINING–INFERENCE MISMATCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Production agent harnesses, including Claude Code and Qwen-Agent, **compress an agent's context** during rollout to make long-horizon interaction tractable. Training policies under such compression, however, introduces a fundamental conditioning problem: ***each compression-induced eviction branches the effective interaction history, making the object presented to the learning objective a tree rather than a single trajectory sequence***. Existing learning pipelines typically linearize this tree in one of two ways: ❶ retaining only the *rightmost root-to-leaf path*, which causes **time-travel leakage**, or ❷ replaying the full *depth-first traversal*, which induces a **train–inference mismatch**. Thus, we introduce two exact conditioning-level corrections: ❶ **LogitTree**, a segmented K-forward traversal of the trajectory tree, and ❷ an equivalent packed **Logit attention mask**. LogitTree requires $K+1$ backward passes for every root-to-leaf, whereas the Logit formulation requires both a custom masked-attention kernel and white-box access to harness eviction records. To avoid these costs, we further propose **SDCC** (**S**elf-**D**istillation for **C**onditioning **C**onsistency), a training-friendly variational relaxation requiring only a single backward pass and multiple parallel forward passes. At each eviction junction, SDCC minimizes the forward KL divergence from a stop-gradient teacher distribution conditioned on the reconstructed pre-eviction prefix to the compressed-context student policy. By Pinsker's inequality, a residual per-junction KL divergence of $\varepsilon_{\mathrm{KL}}$ yields an $\mathcal{O}(\sqrt{\varepsilon_{\mathrm{KL}}})$ bound on the resulting train–deployment behavioral gap in total variation. In black-box harnesses, SDCC uses captured per-turn request payloads and requires no span-level eviction logs. Experiments across three white-box editors, two black-box harnesses, and seven web-search benchmarks reveal substantial conditioning drift under naive replay. LogitTree and Logit masking recover the no-compression numerical baseline, while SDCC substantially narrows the gap with a modest computational budget. On the WideSearch benchmark, our LogitTree method improves the Item F1 of Qwen3.7-Air (125B-A6B) from 67.32 to 74.03, surpassing all reference models in our evaluation.

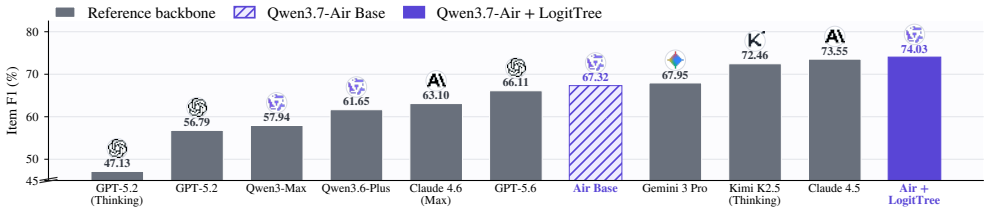

Figure 1: **WideSearch Item F1 across backbones.**

1 INTRODUCTION

Long-horizon agents, including deep-search assistants (Xi et al., 2025), coding copilots (Wang et al., 2025), browser-based research agents (Shi et al., 2025), can accumulate trajectories spanning tens to hundreds of thousands of tokens. Retaining the full interaction history is not only computationally expensive but can also impair decision making: ❶ outdated observations, abandoned plans, and failed attempts introduce substantial contextual noise (Jiang et al., 2025); ❷ state rollback can invalidate the actions and observations associated with a reverted branch (Jiang et al., 2025); and ❸ critical

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.

goals and instructions may be diluted or overlooked as the context grows, exacerbating the *lost-in-the-middle* effect (Liu et al., 2023; Chen et al., 2026). **Production agent harnesses therefore actively compress and rewrite interaction histories to maintain compact, decision-relevant contexts**. For example, Claude Code (Anthropic, 2024) and Qwen-Agent (Team, 2024) maintain bounded context windows and replace evicted prefixes with automatically generated summaries. Other approaches, including TC-RAG (Jiang et al., 2025), MemexRL (Wang et al., 2026), and AgentFold (Ye et al., 2025; Zhou et al., 2025), introduce model-aware memory mechanisms that allow agents to manage what information to compress, offload, or retrieve during interaction.

Across these context-compression designs, a ***memory editor*** (hook) rewrites the interaction history to maintain a bounded live context. Although this can improve inference-time tractability and decision quality, it creates a subtle problem for post-training (SFT and RL) when learning these summary behaviors: ***when the trainer recomputes the log probabilities of rollout tokens, the context used for recomputation may differ from the context under which each token was originally generated***.

Training on the conditioning tree, not the sequence. This conditioning mismatch arises because harness-level compression breaks the correspondence between token order and conditioning history. At each eviction junction, the original prefix is replaced by a compressed view: tokens preceding the edit were generated under the pre-edit context, whereas subsequent tokens are generated under its compressed replacement. Repeated edits therefore induce a *tree of conditioning histories*, rather than a single sequence that can be replayed consistently, as illustrated in Figure 2(a). Existing pipelines (Ye et al., 2025; Zhou et al., 2025; Wang et al., 2026) typically serialize this tree in one of two ways: ❶ Training on surviving compressed stream follows only the ***rightmost root-to-leaf path***, causing earlier tokens to be reevaluated under contexts that retroactively incorporate later deletions or summaries—an error we call ***time-travel leakage***. ❷ Training on full raw trace follows a ***depth-first traversal***, causing later tokens to be reevaluated with no longer available evicted content when they were generated—a ***train–inference conditioning mismatch***. **In both cases, logits are recomputed under contexts different from those at generation, causing train–inference logit inconsistency.**

From the conditioning tree to exact solutions. To restore the context under which each rollout token was originally generated, we formulate training as a traversal of the conditioning tree and introduce two exact, gradient-equivalent constructions. ❶ **LogitTree** performs a segmented traversal, separately evaluating the $K{+}1$ branches induced by K evictions; ❷ a packed **Logit attention mask** encodes the same branch structure within a single forward pass. Under the equivalence conditions established in §3, both constructions yield the same policy gradients and exactly recover the rollout conditioning. This exactness, however, comes at a computational or systems cost: LogitTree requires $K{+}1$ forward and backward passes per rollout, whereas the packed formulation relies on a custom masked-attention kernel that is difficult to integrate efficiently with fused attention implementations and requires white-box access to harness eviction records. ❸ To reduce this cost, we propose **SDCC** (**S**elf-**D**istillation for **C**onditioning **C**onsistency; §4). At each eviction junction, SDCC distills behavior under the reconstructed pre-eviction prefix into the compressed-context policy by minimizing teacher-to-student forward KL. Given a residual KL divergence of $\varepsilon_{\mathrm{KL}}$ at an aligned token, Pinsker's inequality bounds the corresponding distribution gap by $\mathcal{O}(\sqrt{\varepsilon_{\mathrm{KL}}})$ in total variation. SDCC combines a single student backward pass with multiple gradient-free teacher forward passes that can be parallelized. For black-box harnesses, it requires only captured per-turn request payloads, without span-level eviction logs or modified attention kernels. Our contributions are fourfold:

- **We formalize training under context compression as a conditioning-inconsistency problem** (§2). We characterize how two common replay strategies—*Naive-Compressed* and *Naive-Full*—lead to time-travel leakage and train–inference conditioning mismatch, respectively.
- **We propose two exact, gradient-equivalent constructions for traversing the conditioning tree** (§3). **LogitTree** explicitly evaluates its branches, while a packed **Logit attention mask** realizes equivalent replay in a single forward pass under the stated equivalence conditions.
- **We introduce SDCC, a training-efficient relaxation** (§4). SDCC distills policy behavior under the original pre-eviction context into the surviving compressed context, requires only one backward pass, and supports black-box harnesses through captured per-turn request payloads.
- Across three white-box and two black-box harnesses, and seven web-search benchmarks, our methods reduce conditioning drift and greatly improve downstream accuracy (§5). We further demonstrate effectiveness on a large-scale backbone for WideSearch, LogitTree improves the Item F1 of Qwen3.7-Air (125B-A6B) from 67.32 to 74.03.

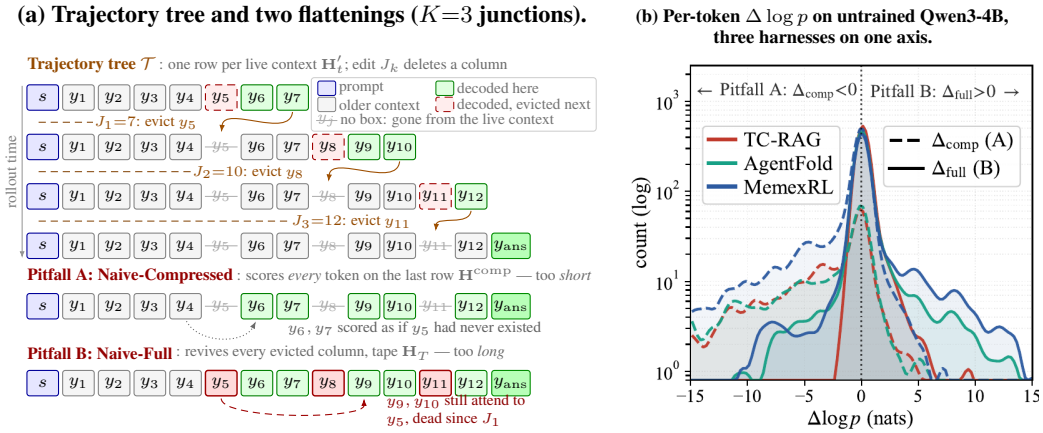

Figure 2: **The two pitfalls and Logits Drift.** **(a)** Each eviction forks the live-context staircase. Naive-Compressed scores tokens on the final spine $\mathbf{H}^{\text{comp}}$; Naive-Full replays $\mathbf{H}_T$. **(b)** The resulting per-token logit differences on an untrained Qwen3-4B have opposite signs across all three white-box harnesses: Δ_{comp} is predominantly negative, while Δ_{full} is predominantly positive. Full statistics are in Table 7 (§G.2).

2 Pitfalls: Two Default Ways to Train on an Edited Rollout

Every harness in §A rewrites the agent's context mid-rollout, posing one question: ***which context did each token condition on, and does training match it?*** An edited rollout is therefore a *trajectory tree*, not a single sequence that can be replayed consistently. Existing pipelines flatten it in one of two ways, which fail in opposite directions yet violate the same token-level conditioning invariant.

2.1 Train–Inference Logits Drift in Memory Compressing Harness

❶ ***Train–Inference Logit Drift.*** To quantify conditioning mismatch, we compare the log probability assigned to the same token during rollout and training. For a generated token y_t, let c_t^{rollout} denote the context from which it was generated, and let c_t^{train} denote the context used to score it during training. The rollout engine records $\log P_\theta(y_t \mid c_t^{\text{rollout}})$ at generation time; the trainer then recomputes $\log P_\theta(y_t \mid c_t^{\text{train}})$ using the *same model weights*. We define **train–inference logit drift** (`logdiff` in §5) as the mean absolute difference over response-token positions $\mathcal{M}$ included in the loss:

$$\text{logdiff} \;=\; \frac{1}{|\mathcal{M}|} \sum_{t \in \mathcal{M}} \left| \log P_\theta(y_t \mid c_t^{\text{train}}) - \log P_\theta(y_t \mid c_t^{\text{rollout}}) \right|. \tag{1}$$

If the two contexts also match, their log probabilities should agree up to numerical differences between the training and inference implementations (Zhang et al., 2026b). Our no-compression control measures this numerical baseline at approximately 0.014 (§5). We therefore use this baseline as a reference: drift substantially above it indicates disagreement beyond ordinary numerical variation and, in our controlled comparison, provides evidence of conditioning mismatch.

❶ ***Setup: the live view and the trajectory tree*** We model the agent as a ReAct-style policy (Yao et al., 2023). We use r for interaction turns and reserve t for generated-token positions. At the start of turn r, the physical history is $\mathbf{H}_r = (s_0, a_0, o_0, \ldots, s_{r-1}, a_{r-1}, o_{r-1})$, where s_k, a_k, and o_k denote the thought, tool call, and tool observation at turn k. A memory-editing harness inserts an editor $\textsc{Edit}(\cdot)$ into the rollout loop to perform deletion (Wang et al., 2026) or summarization (Ye et al., 2025). For a token y_t emitted at turn $r(t)$, let $\mathbf{H}_{<t}$ denote the rendered physical history preceding that token, including initial prompt, prior interactions, and any tokens already generated in the current turn. Let $\mathcal{E}_{\le r(t)}$ collect the edits applied before y_t is decoded. Its **live context view** is:

$$\mathbf{H}'_t \;=\; \textsc{View}_{r(t)}\left(\mathbf{H}_{\le r(t)};\, \mathcal{E}_{\le r(t)}\right), \qquad \mathcal{E}_{\le r(t)} = \textsc{Edit}(\mathbf{H}_{\le r(t)}), \tag{2}$$

the partially observable prefix from which y_t is decoded. At the end of the rollout, the harness reports the *final compressed walk* $\mathbf{H}^{\text{comp}}$. Crucially, the right-hand side incorporates edits that had not yet occurred when y_t was generated:

$$\underbrace{\mathbf{H}'_t}_{\text{generation-time context}} \;=\; \underbrace{\textsc{View}_{r(t)}\left(\mathbf{H}_{\le r(t)};\, \mathcal{E}_{\le r(t)}\right)}_{\text{live view at decode of } y_t} \;\neq\; \underbrace{\mathbf{H}^{\text{comp}}[:t]}_{\text{prefix of final walk}}. \tag{3}$$

To see how this mismatch arises, consider an eviction at token position J_k that removes or replaces a span E_k. Tokens generated while that span was visible retain the pre-edit conditioning history, whereas subsequent generation proceeds from the edited context. Each edit therefore separates a *generation-time leg*, which preserves the earlier conditioning history, from a *compressed spine*, along which the rollout continues. Unrolling K such evictions at positions $J_1 < \cdots < J_K$ yields a *trajectory tree* $\mathcal{T}$ rooted at the initial context. We represent each generated target y_t by a leaf whose root-to-leaf conditioning path is $\mathbf{p_t}(y_t) = \mathbf{H}'_t$. For targets that survive in the final compressed walk, $\mathbf{p_s}(y_t) = \mathbf{H}^{\text{comp}}[:t]$ denotes corresponding final-walk prefix. Figure 2(a) illustrates a running example with $E_1 = \{y_5\}$, $E_2 = \{y_8\}$, and $E_3 = \{y_{11}\}$, evicted immediately after positions $J_1 = 7$, $J_2 = 10$, and $J_3 = 12$, respectively. Corresponding per-leaf prefixes are in Table 4 and Appendix C.4.

2.2 The two pitfalls and the conditioning invariant

Two natural replay strategies flatten $\mathcal{T}$ into a sequence: ❶ the final compressed walk $\mathbf{H}^{\text{comp}}$, which retains only the compressed spine, and ❷ the full physical trace $\mathbf{H}_T$, which retains the chronological interaction history, including evicted content:

❶ ***Pitfall A: Naive-Compressed follows the final compressed path (time-travel leakage).*** Naive-Compressed trains on the rightmost root-to-leaf path of $\mathcal{T}$, namely the final compressed walk: $\mathbf{H}^{\text{comp}} = \textsc{RootToRightmostLeaf}(\mathcal{T})$. When a later edit changes the prefix of an earlier target y_t, training scores that target under a context different from the one that generated it:

$$\underbrace{c_t^{\text{train}} = \mathbf{p_s}(y_t)}_{\text{prefix of } \mathbf{H}^{\text{comp}}} \neq \underbrace{c_t^{\text{rollout}} = \mathbf{p_t}(y_t)}_{\text{generation-time context}}. \tag{4}$$

Here, $\mathbf{p_s}(y_t)$ denotes the prefix preceding the aligned occurrence of y_t in $\mathbf{H}^{\text{comp}}$, while $\mathbf{p_t}(y_t)$ denotes its original live context. Later deletions or summaries are thus applied retroactively, a failure mode we call ***time-travel leakage***. In the running example, y_6 and y_7 are scored without y_5, although both were generated with y_5 in context. For deletion edits, final-walk prefix is a subsequence of the original context; summarization may additionally introduce text that did not exist at generation time.

❷ ***Pitfall B: Naive-Full trains on the depth-first traversal (train–inference mismatch).*** Naive-Full instead replays the full physical trace, obtained by a depth-first serialization of $\mathcal{T}$: $\mathbf{H}_T = \textsc{DepthFirstTraversal}(\mathcal{T})$. Let $\mathbf{H}_{<t}$ denote the prefix preceding y_t in $\mathbf{H}_T$. When this prefix retains already-evicted content, training again uses a context different from that generated the target:

$$\underbrace{c_t^{\text{train}} = \mathbf{H}_{<t}}_{\text{prefix of } \mathbf{H}_T} \neq \underbrace{c_t^{\text{rollout}} = \mathbf{p_t}(y_t)}_{\text{generation-time context}}. \tag{5}$$

In the running example, y_9 and y_{10} are scored with y_5 still present, although it was evicted after y_7. Likewise, y_{12} is scored with both y_5 and y_8, although neither remains in its live context. We call this *stale-context leakage*: training can reward reliance on information unavailable at deployment.

To illustrate the resulting probability shifts, the signed per-token `logdiff` in untrained Qwen3-4B includes discrepancies of $\Delta_{\text{comp}} = -22.8$ nats and $\Delta_{\text{full}} = +18.5$ nats in Figure 2(b). These observed signs illustrate the two failure modes; they are not guaranteed by context inclusion alone. The formal statement, mirror symmetry and its SFT/RL implications appear in Appendix C.4.

3 Exact Solutions: Two Ways to Walk the Context Tree

To eliminate the conditioning errors introduced by the two pitfalls, we propose two exact constructions that score each target under its generation-time context. ❶ **LogitTree** evaluates the branches of $\mathcal{T}$ as separate sequences; ❷ the **Logit attention mask** packs these branches into a single forward pass. Both preserve the same conditioning histories and token-level loss assignments. They yield identical logits and gradients (in Theorem 1), eliminating bias due to conditioning mismatch (Figure 3).

❶ ***LogitTree.*** **LogitTree** extends token-level TITO (Gallouédec & Rasul, 2026) to trajectory trees by separately evaluating the $K+1$ branches from root to leaf, induced by K evictions. Here, $\mathbf{H}'^{\pi}$ is the token sequence on branch π, and f_θ returns its next-token logits as:

$$\{\mathbf{H}'^{\pi}\}_{\pi=1}^{K+1} = \textsc{RootToLeaf}(\mathcal{T}), \qquad \mathbf{Logits}_\pi = f_\theta(\mathbf{H}'^{\pi}), \qquad \pi \in \{1, \ldots, K+1\}. \tag{6}$$

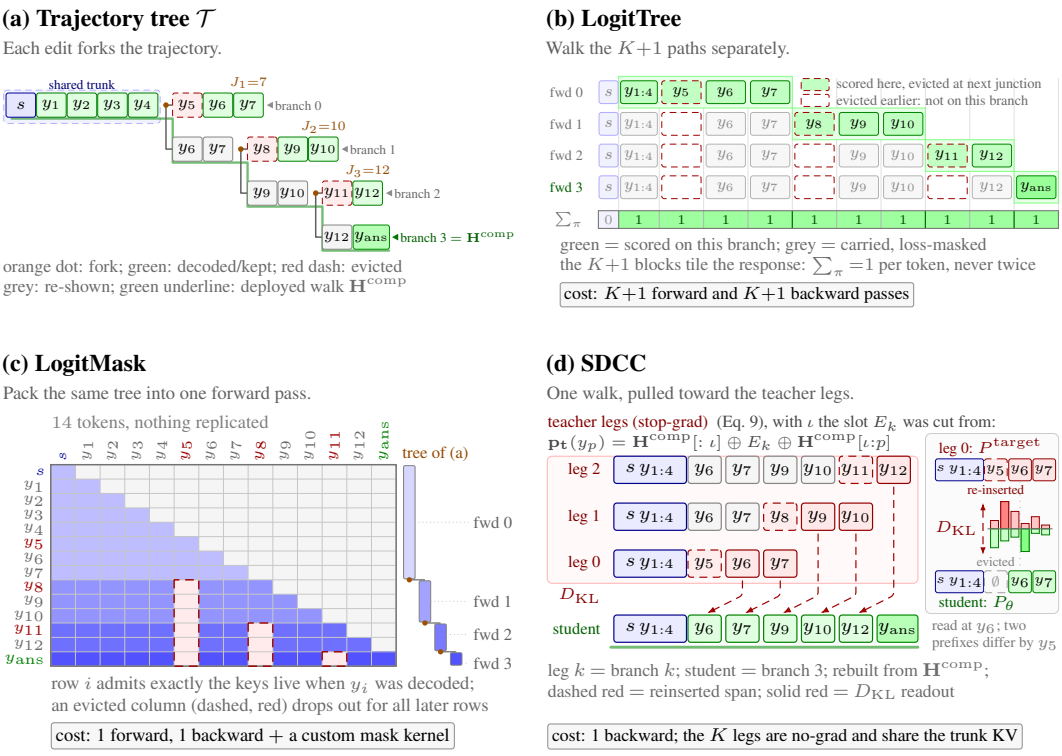

Figure 3: **Three conditioning-consistent views of the context tree.** **(a)** Each eviction creates another live root-to-leaf branch. **(b)** LogitTree evaluates the K+1 branches separately with disjoint loss masks. **(c)** A packed Logit mask realizes the same traversal in one forward pass and yields identical gradients. **(d)** SDCC keeps one backward pass and aligns compressed student with stop-gradient live-context teachers only at diverging leaves

Each target y_t is scored on one branch that reproduces its live context $\mathbf{p_t}(y_t) = \mathbf{H}'_t$. A binary loss mask $m_t^\pi \in \{0,1\}$ with $\sum_{\pi=1}^{K+1} m_t^\pi = 1$ ensures that each token contributes exactly once, while replicated tokens remain available as context. In Figure 3(b), the branch ending at $J_1 = 7$ retains y_5 when scoring y_6, y_7, whereas final branch scores y_{ans} without the evicted y_5, y_8, y_{11}.

LogitTree uses K+1 forward and backward passes per rollout, with cost determined by branch lengths. Exact replay requires generation-time live views with matching tokenization and positional encoding. SFT/RL equivalence and implementation details appear in Appendix D (Proposition 2) and Appendix F.2, respectively.

❷ ***The Logit attention mask.*** Moreover, the entire conditioning tree can also be traversed in a single masked forward pass. Following the Logit attention-mask formulation in (Jiang et al., 2026; Zhou et al., 2025), we pack $\mathbf{H}_T$ into a single sequence while allowing each query token to attend only to **the context tokens visible when it was generated**. Let V_i be the set of packed token positions in the live view of query position i. This row-wise set allows a token to leave and later re-enter the context, as in an offload–retrieve editor. We define

$$M_{i,j}^{\text{logical}} = \begin{cases} 0, & j \in V_i, \\ -\infty, & \text{otherwise}, \end{cases} \qquad M_{i,j} = M_{i,j}^{\text{causal}} \oplus M_{i,j}^{\text{logical}}, \tag{7}$$

The causal mask M^{causal} prevents attention to future tokens, while M^{logical} further removes attention edges between different branches of the conditioning tree (those are not visible). We also reassign position ids so that the tokens visible to each query form a gap-free logical sequence (Zhou et al., 2025). As a result, each query y_i attends to exactly the root-to-leaf path of $\mathcal{T}$ under which it was generated, allowing all branches to be evaluated in a single forward pass (Figure 3(c)).

Throughout, both materializations share **one standing conditioning invariant** proposition — and gradient equivalence (Proposition 3, App. D), which holds under five conditions: (i) exact-visibility encoding, (ii) row-wise position reassignment, (iii) decoupled loss/attention masks, (iv) no cross-mask

normalization, and (v) rollout and training sharing the same M^{logical}. However, linear attention, sparse/top-k selection, and vLLM/SGLang violate the premise or one of (i)–(v) (App. F.2); formal equivalence and deployment costs appear in App. D.3.

4 SDCC: Self-Distillation for Conditioning Consistency

The two exact methods in Section 3 recover generation-time conditioning, but incur practical costs: LogitMask requires specialized attention support, complicating integration with existing training systems. LogitTree requires $K+1$ backward passes per rollout, making training increasingly expensive as the number of branches grows. Yet our goal is to train a policy that acts **effectively** under large compressed behaviors. This raises a question: ***can we retain the supervision provided by the original context branches without backpropagating through each of them?*** We propose SDCC, which uses predictions from these branches as distillation targets for the compressed-context policy. The original branches require only gradient-free parallel forward passes, while all gradient updates flow through a single student pass on $\mathbf{H}^{\text{comp}}$.

Two contexts, one policy. For a response token y_p retained in the final compressed walk, define

$$z_p := \mathbf{p_t}(y_p), \qquad x_p := \mathbf{p_s}(y_p), \qquad \mathcal{C} := \{p : z_p \neq x_p\}. \tag{8}$$

Here, z_p is the live prefix from which y_p was generated, x_p is its prefix in $\mathbf{H}^{\text{comp}}$, and $\mathcal{C}$ indexes the *diverging leaves*. In Figure 3(d), y_5 is evicted after y_6 is generated, so z_6 contains y_5 whereas x_6 does not. Exact replay scores y_6 under z_6. SDCC instead retains the task loss under x_6 and uses the prediction under z_6 to guide the update. Specifically, in SDCC, the policy evaluated on z_p serves as a stop-gradient *teacher*, while the policy evaluated on x_p serves as the gradient-carrying *student*.

Recovering teacher supervision. The student performs the ordinary task forward pass on $\mathbf{H}^{\text{comp}}$. Teacher predictions are obtained by evaluating the corresponding live prefixes without gradients. For a deletion-only edit, suppose the two prefixes differ by a single span $E_{\text{act}}(p)$ that was visible when y_p was generated but subsequently removed. Let $\iota(E_{\text{act}}(p))$ denote its reinsertion position in the compressed prefix. The live prefix is reconstructed as

$$\mathbf{p_t}(y_p) = \underbrace{\mathbf{H}^{\text{comp}}[:\iota(E_{\text{act}}(p))]}_{\text{shared trunk}} \oplus \underbrace{E_{\text{act}}(p)}_{\text{re-inserted span}} \oplus \underbrace{\mathbf{H}^{\text{comp}}[\iota(E_{\text{act}}(p)) : p]}_{\text{retained suffix}}. \tag{9}$$

Here, $\oplus$ denotes sequence concatenation, and p in the slice notation refers to the aligned position of y_p in $\mathbf{H}^{\text{comp}}$. Reconstruction restores the context visible at generation, rather than indiscriminately reinserting all evicted content. Multiple edits require restoring the appropriate spans and undoing any corresponding summary replacements; the general construction is given in Appendix E.1. For black-box harnesses, the live prefixes can instead be recovered from captured per-turn request payloads and the corresponding response prefixes, without span-level eviction logs.

Distilling conditioning consistency. SDCC add KL penalty only at diverging leaves:

$$\mathcal{L}_{\text{SDCC}} = \underbrace{\mathcal{L}_{\text{task}}(\theta; \mathbf{H}^{\text{comp}})}_{\text{compressed-stream task loss}} + \lambda \sum_{p \in \mathcal{C}} D_{\text{KL}} \left(\underbrace{P^{\text{target}}(\cdot \mid z_p)}_{\text{live-context teacher; stop-grad}} \,\middle\|\, \underbrace{P_\theta(\cdot \mid x_p)}_{\text{compressed-context student}} \right). \tag{10}$$

The teacher distribution is $P^{\text{target}}(\cdot \mid z_p) := \text{sg}[P_\theta(\cdot \mid z_p)]$, where sg denotes stop-gradient. The student distribution $P_\theta(\cdot \mid x_p)$ is read from the corresponding prediction position in the forward pass on $\mathbf{H}^{\text{comp}}$. The KL is ***forward***: the teacher distribution provides a fixed soft target, encouraging the student to preserve continuations supported by the original live context. With the teacher detached, minimizing this term is equivalent to minimizing cross-entropy against its next-token distribution. This penalty is ***leaf-gated***: If no target context differs, $\mathcal{C}$ is empty and SDCC reduces to the ordinary task loss. Setting $\lambda = 0$ recovers Naive-Compressed exactly.

Appendix E derives the details of SDCC's KL direction. And this residual KL also controls behavior: a per-leaf residual ε_p bounds the teacher–student next-action discrepancy by $\text{TV}(P^{\text{target}}(\cdot \mid z_p), P_\theta(\cdot \mid x_p)) \leq \sqrt{\frac{\varepsilon_p}{2}}$ in total variation. The formal guarantee, its proof, and the resulting policy-gradient-bias bound are deferred to App. E.4.

5 EXPERIMENTS

We validate the proposed conditioning-inconsistency framework across multiple model scales, white-box and black-box agent harnesses, and multiple logit recomputation schemes.

5.1 EXPERIMENTAL SETUP

We validate conditioning inconsistency across different backbone models, agent harnesses, and training-time logit recomputation schemes.

❶ **Backbone Models.** We use Qwen3-4B and Qwen3.7-Air (125B-A6B) as backbone models.

❷ **Harnesses.** For white-box context editors, we use three representative learnable-action editors: **TC-RAG** (Jiang et al., 2025), which emits `pop()`; **AgentFold** (Ye et al., 2025), which folds spans past a length threshold; and **MemexRL** (Wang et al., 2026), which offloads spans to an external store. A **Search-R1** no-compression control emits tool calls but never evicts, anchoring the no-compression drift floor. Knobs and per-method inputs are given in §G.1–G.3. For black-box production-style harnesses, we instrument their HTTPS traffic, recover the per-turn rendered context prefixes, and construct the corresponding LogitTree views from these observed prefix states. We evaluate two such harnesses: **Claude Code** and **OpenCode**.

❸ **Logit Recomputation Schemes.** We compare replay on the final compressed stream (**Naive-Compressed**), replay on the full physical trace (**Naive-Full**), two exact conditioning-consistent replay methods (**Logit-Mask** and **LogitTree**), and our proposed single-backward approximation **SDCC**.

❹ **Datasets.** *Train:* a pooled composite-QA corpus of $81{,}638$ instances drawn from RED-SEARCHER (Chu et al., 2026) and the ASEARCHER agentic-search training set, whose multi-hop structure reliably induces long-horizon interaction and frequent context editing. *Test:* the same live pipeline evaluates $38{,}270$ questions per checkpoint across NQ (Kwiatkowski et al., 2019), TRIVIAQA (Joshi et al., 2017), HOTPOTQA (Yang et al., 2018), 2WIKI (Ho et al., 2020), MUSIQUE (Trivedi et al., 2022), BAMBOOGLE (Press et al., 2023), and FRAMES (Krishna et al., 2025).

❺ **Search Infrastructure.** All rollouts use live web tools rather than a local retrieval server. Web search is backed by the online DashScope text-search API, which returns up to 10 ranked results per query. For full-page evidence extraction, we use a two-stage pipeline: retrieved pages are first fetched via Firecrawl and then summarized by Qwen3-Turbo.

❻ **Evaluation Metrics.** We report the following quantities, each with a distinct role. (1) **Recorded-token `logdiff`** $= \left|\log \pi_{\text{train}}(y|c^{\text{train}}) - \log \pi_{\text{rollout}}(y|c^{\text{rollout}})\right|$ is a per-token conditioning-fidelity diagnostic rather than an efficacy metric; (2) **SDCC training KL** KL_{SDCC} (§5.2, §G.6) is SDCC's leaf-gated consistency regularizer, and is not intended as a cross-method comparator. (3) **Rollout reward** measures the behavioral consequence of conditioning mismatch during training and rollout. (4) **Compression depth** $\text{depth} = \mathbb{E}[c_i \mid c_i > 0]$, for c_i the fraction of rendered context evicted in rollout episode i, is how much an editor removes when it fires. Its companion factor $\text{freq} = \Pr[c_i > 0]$ is tabulated per cell in §G.6 instead of here: what triggers a firing is harness-specific, so freq compares down a column, not across. (5) Agentic **EM** is the primary downstream task metric and determines the final correctness ranking in §5.2. Full definitions are given in §G.

❼ **Optimization.** All cells are trained with **GRPO**: group size $G = 16$, learning rate 10^{-6}, KL coefficient $\beta_{\text{KL}} = 10^{-3}$, and a live-compression rollout pipeline, so training and deployment use the same rollout setting. Each cell retains at least 400 steps; a frozen dev split of $1{,}325$ questions, disjoint from every test benchmark, shortlists candidates for full-suite evaluation, and each reported EM is the best full-suite result among the cell's fully evaluated checkpoints (§G).

5.2 RESULTS ACROSS WHITE-BOX AND BLACK-BOX HARNESSES

Overview. Under live compression, the no-compression `logdiff` floor is 0.014, and separation grows with eviction: Naive-Compressed rises $0.012 \rightarrow 0.366$ and Naive-Full $0.021 \rightarrow 0.203$, while LogitTree remains 0.012–0.013 and Logit 0.006–0.022 across a comparably broad frequency range. SDCC reaches 0.071 at the eviction-heavy endpoint. Table 1 reports full matrix; `reward`/`logdiff` use

Table 1: **Grand result matrix (Qwen3-4B, live compression, real web tools).** `logdiff` is measured at each cell's maximum-reward iteration. Compression frequencies are in §G.6; black-box depth is N/A because internal evictions are hidden (§F.2). Bold and underlining mark the best and second-best values within each harness; blue shading highlights the best Avg. EM.

		Reward ↑	logdiff ↓		Compression		Agentic EM per Benchmark (%) ↑							Overall
Method	**Harness**	**Max**	**Mean**	**Max**	**Depth (%)**	**Max Tokens**	**NQ**	**TQA**	**HQA**	**2Wiki**	**MSQ**	**Bamb.**	**Frames**	**Avg. EM ↑**
Qwen3-4B without RL	*(reference checkpoints; absolute scores)*													
No-Compression	Base	N/A	N/A	N/A	0.0	289	3.2	21.6	8.4	2.6	2.5	23.2	8.0	9.9
	Search-R1	N/A	N/A	N/A	0.0	1,806	2.3	22.2	8.7	2.5	2.7	18.4	8.4	9.3
Compression	TC-RAG	N/A	N/A	N/A	2.00	2,117	3.1	23.2	9.2	3.1	3.9	16.0	9.7	9.7
	AgentFold	N/A	N/A	N/A	4.78	1,971	2.0	21.3	7.3	2.5	2.9	13.6	6.1	8.0
	MemexRL	N/A	N/A	N/A	12.58	1,670	1.2	15.8	5.5	1.1	1.7	12.0	7.3	6.4
	Claude Code	N/A	N/A	N/A	N/A	33,804	22.1	50.4	23.2	29.8	7.4	37.6	13.7	26.3
	OpenCode	N/A	N/A	N/A	N/A	34,285	31.0	55.7	26.6	32.7	8.4	36.6	12.1	29.0
Qwen3-4B with RL	*(colored Δ: reward/logdiff vs. RL no-compression; depth/tokens/EM vs. matching no-RL; depth/EM in pp, tokens in k)*													
No-compression	Search-R1	0.697 Δ 0.000	**0.014** Δ 0.000	0.015 Δ 0.000	0.0 Δ 0.0	36,370 Δ +34.6k	19.8 Δ +17.5	57.6 Δ +35.4	19.9 Δ +11.2	11.1 Δ +8.6	7.9 Δ +5.2	32.0 Δ +13.6	12.7 Δ +4.3	23.0 Δ +13.7
Naive-Full	TC-RAG	0.838 Δ +0.141	0.197 Δ +0.183	0.335 Δ +0.320	**60.5** Δ +58.5	37,044 Δ +34.9k	30.1 Δ +27.0	64.4 Δ +41.2	28.2 Δ +19.0	34.1 Δ +31.0	12.9 Δ +9.0	52.0 Δ +36.0	22.8 Δ +13.1	34.9 Δ +25.2
	AgentFold	0.600 Δ -0.097	0.203 Δ +0.189	0.333 Δ +0.318	78.2 Δ +73.4	14,190 Δ +12.2k	25.6 Δ +23.6	58.8 Δ +37.5	24.0 Δ +16.7	18.7 Δ +16.2	9.1 Δ +6.2	40.8 Δ +27.2	17.1 Δ +11.0	27.7 Δ +19.7
	MemexRL	0.734 Δ +0.037	0.021 Δ +0.007	0.051 Δ +0.036	43.2 Δ +30.6	**65,630** Δ +64.0k	29.2 Δ +28.0	63.1 Δ +47.3	28.0 Δ +22.5	31.3 Δ +30.2	9.7 Δ +8.0	48.0 Δ +36.0	15.2 Δ +7.9	32.1 Δ +25.7
Naive-Compressed	TC-RAG	0.850 Δ +0.153	0.015 Δ +0.001	0.041 Δ +0.026	54.3 Δ +52.3	38,171 Δ +36.1k	30.9 Δ +27.8	64.3 Δ +41.1	31.9 Δ +22.7	46.0 Δ +42.9	11.8 Δ +7.9	50.4 Δ +34.4	21.5 Δ +11.8	36.7 Δ +27.0
	AgentFold	**0.734** Δ +0.037	0.366 Δ +0.352	1.095 Δ +1.080	68.4 Δ +63.6	**16,708** Δ +14.7k	30.3 Δ +28.3	64.1 Δ +42.8	25.4 Δ +18.1	24.8 Δ +22.3	12.6 Δ +9.7	56.0 Δ +42.4	18.9 Δ +12.8	33.2 Δ +25.2
	MemexRL	**0.750** Δ +0.053	0.012 Δ -0.002	0.014 Δ -0.001	55.0 Δ +42.4	37,101 Δ +35.4k	23.9 Δ +22.7	60.5 Δ +44.7	22.9 Δ +17.4	20.5 Δ +19.4	9.7 Δ +8.0	47.2 Δ +35.2	17.6 Δ +10.3	28.9 Δ +22.5
Logit mask (ours)	TC-RAG	0.800 Δ +0.103	**0.006** Δ -0.008	**0.013** Δ -0.002	44.6 Δ +42.6	**40,182** Δ +38.1k	29.4 Δ +26.3	60.5 Δ +37.3	30.2 Δ +21.0	44.9 Δ +41.8	13.0 Δ +9.1	58.4 Δ +42.4	23.7 Δ +14.0	37.2 Δ +27.5
	AgentFold	0.632 Δ -0.065	0.022 Δ +0.008	0.091 Δ +0.076	79.7 Δ +74.9	15,383 Δ +13.4k	33.4 Δ +31.4	66.0 Δ +44.7	31.9 Δ +24.6	43.9 Δ +41.4	11.1 Δ +8.2	37.6 Δ +24.0	15.8 Δ +9.7	34.2 Δ +26.2
	MemexRL	0.416 Δ -0.281	0.013 Δ -0.001	0.017 Δ +0.002	45.4 Δ +32.8	36,528 Δ +34.9k	33.7 Δ +32.5	65.9 Δ +50.1	31.2 Δ +25.7	36.2 Δ +35.1	11.7 Δ +10.0	40.8 Δ +28.8	14.3 Δ +7.0	33.4 Δ +27.0
LogitTree (ours)	TC-RAG	0.914 Δ +0.217	0.012 Δ -0.002	**0.013** Δ -0.002	53.0 Δ +51.0	39,743 Δ +37.6k	36.2 Δ +33.1	65.6 Δ +42.4	36.1 Δ +26.9	54.1 Δ +51.0	13.9 Δ +10.0	55.2 Δ +39.2	19.3 Δ +9.6	40.1 Δ +30.4
	AgentFold	0.629 Δ -0.068	**0.012** Δ -0.002	**0.018** Δ +0.003	**83.3** Δ +78.5	16, 595 Δ +14.6k	**34.2** Δ +32.2	**67.9** Δ +46.6	34.7 Δ +27.4	46.8 Δ +44.3	12.5 Δ +9.6	46.4 Δ +32.8	19.8 Δ +13.7	37.5 Δ +29.5
	MemexRL	0.715 Δ +0.018	0.013 Δ -0.001	0.013 Δ -0.002	**64.1** Δ +51.5	40, 485 Δ +38.8k	**37.1** Δ +35.9	69.1 Δ +53.3	**42.0** Δ +36.5	**61.5** Δ +60.4	**19.1** Δ +17.4	**64.0** Δ +52.0	**28.5** Δ +21.2	**45.9** Δ +39.5
	Claude Code	**0.839** Δ +0.142	0.016 Δ +0.002	0.016 Δ +0.001	N/A	65,167 Δ +31.4k	38.2 Δ +16.1	64.1 Δ +13.7	36.5 Δ +13.3	45.8 Δ +16.0	**23.1** Δ +15.7	**57.2** Δ +19.6	**32.0** Δ +18.3	42.4 Δ +16.1
	OpenCode	**0.929** Δ +0.232	**0.012** Δ -0.002	**0.012** Δ -0.003	N/A	73,589 Δ +39.3k	**36.6** Δ +5.6	**63.2** Δ +7.5	38.6 Δ +12.0	59.0 Δ +26.3	22.4 Δ +14.0	53.9 Δ +17.3	26.7 Δ +14.6	42.9 Δ +13.9
SDCC (ours)	TC-RAG	**0.922** Δ +0.225	0.013 Δ -0.001	0.015 Δ 0.000	57.8 Δ +55.8	37,590 Δ +35.5k	**37.8** Δ +34.7	**70.2** Δ +47.0	**42.9** Δ +33.7	**63.8** Δ +60.7	**17.7** Δ +13.8	**60.2** Δ +44.2	**29.6** Δ +19.9	**46.0** Δ +36.3
	AgentFold	0.713 Δ +0.016	0.071 Δ +0.057	0.156 Δ +0.141	73.3 Δ +68.5	9,416 Δ +7.4k	33.9 Δ +31.9	67.1 Δ +45.8	**39.7** Δ +32.4	**57.9** Δ +55.4	**14.4** Δ +11.5	**58.4** Δ +44.8	**25.1** Δ +19.0	**42.4** Δ +34.4
	MemexRL	**0.750** Δ +0.053	**0.011** Δ -0.003	**0.012** Δ -0.003	50.3 Δ +37.7	36,364 Δ +34.7k	37.0 Δ +35.8	**69.2** Δ +53.4	39.5 Δ +34.0	56.9 Δ +55.8	15.6 Δ +13.9	59.2 Δ +47.2	24.5 Δ +17.2	43.1 Δ +36.7
	Claude Code	0.648 Δ -0.049	**0.015** Δ +0.001	**0.015** Δ 0.000	N/A	**82,779** Δ +49.0k	**38.8** Δ +16.7	**67.7** Δ +17.3	**36.4** Δ +13.2	**44.3** Δ +14.5	24.8 Δ +17.4	55.0 Δ +17.4	31.5 Δ +17.8	**42.6** Δ +16.3
	OpenCode	0.717 Δ +0.020	0.013 Δ -0.001	0.013 Δ -0.002	N/A	**76,807** Δ +42.5k	38.7 Δ +7.7	68.6 Δ +12.9	**39.0** Δ +12.4	**58.2** Δ +25.5	**17.5** Δ +9.1	**57.9** Δ +21.3	**27.6** Δ +15.5	**43.9** Δ +14.9

each cell's argmax-reward iteration (semantics: §G.4). Depth is mean removal conditional on firing, "Max tok." the largest rendered request, and black-box depth is N/A because of hidden evictions.

The predicted failure signs and matrix structure agree. On an untrained Qwen3-4B, Naive-Comp under-places ($\Delta_{\text{comp}} < 0$) and Naive-Full-leak over-places ($\Delta_{\text{full}} > 0$) across all three white-box harnesses, with the predicted sign holding for most tokens. Per token, $\Delta_{\text{comp}} = \log p(y_t|\mathbf{H}^{\text{comp}}[:t]) - \log p(y_t|\mathbf{p_t}(y_t))$ (diverging leaves) and $\Delta_{\text{full}} = \log p(y_t|c_t^{\text{leak}}) - \log p(y_t|\mathbf{H}^{\text{comp}}[:t])$ (post-eviction; c_t^{leak} reinjects Naive-Full's training-only content); magnitudes are ranked TC-RAG $\approx$ AgentFold $\gg$ MemexRL for Δ_{comp} and reverse for Δ_{full} (Figure 2(b); §G.2). In the trained matrix, exact walks recover the floor while Naive-Compressed reaches 0.366 on AgentFold. Small LogitMask–LogitTree differences at the floor do not contradict Theorem 1, which equates same-batch gradients; table rows are independent runs.

SDCC and black-box transfer. SDCC's training KL spans 10^{-4} on MemexRL to 0.356 on AgentFold. Over the logged AgentFold window, its mean `logdiff` falls by 55% and crosses the trendless Naive-Comp trace; this single-editor, single-scale comparison has no confidence interval, and Naive-Comp is the matched $\lambda = 0$ ablation (Figure 11; §G.6). Every white-box trained row

exceeds the Search-R1 EM of 23.0. On Claude Code, SDCC reaches 37.5 average EM versus LogitTree's 35.9; on OpenCode, 36.9 versus 35.0, with both harnesses near drift floor (0.012–0.016).

5.3 Large-scale evaluation on WideSearch

Claude Code and OpenCode devote roughly 25K of their 40K-token budget to the system prompt, leaving limited interaction context. We therefore scale to **Qwen3.7-Air (125B-A6B)** with **Claude Code** on WideSearch (Wong et al., 2025): 256 H100s, 15 LogitTree steps, and 200 questions with four trials each (800 executions), comparing Base with nine reference models.

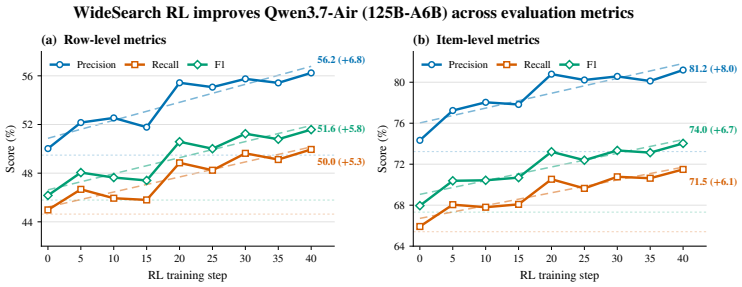

Figure 4: WideSearch training dynamics.

Relative to Base, LogitTree raises Item P/R/F1 from 73.23/65.41/67.32 to 81.19/71.49/74.03. All six curves finish above Base (Figure 4), and LogitTree leads all item-level metrics.

Table 2: **WideSearch comparison.** References are ordered by Item F1; the matched Air pair is below the rule. P/R denotes precision/recall; bold/underline marks best/second-best, and blue shading highlights the matched-pair best item scores.

	Row-Level (%)			Item-Level (%)		
Model	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Reference models *(higher is better; absolute scores)*						
GPT-5.2 (Thinking)	38.79	37.39	37.51	49.14	46.91	47.13
GPT-5.2	39.74	32.93	34.29	67.51	54.24	56.79
Qwen3-Max	39.03	32.13	33.49	67.77	55.52	57.94
Qwen3.6-Plus	37.68	33.89	34.82	66.51	59.91	61.65
Claude Opus 4.6 (Max)	52.03	48.01	49.03	67.98	61.45	63.10
GPT-5.6	52.63	**52.35**	<u>52.22</u>	66.49	66.63	66.11
Gemini 3 Pro	53.07	46.86	48.37	76.03	65.44	67.95
Kimi K2.5 (Thinking)	54.76	48.69	50.18	79.69	70.19	72.46
Claude Opus 4.5	**57.04**	<u>51.25</u>	**52.74**	<u>79.99</u>	<u>71.31</u>	<u>73.55</u>
Matched Qwen3.7-Air pair *(higher is better; colored Δ in pp vs. Base SFT)*						
Qwen3.7-Air (125B-A6B) Base (SFT)	49.48	44.63	45.79	73.23	65.41	67.32
Qwen3.7-Air (125B-A6B) + LogitTree (ours)	<u>56.24</u> Δ +6.76	49.95 Δ +5.32	51.58 Δ +5.79	**81.19** Δ +7.96	**71.49** Δ +6.08	**74.03** Δ +6.71

6 Conclusion and Future Work

Viewing a compressed rollout as a *conditioning tree* reveals a distinct train–inference mismatch in editor-based agent RL. The two default replay schemes fail in opposite directions: Naive-Compressed conditions tokens on prefixes that are too short, while Naive-Full conditions them on prefixes that are too long. Across different models, white-box and black-box harnesses, and multiple logit recomputation schemes, we show that this mismatch appears as measurable logit drift and degraded rollout reward. To address it, we present two exact conditioning-consistent solutions, **LogitTree** and the **Logit attention mask**, together with a training-efficient approximation, **SDCC**. Exact replay restores the no-compression drift floor, while SDCC substantially narrows the gap with a single-backward objective. Future work includes extending the framework to broader compression policies (e.g., dropping tool observations or using latent summaries), testing across a broader range of harnesses, and adapting exact tree-consistent replay to sparse and linear-attention architectures.

AI USE STATEMENT

Large language models (LLMs) were used to assist with manuscript writing, language polishing, figure presentation, and LaTeX formatting. Specifically, LLMs helped refine clarity, grammar, and readability, suggest alternative phrasings, and reduce repetitive expressions. They also assisted with visualization layouts and plotting and document-preparation code. The experimental values used in these figures were supplied and verified by the authors; generative AI was not used to create these values. The authors retain responsibility for reviewing AI-assisted text, code, and figures, for the interpretation of the results, and for all final scientific and editorial decisions.

ETHICS STATEMENT

This work studies automated search agents using publicly available language models, question-answering benchmarks. The experiments evaluate model behavior rather than human participants. The study aims to reduce unsupported answers by improving how agents use external knowledge, retrieved evidence. Nevertheless, retrieved passages may contain inaccurate, outdated, or biased information, and lower unreliability does not by itself establish factual correctness or suitability for high-stakes use. Use of the models, datasets, and retrieval corpus should respect their respective licenses and terms of use.

REPRODUCIBILITY STATEMENT

Experiments use Qwen3-4B on HotpotQA, 2WikiMultiHopQA, Bamboogle, FRAMES, GAIA, MuSiQue, NQ, PopQA, and TriviaQA. Experiment section describes the data sources, metrics, baseline reproduction protocol, and training setup. All baselines are reimplemented and retuned under the shared evaluation protocol at most five retrieval iterations, method-adapted prompt templates. Appendix records the model configuration, software environment, hardware, generation settings, and training hyperparameters.

REFERENCES

Alexander A. Alemi, Ben Poole, Ian Fischer, Joshua V. Dillon, Rif A. Saurous, and Kevin Murphy. Fixing a broken ELBO. In *International Conference on Machine Learning (ICML)*, 2018.

Anthropic. Claude code. `https://docs.anthropic.com/en/docs/claude-code`, 2024.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning (ICML)*, 2022.

Vivek S. Borkar. Stochastic approximation with two time scales. *Systems & Control Letters*, 29(5): 291–294, 1997.

Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. In *International Conference on Learning Representations (ICLR)*, 2015. Oral presentation; arXiv:1406.2751.

Ming-Bin Chen, Jey Han Lau, and Lea Frermann. Cig: Measuring conversational information gain in deliberative dialogues with semantic memory dynamics, 2026. URL `https://arxiv.org/abs/2604.15647`.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. ReSearch: Learning to reason with search for LLMs via reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Zheng Chu, Xiao Wang, Jack Hong, Huiming Fan, Yuqi Huang, Yue Yang, Guohai Xu, Chenxiao Zhao, Cheng Xiang, Shengchao Hu, Dongdong Kuang, Ming Liu, Bing Qin, and Xing Yu. Redsearcher: A scalable and cost-efficient framework for long-horizon search agents, 2026. URL `https://arxiv.org/abs/2602.14234`.

Hantian Ding, Zijian Wang, Giovanni Paolini, Varun Kumar, Anoop Deoras, Dan Roth, and Stefano Soatto. Fewer truncations improve language modeling. In *International Conference on Machine Learning (ICML)*, 2024.

Quentin Gallouédec and Kashif Rasul. Agentic RL: Token-in, token-out done right. Hugging Face blog, `https://huggingface.co/blog/huggingface/tito`, 2026. Token-level loss-mask baseline; defers on history rewriting.

Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations (ICLR)*, 2017.

Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, pp. 6609–6625, 2020.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Xinke Jiang, Yue Fang, Rihong Qiu, Haoyu Zhang, Yongxin Xu, Hao Chen, Wentao Zhang, Ruizhe Zhang, Yuchen Fang, Xinyu Ma, Xu Chu, Junfeng Zhao, and Yasha Wang. TC-RAG: Turing–complete RAG's case study on medical LLM systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11400–11426, Vienna, Austria, 2025. Association for Computational Linguistics.

Xinke Jiang, Yue Fang, Zhibang Yang, Jiaran Gao, Zhixin Zhang, Tao Feng, Rihong Qiu, Wentao Zhang, Hongxin Ding, Ruizhe Zhang, et al. AgenticRAG-R1: Agentic reinforcement learning with stack memory for multi-step reasoning, retrieval and memorizing. In *EMNLP*, 2026. URL `https://github.com/jiangxinke/Agentic-RAG-R1`.

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. *arXiv preprint arXiv:2503.09516*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, 2017.

Satyapriya Krishna, Kalpesh Krishna, Anhad Mohananey, Steven Schwarcz, Adam Stambler, Shyam Upadhyay, and Manaal Faruqui. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL) — Volume 1: Long Papers*, pp. 4745–4759, Albuquerque, New Mexico, 2025. arXiv preprint arXiv:2409.12941.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics (TACL)*, 7:453–466, 2019.

Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

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*, 2024.

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, 2023. URL https://arxiv.org/abs/2307.03172.

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.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with VQ-VAE-2. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Machel Reid, Nikolay Savinov, Denis Teplyashin, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, 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.

Zhengliang Shi, Yiqun Chen, Haitao Li, Weiwei Sun, Shiyu Ni, Yougang Lyu, Run-Ze Fan, Bowen Jin, Yixuan Weng, Minjun Zhu, Qiujie Xie, Xinyu Guo, Qu Yang, Jiayi Wu, Jujia Zhao, Xiaqiang Tang, Xinbei Ma, Cunxiang Wang, Jiaxin Mao, Qingyao Ai, Jen-Tse Huang, Wenxuan Wang, Yue Zhang, Yiming Yang, Zhaopeng Tu, and Zhaochun Ren. Deep research: A systematic survey, 2025. URL https://arxiv.org/abs/2512.02038.

Alibaba Cloud Model Team. Qwen-agent: An agent framework for qwen large language models, 2024.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Alexandre B. Tsybakov. *Introduction to Nonparametric Estimation*. Springer, 2009.

Huanting Wang, Jingzhi Gong, Huawei Zhang, Jie Xu, and Zheng Wang. AI agentic programming: A survey of techniques, challenges, and opportunities, 2025. URL https://arxiv.org/abs/2508.11126.

Zhenting Wang, Huancheng Chen, Jiayun Wang, and Wei Wei. Memex(RL): Scaling long-horizon LLM agents via indexed experience memory, 2026. URL https://arxiv.org/abs/2603.04257.

Ryan Wong, Jiawei Wang, Junjie Zhao, Li Chen, Yan Gao, Long Zhang, Xuan Zhou, Zuo Wang, Kai Xiang, Ge Zhang, Wenhao Huang, Yang Wang, and Ke Wang. WideSearch: Benchmarking agentic broad info-seeking. *arXiv preprint arXiv:2508.07999*, 2025.

Yunjia Xi, Jianghao Lin, Yongzhao Xiao, Zheli Zhou, Rong Shan, Te Gao, Jiachen Zhu, Weiwen Liu, Yong Yu, and Weinan Zhang. A survey of LLM-based deep search agents: Paradigm, optimization, evaluation, and challenges, 2025. URL https://arxiv.org/abs/2508.05668.

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.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *International Conference on Learning Representations (ICLR)*, 2024.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schütze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning, 2026. URL https://arxiv.org/abs/2508.19828.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

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.

Rui Ye, Zhongwang Zhang, Kuan Li, Huifeng Yin, Zhengwei Tao, Yida Zhao, Liangcai Su, Liwen Zhang, Zile Qiao, Xinyu Wang, Pengjun Xie, Fei Huang, Siheng Chen, Jingren Zhou, and Yong Jiang. AgentFold: Long-horizon web agents with proactive context management. *arXiv preprint arXiv:2510.24699*, 2025.

Ruizhe Zhang, Xinke Jiang, Zhibang Yang, Zhixin Zhang, Jiaran Gao, Yuzhen Xiao, Tao Feng, Yue Fang, Yuxuan Liu, Ruiqing Li, Hongbin Lai, Huheng Huang, Xu Chu, Junfeng Zhao, and Yasha Wang. Stackplanner: A centralized hierarchical multi-agent system with task-experience memory management, 2026a. URL https://arxiv.org/abs/2601.05890.

Yaxiang Zhang, Yingru Li, Jiacai Liu, Jiawei Xu, Ziniu Li, Qian Liu, and Haoyuan Li. Beyond precision: Training-inference mismatch is an optimization problem and simple lr scheduling fixes it, 2026b. URL https://arxiv.org/abs/2602.01826.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Dongzhuoran Zhou, Yuqicheng Zhu, Yule Liu, Zhen Yang, Rui Lu, Yuxiao Dong, Jie Tang, and Evgeny Kharlamov. Awm: Answerable working memory for long-document vqa agents, 2026. URL https://arxiv.org/abs/2608.25618.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

Zilin Zhu, Chengxing Xie, Xin Lv, and slime Contributors. slime: An LLM post-training framework for RL scaling. https://github.com/THUDM/slime, 2025. GitHub repository. Corresponding author: Xin Lv.

PART I POSITION IN THE AGENTIC-RL LITERATURE

At the time of writing, published baselines for harness-RL training on QA-style benchmarks (Search-R1 (Jin et al., 2025), ReSearch (Chen et al., 2025)) all train on the physical trajectory $\mathbf{H}_t$. AgenticRAG-R1 (Jiang et al., 2026) and MEM1 (Zhou et al., 2025) have begun exploring RL with stack-based or learned memory mechanisms for long-horizon agents. To our knowledge, our work is the first to formalize the conditioning invariant, characterize the failure modes that follow from violating it, and propose a soft remedy with a provable bias bound.

A RELATED WORK

Harness context compression. Most production long-horizon agent harnesses include an external editor that mutates the physical history before the next decoding step. Claude Code (Anthropic, 2024) and Qwen-Agent (Team, 2024) maintain a bounded context window and inject an auto-compact summary in place of the evicted prefix. MemexRL (Wang et al., 2026) and MEMGPT (Packer et al., 2023) let agents offload content to, and retrieve it from, an external store. TC-RAG and StackPlanner (Jiang et al., 2025; Zhang et al., 2026a) manage context as an explicit stack whose pop evicts the oldest retrieval envelope. AgentFold and Mem1 (Ye et al., 2025; Zhou et al., 2025) periodically fold the prefix into a self-generated summary. A complementary line studies working-memory design for agents operating over long documents, retaining answerable evidence while compacting the rest of the context (Zhou et al., 2026). A parallel line trains dedicated compressors, including token-budget compression with small auxiliary models such as LLM-LINGUA (Jiang et al., 2023) and RECOMP (Xu et al., 2024), which act as scheduled deterministic editors. At the model-internal level, H2O, SnapKV, and StreamingLLM (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2024) compress the KV cache, while RETRO (Borgeaud et al., 2022) augments the model with retrieval. These mechanisms are complementary to harness-level editing: they alter internal storage or retrieval, whereas our setting rewrites the rendered transcript between model calls. Under such edits, the context retained for training need not be the live view under which a token was decoded.

Agent RL over edited contexts. Recently, policy-gradient RL (Schulman et al., 2017; Shao et al., 2024; Zheng et al., 2025) and their agentic extensions (Jin et al., 2025; Chen et al., 2025; Jiang et al., 2026) fine-tune LLMs on rollout trajectories, typically treating the edited context as ordinary sequential data. Memory-R1 (Yan et al., 2026; Zhang et al., 2026a) and MemexRL (Wang et al., 2026) go further by making memory management itself learnable: the policy can offload spans to external storage and retrieve on demand, thereby optimizing the memory-editing process. However, neither line of work asks **which context the training gradient should condition on once the rollout history has been rewritten**. Related notions of train–inference consistency address different failure modes. TITO (Gallouédec & Rasul, 2026), whose token-level scheme our LogitTree extends to trajectory trees, studies token-level consistency—whether the token IDs observed during rollout are replayed identically during training—but does not address harness-induced history rewriting. Our focus is orthogonal: the conditioning context itself changes, which extends block-causal masking for long-context training (Ding et al., 2024; Reid et al., 2024; Zhou et al., 2025).

PART II METHOD SUPPLEMENTS: PITFALLS, STRUCTURE, PROOFS, VARIATIONAL DERIVATION, CONVERGENCE

This part supplies the technical scaffolding behind the development of §2–§4: a worked tool-using rollout in which a single eviction places two target spans on opposite sides of it, so that both pitfalls and the mirror symmetry between them can be read off one concrete trajectory (§B); the trajectory-tree object itself (§C); every proof stated in §3–§4, restated so the appendix reads self-contained (§D); and the end-to-end β-VAE derivation of SDCC with its convergence analysis (§E, §E.6).

B A WORKED EXAMPLE: ONE WEATHER LOOKUP, TWO PITFALLS

§2 states both pitfalls over abstract tokens y_t; here both sit on one rollout with one eviction, so the mirror symmetry of Table 5 is read off rather than derived. **The rollout is constructed for exposition: it is not a logged trajectory, and no quantity in it is a measurement.**

Setup. TC-RAG's learnable pop (§F.2) deletes an envelope outright, leaving no summary behind to carry the forecast forward. Asked about an umbrella for tomorrow's meeting in Shanghai, the policy calls `get_weather`, receives envelope o_1, and writes span S — "*rain is likely tomorrow afternoon — bring an umbrella*" — **while** o_1 **is in view**. The budget is then exceeded, pop evicts the largest envelope o_1 at junction J_1 ($E_1 = o_1$), and a later `get_transit` turn produces span T **after the eviction**. Below, *rollout* is the live view a target was decoded under and *training* the constructed prefix the objective scores it under; the one row where they disagree is the whole failure.

B.1 ❶ PITFALL A ON S: THE MODEL IS TAUGHT THAT IT KNOWS THE WEATHER

Naive-Compressed, time-travel leakage: *"I already know the weather."*

Naive-Compressed trains on the final compressed walk $\mathbf{H}^{\text{comp}}$, so S is scored under $\mathbf{p_s}(S)$: J_1 fires *after* S is decoded, yet its effect is already applied (Eq. 4, $\mathbf{p_s}(S) \subsetneq \mathbf{p_t}(S)$).

	rollout $\mathbf{p_t}(S)$	*training* $\mathbf{p_s}(S)$
`[user] umbrella for tomorrow's Shanghai meeting?`	✓	✓
`[call] get_weather("Shanghai", +1d)`	✓	✓
`[obs ] weather envelope (afternoon rain)`	✓	×
`[tgt ]` S`: "...rain tomorrow afternoon..."`	*scored here*	

The tool *call* survives, only its *result* is gone, so $-\log P_\theta(S \mid \mathbf{p_s}(S))$ demands the forecast from a prefix that no longer holds it; the only way to cut the term is to move mass onto "rain tomorrow afternoon" as a *prior over Shanghai weather*. A tool-grounded assertion becomes a from-memory one, and the deployed model states the forecast before reading the tool. **Too little context.**

B.2 ❷ PITFALL B ON T: THE MODEL IS TAUGHT TO READ A DISCARDED CONTEXT

Naive-Full, stale-context leakage: *"I compressed that — why is it still in my context?"*

Naive-Full trains on the depth-first tape $\mathbf{H}_T$, which deletes nothing, so T is scored under the physical prefix $\mathbf{H}_{<t}$, which still holds the o_1 its live view had lost (Eq. 5, $\mathbf{H}_{<t} \supsetneq \mathbf{p_t}(T)$).

	rollout $\mathbf{p_t}(T)$	*training* $\mathbf{H}_{<t}$
`[user] umbrella for tomorrow's Shanghai meeting?`	✓	✓
`[call] get_weather("Shanghai", +1d)`	✓	✓
`[obs ] weather envelope (afternoon rain)`	×	✓
`[span]` S`: "...rain tomorrow afternoon..."`	✓	✓
`[pop ]` J_1`:` E_1 `= weather envelope`	✓	✓
`[turn] get_transit(...)` $\rightarrow$ `transit envelope`	✓	✓
`[tgt ]` T`: "Line 2 runs every four minutes..."`	*scored here*	

The gradient therefore rewards reading the forecast back *after* the pop — a move the deployed agent cannot make, because there the deletion is real. **Too much context.**

The mirror. On row `[obs] weather envelope`, Pitfall A reads $(\checkmark, \times)$ and Pitfall B reads $(\times, \checkmark)$: A drops the forecast from a target that used it, B keeps it for one that had lost it, so no reweighting of a single serialization repairs both. §C.4 measures this — $\Delta_{\text{comp}} = -22.8$ against $\Delta_{\text{full}} = +18.5$ nats on an untrained Qwen3-4B, opposite in sign and comparable in magnitude. Those two numbers are measurements; the weather rollout is not. Both defects vanish under one repair: score every target under the live view it was decoded from (Definition 2).

Table 3: **Tree sizes induced by the three white-box harnesses** on the running example (probe of Table 7); K = junctions per rollout, D = diverging leaves per rollout, $|E_k|$ = mean evicted-span length per junction.

Harness	mean K	mean diverging leaves D	mean $\lvert E_k \rvert$
TC-RAG	0.87	29	156
AgentFold	1.94	58	84
MemexRL	2.00	125	41

C THE TRAJECTORY TREE: A UNIFIED STRUCTURAL ANALYSIS

This appendix collects, in one place, the tree-theoretic view underlying every construction in the main text. §2 introduces the running-example tree $\mathcal{T}$; §C.4 states the conditioning invariant on its leaves; §3 and §4 materialize that invariant with a Logit mask, with LogitTree, and with SDCC. The purpose here is to make the object $\mathcal{T}$ explicit as a graph, count its components (junctions, leaves, branches), identify where each method places gradient, and read off the compute budget from the counts.

C.1 DEFINITION OF THE TRAJECTORY TREE

Definition 1 (Trajectory tree)**.** Fix a rollout with physical trajectory $\mathbf{H}_T = (x_1, \ldots, x_T)$ and eviction records $\{(J_k, E_k)\}_{k=1}^{K}$, where the k-th compression fires at physical position J_k and removes the positions $E_k \subset \{1, \ldots, J_k\}$. The trajectory tree $\mathcal{T} = (V, E, \phi)$ has a *root* (the initial user prompt), a *junction* J_k for each k, and a *leaf* for each generated token y_t; edges are labeled by physical-token spans, so following edges from the root to a node v recovers the physical prefix that produced v. Two colorings ϕ_{spine}, $\phi_{\text{leg}} : V \to 2^{\{1,\ldots,T\}}$ record the two conditioning sets of interest: $\phi_{\text{spine}}(v)$ is the prefix visible on the final compressed walk, with *every* eviction applied, and $\phi_{\text{leg}}(v)$ is the live view of equation 2 — the prefix visible when v was decoded, carrying exactly the evictions that had fired by that step and no others. A node decoded at step t is *diverging* iff the two disagree,

$$\phi_{\text{leg}}(v) \setminus \phi_{\text{spine}}(v) \;=\; \bigcup_{j:\, J_j > t} E_j \cap \{1, \ldots, t-1\} \;\neq\; \emptyset,$$

i.e. iff some eviction that fires *after* t reaches back into v's prefix.

Two regions of $\mathcal{T}$ are non-diverging by construction: nodes whose prefixes precede every evicted span, and nodes decoded at or after the final junction J_K, for which all evictions have already been applied and the tree *re-converges* (cf. leaf y_{ans} in Table 4). Nodes decoded within $[J_{k-1}, J_k)$ form the k-th *branch layer*, with $J_0 = 0$.

Counts. Write R for the number of response tokens and D for the number of diverging leaves. Then $\mathcal{T}$ carries one root, K junctions, R leaves, and $K+1$ root-to-leaf branches, and each leaf has an eviction mass $|\phi_{\text{leg}} \setminus \phi_{\text{spine}}|$ — the tokens it conditioned on at generation that its final-walk prefix no longer contains. Only two of these numbers are read off downstream: the junction depth K, which upper-bounds the number of independent forwards a LogitTree-style method needs, and the diverging-leaf count D, which is exactly the set of positions where SDCC places KL gradients. Table 3 reports both for the three white-box harnesses at $T_{\max} = 5$, so tree size can be read directly off the harness knob. Two regularities matter later: the per-eviction mass $|E_k|$ ranks inversely with K (TC-RAG evicts large spans rarely, MemexRL small slices often), and these are *structural* capacities under a forced schedule, not live firing rates — MemexRL has the deepest tree here but the *lowest* model-triggered eviction density in Table 10.

C.2 WHERE EACH METHOD PLACES GRADIENT ON $\mathcal{T}$

- **Naive-Full** runs one forward on the union $\mathbf{H}_T$ and takes gradients at every response token position. On $\mathcal{T}$ this collapses the tree into its depth-first traversal: every leaf is conditioned on the *entire* physical prefix $\mathbf{H}_{<t} \supseteq \phi_{\text{leg}}(v)$, which contains all evicted spans — including spans that had already left the live view when v was decoded. The training conditioning strictly exceeds the live

view, so Naive-Full trains on information the policy never had at any point in the rollout (Pitfall B, §2.2).

- **Naive-Compressed** runs one forward on the compressed walk $\mathbf{H}^{\text{comp}}$ and takes gradients at every response token. On $\mathcal{T}$ this collapses all branches into the final walk: the training forward sees $\phi_{\text{spine}}(v)$, which at the D diverging leaves is a *strict subset* of the live view $\phi_{\text{leg}}(v)$ that generated the token — and, worse, contains summary content created after t. The loss on those leaves is therefore computed under conditioning the model never decoded from (Pitfall A, §2.2).
- **LogitTree (ours, K-forward)** runs one forward per branch layer, so $K+1$ forwards per rollout. Each forward covers a root-to-leaf path with $\phi_{\text{spine}} = \phi_{\text{leg}}$ along the entire branch (because the branch's forward uses the *live view* as it existed during that layer). Gradients are placed on the leaves of that branch only, so the union of LogitTree's gradient-carrying positions equals the leaf set of $\mathcal{T}$ without double-counting.
- **Logit mask (ours, packed)** realizes the same partition as LogitTree but in a *single* forward: one structured mask admits, in each query row, exactly the keys that were live when that token was decoded, so $\phi_{\text{spine}} = \phi_{\text{leg}}$ holds row by row and the branches share the tree trunk instead of being replayed. Gradient placement is identical to LogitTree's; the difference is the pass count: one forward instead of $K+1$.
- **SDCC (ours)** runs one student forward on $\mathbf{H}^{\text{comp}}$ (with gradient) plus one gradient-free teacher forward per branch layer that holds a diverging leaf — at most K, since the layer after J_K re-converges. The policy-gradient loss remains on the student's R-length response tokens. The consistency KL $D_{\text{KL}}(\pi_{\theta,\text{teach}}(\cdot \mid \mathbf{H}'_t) \,\|\, \pi_{\theta,\text{stud}}(\cdot \mid \mathbf{H}^{\text{comp}}[:t]))$ is summed only over the D diverging leaves, whose positions are given by the diverging-leaf mask constructed from the harness's $\{J_k, E_k\}$ records. On $\mathcal{T}$, SDCC pulls the student's per-leaf distribution toward the teacher's at exactly the diverging positions, without touching the trunk. This is where the $O(\sqrt{\varepsilon_{\text{KL}}})$ bound of §E.4 applies: as SDCC's KL contracts, the student's leaf distributions approach the teacher's, and the diverging-leaf gap closes.

C.3 Compute budget as tree traversal cost

Three lengths govern the budget: the union length $L = |\mathbf{H}_T|$; the compressed walk $C = |\mathbf{H}^{\text{comp}}| = L - |\bigcup_k E_k|$, shorter than L by the token *mass* evicted and not by the eviction *count*; and ℓ_i, the live view in force during branch layer i. Consecutive branches share the trunk, so $C \le \ell_i \le L$ — with $\ell_{K+1} = C$, every eviction having fired by the last layer — and $\sum_{i=1}^{K+1} \ell_i \ge L$ with equality only at $K = 0$: LogitTree re-reads the shared prefix once per layer rather than splitting L into $K+1$ disjoint pieces. Table 6 tabulates the resulting per-rollout pass and token budgets next to the infrastructure each method demands.

Tokens are a proxy, not a cost: a forward over n tokens costs $F(n) = \Theta(nd^2 + n^2 d)$, and for Qwen3-4B the quadratic term overtakes the linear one at $n \approx 12$k — inside the range of contexts we observe (Table 1). A dense kernel scores the packed sequence in full and discards the masked entries, paying $\Theta(L^2 d)$, while $K+1$ segmented forwards pay $\Theta(\sum_i \ell_i^2 d)$, so neither dominates — it turns on the evicted mass. The Logit mask's dependable advantage over LogitTree is one backward instead of $K+1$ and one well-filled kernel launch instead of $K+1$ short ones, not fewer FLOPs; a genuine FLOP saving needs a block-sparse kernel. SDCC saves nothing on the forward: because $\ell_{K+1} = C$, the student pass *is* the last branch layer's, and SDCC's token budget $C + \sum_{i=1}^{K} \ell_i$ equals LogitTree's $\sum_{i=1}^{K+1} \ell_i$ term for term. The entire saving is the backward, 1 instead of $K+1$; and unlike either exact walk it gives up exactness, down to $\mathcal{O}(\sqrt{\varepsilon_{\text{KL}}})$.

C.4 Per-leaf divergence and the two pitfalls

Recall that the editor fires repeatedly *inside* the rollout loop: at step t the policy sees the live view $\mathbf{H}'_t = \textsc{View}_t(\mathbf{H}_{\le t}; \mathcal{E}_{\le t})$ of equation 2 — the physical history with exactly the edits that have fired by step t applied, and no others — whereas the length-t prefix $\mathbf{H}^{\text{comp}}[:t]$ of the final compressed walk retroactively applies *future* edits $\mathcal{E}_{>t}$ to a token generated before those edits existed. Table 4 instantiates the divergence for each leaf of the running example.

Table 4: Per-leaf live-view prefix vs. final-walk prefix for the running example ($E_1 = \{y_5\}$ at $J_1 = 7$; $E_2 = \{y_8\}$ at $J_2 = 10$; $E_3 = \{y_{11}\}$ at $J_3 = 12$). The leaf whose two prefixes coincide (y_{ans}) does not need any correction. Evicting one token per junction keeps each row on a single line; widening a span enlarges E_k and each leaf's eviction mass, but changes neither which leaves diverge nor where the tree re-converges.

Leaf y_t	Live-view prefix $\mathbf{p_t}(y_t)$	Final-walk prefix $\mathbf{p_s}(y_t)$	Diverges by
y_6	$\{y_1, y_2, y_3, y_4, y_5\}$	$\{y_1, y_2, y_3, y_4\}$	$+\, y_5$
y_7	$\{y_1, y_2, y_3, y_4, y_5, y_6\}$	$\{y_1, y_2, y_3, y_4, y_6\}$	$+\, y_5$
y_9	$\{y_1, y_2, y_3, y_4, y_6, y_7, y_8\}$	$\{y_1, y_2, y_3, y_4, y_6, y_7\}$	$+\, y_8$
y_{10}	$\{y_1, y_2, y_3, y_4, y_6, y_7, y_8, y_9\}$	$\{y_1, y_2, y_3, y_4, y_6, y_7, y_9\}$	$+\, y_8$
y_{12}	$\{y_1, y_2, y_3, y_4, y_6, y_7, y_9, y_{10}, y_{11}\}$	$\{y_1, y_2, y_3, y_4, y_6, y_7, y_9, y_{10}\}$	$+\, y_{11}$
y_{ans}	$\{y_1, y_2, y_3, y_4, y_6, y_7, y_9, y_{10}, y_{12}\}$	same	—

Pitfall A, SFT view (time-travel leakage).

$$\mathcal{L}_{\text{SFT}}^{\text{A}} \;=\; -\sum_t \log P_\theta(y_t \mid \mathbf{p_s}(y_t)) \;\neq\; -\sum_t \log P_\theta(y_t \mid \mathbf{p_t}(y_t)) \;=\; \mathcal{L}_{\text{SFT}}^{\star}. \tag{11}$$

The model is fitted to a shorter-prefix distribution than the one that generated the target token, while at inference the harness still delivers the live view $\mathbf{H}'_t = \mathbf{p_t}(y_t)$ — a prefix the model was never trained to match.

Pitfall A, RL view (broken importance ratio). The rollout policy sampled $a_t \sim \pi_{\theta_{\text{old}}}(\cdot \mid \mathbf{p_t}(y_t))$, so the training ratio

$$\rho_t^{\text{A}} \;=\; \frac{\pi_\theta(a_t \mid \mathbf{p_s}(y_t))}{\pi_{\theta_{\text{old}}}(a_t \mid \mathbf{p_t}(y_t))} \;\neq\; \frac{\pi_\theta(a_t \mid \mathbf{p_t}(y_t))}{\pi_{\theta_{\text{old}}}(a_t \mid \mathbf{p_t}(y_t))} \;=\; \rho_t \tag{12}$$

invalidates PPO's clip guarantee and mis-weights GRPO's group-relative advantage; empirically, the clip-rate map concentrates immediately after each junction, precisely where the two prefixes diverge.

Pitfall B, SFT view (train–inference mismatch).

$$\mathcal{L}_{\text{SFT}}^{\text{B}} \;=\; -\sum_t \log P_\theta(y_t \mid \mathbf{H}_{<t}). \tag{13}$$

Cross-entropy is computed using a prefix that carries strictly more information — including already-evicted content — than the deployed model will ever see.

Pitfall B, RL view (advantage on the wrong observation). Eventually-evicted tokens remain present during training, so they still absorb a share of the outcome reward r_{out}/T, and the advantage baseline is fitted to $V(\mathbf{H}_{<t})$ instead of $V(\mathbf{H}'_t)$; the step-level credit-assignment error accumulates linearly in K (Jiang et al., 2026):

$$\mathbb{E}\big[\|B_e^{(K)}\|_2\big] \simeq \varepsilon_e \cdot K \cdot \mathbb{E}\big[\|\nabla_\theta \log \pi_\theta(a_e \mid s_e)\|_2\big], \tag{14}$$

where $B_e^{(K)}$ is the cumulative advantage-baseline bias at an evicted-token position e after K eviction events occur along the trajectory, and ε_e is the per-event bias scale (the misfit between $V(\mathbf{H}_t)$ and $V(\mathbf{H}'_t)$ attributable to one evicted span).

❸ Mirror symmetry and the conditioning invariant. Pitfall A conditions on too little context, whereas Pitfall B conditions on too much (Table 5, App. C.4). Accordingly, on an untrained Qwen3-4B, Δ_{comp} is predominantly negative and Δ_{full} predominantly positive across three white-box harnesses (Figure 2). Under eviction-heavy compression, the Naive-Compressed train–inference log-probability gap reaches 0.366 on AgentFold, $\approx 26\times$ the no-compression floor of 0.014 (§5). Both failures violate the same requirement. In the notation of §2.1, the loss prefix is $c_t^{\text{train}} = \mathbf{p_s}(y_t)$ for Pitfall A and $\mathbf{H}_{<t}$ for Pitfall B, against $c_t^{\text{rollout}} = \mathbf{H}'_t$ in both cases.

Definition 2 (Conditioning consistency)**.** A training pipeline is *conditioning-consistent* if, at every step t in every trajectory, using only the edits fired through step t, never future ones:

$$\forall t, \quad c_t^{\text{train}} \;\equiv\; c_t^{\text{rollout}} \;\equiv\; \textsc{View}_t\big(\mathbf{H}_{\le t};\, \mathcal{E}_{\le t}\big) \;=\; \mathbf{H}'_t\,, \qquad \textit{(Conditioning Invariant)}$$

Proposition 1 (Failure of naive training)**.** *$\mathbf{H}^{\text{comp}}[:t]$ yields Pitfall A, while $\mathbf{H}_{<t}$ yields Pitfall B; neither matches the live view $\mathbf{H}'_t$ from which y_t was decoded. Whenever a nontrivial edit changes the prefix used to evaluate some target y_t, both Naive-Compressed and Naive-Full violate the conditioning invariant. §3 gives two exact (bias-zero) fixes, while §4 gives a soft $\mathcal{O}(\sqrt{\varepsilon_{\text{KL}}})$ one.*

Table 5 summarizes this mirror symmetry; concrete per-leaf examples include a $\Delta_{\text{comp}} = -22.8$-nat leaf for Pitfall A and a $\Delta_{\text{full}} = +18.5$-nat leaf for Pitfall B, both on an untrained Qwen3-4B.

Table 5: Two symmetric failure modes of the same conditioning invariant.

	Naive-Compressed (A)	Naive-Full (B)
Training input	$\mathbf{H}^{\text{comp}}$ (rightmost path)	$\mathbf{H}_T$ (full DFS)
Prefix relation	$c^{\text{train}} \subsetneq c^{\text{infer}}$	$c^{\text{train}} \supsetneq c^{\text{infer}}$
Failure mode	time-travel leakage	train–inference mismatch
Directional signature	c too short	c too long
Intuition	"train saw less than infer"	"train saw future info"
Fixable by	any tree-consistent method	any tree-consistent method

For Qwen3-4B under live model-triggered compression (§5, Table 1), the mismatch is directly visible in the `logdiff` statistic: the No-compression (Search-R1) control remains at 0.014 (numerical noise — with nothing compressed, the live view equals the physical history), whereas Naive-Compressed rises to 0.024–0.059 on average across the three editors and peaks at 0.23–0.29 on eviction-heavy batches ($\sim$20$\times$ the baseline), scaling monotonically with eviction density: the direct empirical signature of $c^{\text{train}} \neq c^{\text{infer}}$, present from the very first steps of training.

Correct gradients. Under the invariant, both the supervised loss and the policy gradient condition uniformly on $\mathbf{H}'_t$ — the *observation* of the editor-induced POMDP (§C.4): the rollout and training policies in the importance ratio must share this observation, and the advantage baseline must approximate $V(\mathbf{H}'_t)$, not $V(\mathbf{H}_t)$. A fix must therefore restore $\nabla_\theta \widehat{\mathcal{L}_{\text{task}}}(\theta) = \nabla_\theta \mathcal{L}^{\star}_{\text{task}}(\theta)$ exactly (§3) or up to a quantifiable $\mathcal{O}(\sqrt{\varepsilon_{\text{KL}}})$ bias at the loss level (§4).

C.5 Trajectory forests under sub-agent calls

Sub-agent calls are ordinary harness behavior, but they break the single payload-prefix chain assumed by one trajectory tree. We therefore represent one execution as a forest $\mathfrak{F} = \{\mathcal{T}^{(0)}, \ldots, \mathcal{T}^{(M)}\}$: one component for the parent and one for each child invocation. A dispatch and its return connect components as execution edges; they are not eviction junctions. Child-generated tokens remain owned by the child component, while the returned result is an observation in the parent component. Flattening the entire conversation into one tree would therefore assign the parent the wrong conditioning prefix and token provenance. The construction is recursive: child m may itself dispatch child n, creating a nested component $\mathcal{T}^{(m,n)}$, and this nesting may continue to arbitrary depth. Each result returns only to its immediate caller as an observation, so a deep or unbalanced call graph remains a forest of separately rooted token histories rather than a single flattened trajectory.

Recovery is unchanged within every component: consecutive payloads recover local live views and local junctions as in Figure 7. LogitTree replays each component's views, and SDCC applies its student–teacher consistency only at local diverging leaves. Returned child text is carried context, not caller loss; the same component-wise rule therefore applies inductively to nested calls. The trainable leaf chains from one forest share a single `rollout_id`, so parent and child chains are batched together without being counted as independent rollouts.

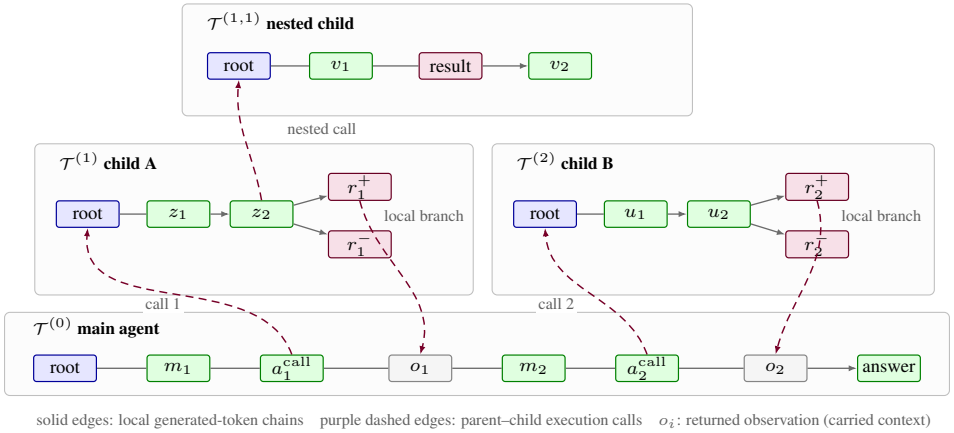

Figure 5: **Sub-agent calls form a forest, not one flattened tree.** The parent component can dispatch multiple sibling children, and a child can dispatch another child. Each component has its own root and local branches; the dashed edges record calls and returns at the execution level. LogitTree and SDCC recover and process each component tree locally, while returned child text is carried into the parent as observation rather than receiving parent loss.

D PROOFS

We proceed bottom-up: the per-branch tree decomposition (§D.1) is the primitive, the packed Logit construction (§D.2) reproduces it in one masked forward, and their equivalence (§D.3) follows. Every proof opens with a one-line sketch. Throughout, $\mathbf{p_t}(y_t) = \mathbf{H}'_t$ is the live-view root-to-leaf prefix of leaf y_t in the conditioning tree $\mathcal{T}$ of definition 1, and $\mathbf{p_s}(y_t) = \mathbf{H}^{\text{comp}}[:t]$ its prefix on the final compressed walk.

D.1 PER-BRANCH EQUIVALENCE ON THE TREE (PROPOSITION 2)

Proposition 2 (SFT and RL equivalence on the tree)**.** *Let $\mathcal{L}_{\text{SFT}}^{\text{tree}}$ be the summed cross-entropy over the $K{+}1$ per-branch forwards, each loss-masked to its own branch. Then $\nabla_\theta \mathcal{L}_{\text{SFT}}^{\text{tree}} = \sum_\pi \nabla_\theta \mathcal{L}_{\text{SFT}}^{\star}(\mathbf{H}'^{\pi})$. For RL, the importance ratio, advantage baseline, and KL regularizer are computed on the same $\mathbf{H}'^{\pi}$ during rollout and update.*

Proof. Sketch: the $K{+}1$ branches partition the loss-carrying leaves, so regrouping the per-leaf loss by branch is an identity, not an approximation.

Each y_t lies on exactly one branch $\pi(y_t)$ — the one live when it was produced — so $\mathbf{H}'^{\pi(y_t)}[:t] = \mathbf{p_t}(y_t)$, and the per-branch loss masks are disjoint and jointly exhaustive on the response. Regrouping $\sum_t -\log P_\theta(y_t \mid \mathbf{p_t}(y_t))$ by branch returns $\mathcal{L}_{\text{SFT}}^{\text{tree}}$ with no leaf double-counted or missed, and the finite sum commutes with ∇_θ. For Proposition 2, the importance ratio, advantage weight and KL regularizer at y_t are all functions of $P_\theta(\cdot \mid \mathbf{p_t}(y_t))$, which the branch-$\pi$ forward evaluates under exactly the rollout-time conditioning. □

D.2 ONE PACKED FORWARD REPRODUCES THE TREE (PROPOSITION 3)

Proposition 3 (SFT and RL equivalence for the Logit mask)**.** *Suppose (i) M^{logical} exactly encodes logical visibility, (ii) position ids are reassigned for each query row, (iii) loss and attention masks are decoupled, (iv) no normalization crosses mask boundaries, and (v) rollout and training share the same M^{logical}. Then, under dense softmax attention, $\nabla_\theta \mathcal{L}_{\text{SFT}}^{\text{Logit}} = \nabla_\theta \mathcal{L}_{\text{SFT}}^{\star}$, and the importance ratio and KL regularizer are computed under identical conditioning during rollout and update.*

Proof. Sketch: a masked query row and its image in the compact sequence attend to the same keys at the same relative offsets, hence produce the same logits; induction over layers and the shared loss mask lift this to gradient equality.

Let Edit delete $E_t \subset \mathbf{H}_t$ and let M implement M^{logical} — the $(T \times T)$ mask of §3 admitting key j in row i iff j was live when y_i was decoded — so $M_{ij} = -\infty$ for $j \in E_t$, $j < i$, and causal otherwise. By (iii) it suffices to prove $\mathbf{Logits}_i(\mathbf{H}_t, M) = \mathbf{Logits}_i(\mathbf{H}'_t)$ for $i \notin E_t$. Write i', j' for the images of i, j under the deletion, so the visible set $V_i := \{j < i : j \notin E_t\}$ maps onto the prefix of the compact sequence $\mathbf{H}'_t$. Row i mixes only over V_i by construction, and by (ii) the reassigned position ids give (i, j) the same relative offset as (i', j'); since RoPE — or any relative-positional encoding — depends only on that offset, $q_i^\top k_j = q_{i'}^\top k_{j'}$, so softmax weights and mixed values agree. Condition (iv) closes the only remaining channel of difference, per-token operations reading across mask boundaries (a layer norm reduced over an unmasked axis, MoE routing pooling evicted positions), so the residual stream at row i equals the compact-forward stream at i'. Induction over layers carries this to the unembedding, and the shared loss mask turns logit equality into the gradient identity. □

D.3 ❸ The centerpiece: Logit masks and LogitTree are the same walk

Theorem 1 (Materialization equivalence)**.** *Under dense softmax attention, the packed Logit mask and the LogitTree K-forward decomposition implement the same traversal of the conditioning tree and compute identical training gradients. Both compute the sum over branches of the per-branch gradient in Proposition 2 (App. D).*

Cost and deployment splits. LogitTree requires $K+1$ backward passes per rollout, resulting in a 5–20× wall-clock overhead in deep-search settings, whereas the Logit formulation requires only one. The Logit formulation, however, imposes a stronger systems requirement: it requires a larger attention mask and white-box access to both the harness and the model. Implementing equation 7 requires the harness to expose the per-query live sets V_i, as well as the model to accept a custom $(Batchsize, NumHeads, T, T)$ attention mask. Neither capability is generally available through black-box APIs such as Claude Code or OpenCode. Moreover, constructing and applying a per-query Logit mask has high memory and computational complexity, making it particularly unfriendly for training with very long contexts. By contrast, LogitTree requires only the live views exposed during rollout and can reconstruct the $K+1$ branch inputs by concatenating the corresponding per-leg prompts. Its main limitation is therefore computational rather than architectural. The concrete Logit Mask realization used in §5 is deferred to App. F.2, a packaging within the equivalence class of equation 7 that inherits Theorem 1.

Proof. Sketch: both materializations are the same scalar function of θ, and identical functions have identical gradients.

By proposition 3 each construction computes $\mathbf{Logits}_i(\mathbf{H}'_t)$ at every loss-carrying position under the identical loss mask; they differ only in cost (union mask: one attention call with a $(B, 1, T, T)$ structured mask; segmented walk: $K+1$ separate forwards). Both losses are therefore the *same scalar function of θ* — the leaf sum $-\sum_t \log P_\theta(y_t \mid \mathbf{p_t}(y_t))$ of proposition 2 — so their gradients coincide up to floating-point association order. □

E Full Variational Derivation of the SDCC Forward-KL Self-Distillation Loss

This appendix derives the SDCC objective (§4, Eq. equation 10) as a forward-KL conditioning-consistency (self-distillation) regularizer, using amortized variational inference as a scaffold. The correctness of SDCC does not rest on the ELBO being tight: it rests on the zero set of the KL residual being exactly the conditioning invariant (§E.3), and on that residual controlling deployment behavior (§E.4). §4 gives the main-text algorithmic view; here we state the assumptions and give the proofs.

Why the forward direction. The slack in the ELBO is a *reverse* KL, and descending it directly is the wake-phase update of amortized inference (Hinton et al., 1995; Bornschein & Bengio, 2015). At the sequence level that requires sampling whole continuations from the *compressed* context: the resulting score-function estimator is weighted by $\log(q_\theta/p_\theta)$, so its variance is unbounded on the surplus side, while the deficit side — which is Pitfall A itself — is almost never drawn from the compressed proposal and is therefore invisible to the update. SDCC instead descends the forward-KL (sleep-

phase) surrogate, which samples from the well-behaved teacher side, has a plain bounded-variance cross-entropy gradient, and, as §E.3 shows, has exactly the same zero set. The four subsections follow that substitution: §E.1 fixes the probabilistic model, §E.2 derives the bound the KL comes from, §E.3 shows the forward direction loses nothing, and §E.4 bounds what a nonzero residual costs at deployment.

E.1 Probabilistic model and the conditioning invariant

Fix a diverging leaf position $p \in \mathcal{C}$. All quantities below are tied to this particular p; sums over $\mathcal{C}$ are taken at the end. Write $\mathcal{A}(p) = \{k : \iota(E_k) \le p \le e(E_k)\}$ for the evictions active at p. The common case is $|\mathcal{A}(p)| = 1$; when several evictions are active the teacher context re-inserts every span in $\mathcal{A}(p)$ and nothing below changes.

Random variables.

- **Clean context** $z_p := \mathbf{p_t}(y_p)$: the pre-eviction prefix that was live at rollout time and that the harness will *again* deliver at deployment time. It is an *externally supplied* conditioning variable, never marginalized out: at inference the harness passes it directly to the policy.
- **Compressed context** $x_p := \mathbf{p_s}(y_p) \subsetneq z_p$: the post-eviction prefix produced by the memory editor, which has removed every span in $\mathcal{A}(p)$.
- **Action** $y \in \mathcal{V}$: the next generated token.
- **Outcome** $y^\star$, with likelihood $p(y^\star \mid y)$: an oracle target for y. We take the RL reading as primary, in which $\log p(y^\star \mid y) = r(y)/\tau$ is the exponentiated-advantage log-likelihood of control-as-inference (Levine, 2018); the SFT reading, $\log p(y^\star \mid y) = \log \mathbf{1}[y = y^\star]$, is its $\tau \to 0$ limit.[1]

Distributions.

$$p_\theta(y \mid z_p) := P_\theta\big(\cdot \mid \mathbf{p_t}(y_p)\big) \qquad \text{deployed / target policy (teacher forward),} \tag{15}$$

$$q_\theta(y \mid x_p) := P_\theta\big(\cdot \mid \mathbf{p_s}(y_p)\big)_p \qquad \text{amortized recognition policy (student forward),} \tag{16}$$

$$p(y^\star \mid y) \qquad \text{outcome likelihood given action.} \tag{17}$$

The two policies share parameters θ; the amortization is *parametric*, not architectural. The conditioning invariant of §C.4 is the statement that this amortization is exact,

$$q_\theta(\cdot \mid x_p) \;=\; p_\theta(\cdot \mid z_p) \qquad \forall\, p \in \mathcal{C}. \tag{CI}$$

equation CI is not a modeling assumption — it is the condition we want to *achieve* through training, and everything that follows is an argument that the SDCC penalty achieves it.

E.2 The ELBO and where the forward KL comes from

This subsection does three things. It derives the variational bound whose slack is the conditioning gap and shows that slack to be an exact KL divergence; it identifies the multiplier λ of equation 10 as the β of a β-VAE free energy, which fixes both its interpretation and its failure mode at large values; and it shows why that slack cannot be descended in the direction the bound presents it in — the fact that forces the forward-KL substitution justified in §E.3.

Proposition 4 (Single-step ELBO for the amortized policy)**.** *For every θ and every diverging leaf $p \in \mathcal{C}$,*

$$\log p_\theta(y^\star \mid z_p) \;\ge\; \mathop{\mathbb{E}}_{y \sim q_\theta(\cdot \mid x_p)}\big[\log p(y^\star \mid y)\big] \;-\; D_{\mathrm{KL}}\Big(q_\theta(\cdot \mid x_p) \,\Big\|\, p_\theta(\cdot \mid z_p)\Big), \tag{18}$$

with equality iff $q_\theta(\cdot \mid x_p) = p_\theta(\cdot \mid y^\star, z_p)$ almost everywhere.

[1] The hard-indicator limit is degenerate as a variational target: whenever q_θ places mass off $y^\star$ the bound equation 18 reads $-\infty \ge -\infty$ and is vacuously satisfied. Every statement below is made at finite τ, where the expressions remain finite.

Proof. Marginalizing the action gives the evidence $p_\theta(y^\star \mid z_p) = \sum_y p(y^\star \mid y)\, p_\theta(y \mid z_p)$. Multiplying and dividing by $q_\theta(y \mid x_p)$, rewriting the sum as a q_θ-expectation and applying Jensen to the concave logarithm,

$$\begin{aligned} \log p_\theta(y^\star \mid z_p) \;&=\; \log \mathop{\mathbb{E}}_{y\sim q_\theta}\left[\frac{p(y^\star \mid y)\, p_\theta(y \mid z_p)}{q_\theta(y \mid x_p)}\right] \;\geq\; \mathop{\mathbb{E}}_{y\sim q_\theta}\left[\log \frac{p(y^\star \mid y)\, p_\theta(y \mid z_p)}{q_\theta(y \mid x_p)}\right] \\ &=\; \mathop{\mathbb{E}}_{y\sim q_\theta}\big[\log p(y^\star \mid y)\big] \;-\; D_{\mathrm{KL}}\big(q_\theta(\cdot \mid x_p) \,\big\|\, p_\theta(\cdot \mid z_p)\big), \end{aligned} \tag{19}$$

which is equation 18. The importance-weighting step needs the proposal $q_\theta(\cdot \mid x_p)$ to dominate the integrand; this is automatic here, since both distributions are softmax outputs of the same network and so have full support on $\mathcal{V}$ (up to numerical underflow). The reverse domination — $p_\theta(\cdot \mid z_p)$ dominating $q_\theta(\cdot \mid x_p)$ — is *not* needed for the bound, and is precisely what fails for the reverse-KL estimator below.

For the equality condition, observe that the Jensen gap is itself a KL divergence. Bayes' rule at fixed z_p reads $p_\theta(y \mid y^\star, z_p) = p(y^\star \mid y)\, p_\theta(y \mid z_p)/p_\theta(y^\star \mid z_p)$, so $\log p_\theta(y^\star \mid z_p) = \log p(y^\star \mid y) + \log p_\theta(y \mid z_p) - \log p_\theta(y \mid y^\star, z_p)$ for every y in the support. The left-hand side does not depend on y, so it is unchanged by taking the q_θ-expectation of the right-hand side; adding and subtracting $\mathbb{E}_{q_\theta}[\log q_\theta(y \mid x_p)]$ and regrouping the three resulting expectations gives the *exact* decomposition

$$\begin{aligned} \log p_\theta(y^\star \mid z_p) \;=\; &\underbrace{\mathop{\mathbb{E}}_{q_\theta}\big[\log p(y^\star \mid y)\big] - D_{\mathrm{KL}}\big(q_\theta(\cdot \mid x_p) \,\big\|\, p_\theta(\cdot \mid z_p)\big)}_{\text{the bound } equation\ 18} \\ &+\quad \underbrace{D_{\mathrm{KL}}\big(q_\theta(\cdot \mid x_p) \,\big\|\, p_\theta(\cdot \mid y^\star, z_p)\big)}_{\text{tightness gap, against the outcome-conditioned posterior}}\quad . \end{aligned} \tag{20}$$

The residual term is non-negative by Gibbs' inequality — which re-derives equation 18 without invoking Jensen — and vanishes iff $q_\theta(\cdot \mid x_p) = p_\theta(\cdot \mid y^\star, z_p)$ a.e. □

Reading the bound. The first term of equation 18 is the (negated) task loss evaluated under the compressed conditioning that training actually sees; the second is the *conditioning gap* between the compressed and pre-eviction views, and it is the only term that references z_p. Decomposition equation 20 is worth isolating because it separates two quantities that are easy to conflate in the main text: the conditioning gap, which SDCC exists to close, and the tightness gap $D_{\mathrm{KL}}(q_\theta(\cdot \mid x_p) \| p_\theta(\cdot \mid y^\star, z_p))$ against the *outcome-conditioned* posterior, which SDCC never touches and does not need to. §E.3 makes that separation precise.

The multiplier λ is the β of a β-VAE. Nothing in equation 18 fixes the relative weight of its two terms. Treating the conditioning gap as a *constraint* rather than a fixed-weight penalty — maximize the task evidence subject to $\sum_{p\in\mathcal{C}} D_{\mathrm{KL}} \leq \delta$ — and forming the Lagrangian returns equation 10 exactly, with λ the multiplier and the constant $\lambda\delta$ discarded. In β-VAE terms (Higgins et al., 2017), writing $\beta := \lambda$ and using the task loss as the distortion,

$$\mathcal{F}_\beta(\theta) \;=\; \underbrace{\mathcal{L}_{\mathrm{task}}(\theta; \mathbf{H}^{\mathrm{comp}})}_{\text{distortion}} + \beta \sum_{p\in\mathcal{C}} \underbrace{D_{\mathrm{KL}}\big(p_{\mathrm{sg}(\theta)}(\cdot \mid z_p) \,\big\|\, q_\theta(\cdot \mid x_p)\big)}_{\text{rate}}, \tag{21}$$

with $\beta = 1$ the unit-weight sleep-phase surrogate and $\beta = 0$ Naive-Compressed. The distortion–rate tension is genuine here precisely because $x_p \subsetneq z_p$: distortion is minimized by exploiting the compressed tokens, rate by making the student's output independent of them, and any task-relevant content of $z_p \setminus x_p$ must be recovered through the parameters rather than read off the input. Pushing β too far therefore reproduces the classical failure mode.

Proposition 5 (Collapse regime at large β)**.** *Let θ^{coll} be a parameter at which the student output is functionally independent of x_p and equals the teacher, $q_{\theta^{\mathrm{coll}}}(\cdot \mid x_p) = p_{\mathrm{sg}(\theta^{\mathrm{coll}})}(\cdot \mid z_p)$ for all $p \in \mathcal{C}$. Then the rate term of equation 21 vanishes at θ^{coll} while its distortion term is generically positive, and for every $\beta > \big(\mathcal{L}_{\mathrm{task}}(\theta^{\mathrm{coll}}; \mathbf{H}^{\mathrm{comp}}) - \mathcal{L}_{\mathrm{task}}(\theta^\star; \mathbf{H}^{\mathrm{comp}})\big)/\varepsilon^\star$ — where $\varepsilon^\star > 0$ lower-bounds the rate at a task-optimal $\theta^\star$ — one has $\mathcal{F}_\beta(\theta^{\mathrm{coll}}) < \mathcal{F}_\beta(\theta^\star)$, so descent on $\mathcal{F}_\beta$ from an uninformed initialization is attracted to the collapsed solution.*

Proof. The rate vanishes at θ^{coll} by construction. The distortion is positive because a student that ignores x_p cannot fit a task loss evaluated on $\mathbf{H}^{\mathrm{comp}}$. Substituting both into equation 21 gives $\mathcal{F}_\beta(\theta^{\mathrm{coll}}) = \mathcal{L}_{\mathrm{task}}(\theta^{\mathrm{coll}}; \mathbf{H}^{\mathrm{comp}})$, while $\mathcal{F}_\beta(\theta^\star) \geq \mathcal{L}_{\mathrm{task}}(\theta^\star; \mathbf{H}^{\mathrm{comp}}) + \beta\varepsilon^\star$; the stated threshold is therefore a sufficient crossing condition. This is β-VAE posterior collapse (Alemi et al., 2018; Razavi et al., 2019). □

Prop. 5 is why λ carries a warm-up schedule and the target a stop-gradient rather than being a free hyperparameter: KL annealing holds $\beta = 0$ until task-loss SGD has moved off θ^{coll}, and a frozen (or EMA) target keeps it from tracking the student into collapse. Both are the standard β-VAE remedies, and both are analyzed in §E.6.

Why the slack cannot be descended in the direction it appears. The remaining step is the direction of the KL. The obvious move is to descend the term of equation 18 as written — the *wake-phase* update of amortized inference (Hinton et al., 1995; Bornschein & Bengio, 2015). Because the gradient must then pass through the sampling distribution, its estimator is a score function reweighted by a log-ratio.

Lemma 1 (Wake-phase gradient, and the variance of its estimator)**.** *Let* $L^{\mathrm{wake}}(\theta) := D_{\mathrm{KL}}\big(q_\theta(\cdot \mid x_p) \,\|\, p_{\mathrm{sg}(\theta)}(\cdot \mid z_p)\big)$ *with the target branch frozen, and write* $g_\theta(y) := \nabla_\theta \log q_\theta(y \mid x_p)$ *and* $\rho_\theta(y) := q_\theta(y \mid x_p)/p_{\mathrm{sg}(\theta)}(y \mid z_p)$*. Then*

$$\nabla_\theta L^{\mathrm{wake}}(\theta) \;=\; \mathop{\mathbb{E}}_{y\sim q_\theta}\big[\log \rho_\theta(y)\, g_\theta(y)\big], \tag{22}$$

and its N-sample Monte-Carlo estimator has variance

$$\mathrm{Var}\big[\widehat{\nabla_\theta L^{\mathrm{wake}}}\big] \;=\; \frac{1}{N}\Big(\mathop{\mathbb{E}}_{q_\theta}\big[(\log \rho_\theta)^2\, \|g_\theta\|^2\big] - \big\|\mathop{\mathbb{E}}_{q_\theta}\big[\log \rho_\theta\, g_\theta\big]\big\|^2\Big). \tag{23}$$

Proof. $L^{\mathrm{wake}}(\theta) = \sum_y q_\theta(y \mid x_p) \log \rho_\theta(y)$, and only q_θ carries a gradient, so $\nabla_\theta L^{\mathrm{wake}} = \sum_y \nabla_\theta q_\theta(y) \log \rho_\theta(y) + \sum_y \nabla_\theta q_\theta(y)$. The log-derivative identity $\nabla_\theta q_\theta = q_\theta\, g_\theta$ turns the first sum into equation 22, and the second is $\nabla_\theta \sum_y q_\theta(y) = \nabla_\theta 1 = 0$. Equation equation 23 is then the variance of a mean of N i.i.d. copies of $X := \log \rho_\theta(y)\, g_\theta(y)$, namely $(\mathbb{E}\,\|X\|^2 - \|\,\mathbb{E}\,X\|^2)/N$. □

Identity equation 23 is exact — it uses no $\log \rho \approx \rho - 1$ small-ratio expansion — and it exposes two failure modes, one on each side of the eviction:

- **Surplus side: unbounded variance.** A token $y^\dagger$ the compressed context finds plausible ($q_\theta(y^\dagger \mid x_p) \geq q_0 > 0$) but the pre-eviction context does not ($p_{\mathrm{sg}(\theta)}(y^\dagger \mid z_p) \to 0$) — a redundant re-query that looks reasonable only once the retrieved span is gone — sends $\log \rho_\theta(y^\dagger) \to +\infty$, and the second moment in equation 23 grows like $(\log \rho_\theta(y^\dagger))^2$. No variance bound holds uniformly over Θ.
- **Deficit side: sampling blindness.** A token the pre-eviction context supports ($p_{\mathrm{sg}(\theta)}(y^\dagger \mid z_p) \geq p_0 > 0$) but the compressed one has lost ($q_\theta(y^\dagger \mid x_p) \to 0$) is Pitfall A itself. Here the variance does *not* blow up, since $q \log^2 q \to 0$, and that is exactly the problem: the summand $q_\theta(y^\dagger) \log \rho_\theta(y^\dagger) g_\theta(y^\dagger) \to 0$, so the update draws no signal from the one token whose conditioning it is meant to repair, and an N-sample estimator needs $N = \Omega\big(1/q_\theta(y^\dagger)\big)$ draws to observe it even once.

Both are properties of the sequence-level Monte-Carlo estimator rather than of the reverse KL as a function: a per-token vocabulary sum evaluates L^{wake} exactly and has neither pathology. What survives even in that exact form is the reverse direction's zero-forcing bias, which under-covers precisely the modes an eviction removes. Two substitutions therefore separate equation 18 from equation 10 — the KL is taken forward rather than reverse, and its right argument is frozen by a stop-gradient — and §E.3 justifies both, showing that the forward direction samples from the well-behaved side, has a bounded-variance gradient, and retains the same zero set.

E.3 WHY THE FORWARD KL SUFFICES

Two things must hold before the ELBO's reverse KL may be replaced by the forward KL that SDCC minimizes: the reverse-KL slack must vanish exactly at the conditioning invariant, so that the ELBO points at the right target; and the forward KL must have the same zero set, so that the substitution loses nothing.

The slack is the conditioning gap, and is weaker than tightness.

Proposition 6 (Slack-zero implication). $D_{\mathrm{KL}}\big(q_\theta(\cdot \mid x_p) \,\|\, p_\theta(\cdot \mid z_p)\big) = 0$ *if and only if* $q_\theta(\cdot \mid x_p) = p_\theta(\cdot \mid z_p)$ *a.e., i.e. if and only if the per-leaf conditioning invariant equation CI holds. This is* strictly weaker *than tightness of equation 18, which by Prop. 4 additionally requires the recognition policy to equal the* outcome-conditioned *posterior* $p_\theta(\cdot \mid y^\star, z_p)$ *rather than the outcome-marginal* $p_\theta(\cdot \mid z_p)$.[2]

Proof. Gibbs' inequality gives the first equivalence: a KL divergence is zero iff its two arguments coincide almost everywhere. The second claim is the equality condition of Prop. 4. □

The ELBO is therefore scaffolding rather than the object of interest: driving its slack to zero closes the amortization gap between the two conditioning distributions *without* tightening the bound, and SDCC's justification needs only the former.

The sleep-phase surrogate. Wake–sleep (Hinton et al., 1995) flips the sampling direction: instead of sampling from q_θ and fitting it to p_θ, it samples from the frozen $p_{\mathrm{sg}(\theta)}$ and fits q_θ by maximum likelihood on those samples.

Proposition 7 (The sleep-phase update is a forward KL and recovers equation 10). *Let* $L^{\mathrm{sleep}}(\theta) := \mathbb{E}_{y \sim p_{\mathrm{sg}(\theta)}(\cdot \mid z_p)}\big[-\log q_\theta(y \mid x_p)\big]$. *Then*

$$L^{\mathrm{sleep}}(\theta) \;=\; D_{\mathrm{KL}}\Big(p_{\mathrm{sg}(\theta)}(\cdot \mid z_p) \,\Big\|\, q_\theta(\cdot \mid x_p)\Big) \;+\; H\big(p_{\mathrm{sg}(\theta)}(\cdot \mid z_p)\big), \tag{24}$$

with $H(\cdot)$ *the Shannon entropy. Since the stop-gradient makes the entropy* θ*-independent,* $\nabla_\theta L^{\mathrm{sleep}} = \nabla_\theta D_{\mathrm{KL}}\big(p_{\mathrm{sg}(\theta)} \,\|\, q_\theta\big) = -\,\mathbb{E}_{y \sim p_{\mathrm{sg}(\theta)}}[\nabla_\theta \log q_\theta(y \mid x_p)]$*, a plain cross-entropy score whose* N*-sample Monte-Carlo estimator has variance at most* $\mathbb{E}_{y \sim p_{\mathrm{sg}(\theta)}} \|\nabla_\theta \log q_\theta(y \mid x_p)\|^2 / N$ *— no* q/p *re-weighting appears. Summing over* $p \in \mathcal{C}$ *and adding the task loss recovers the boxed SDCC objective equation 10 verbatim:*

$$\mathcal{L}_{\mathrm{SDCC}}(\theta) \;=\; \mathcal{L}_{\mathrm{task}}(\theta; \mathbf{H}^{\mathrm{comp}}) \;+\; \lambda \sum_{p \in \mathcal{C}} D_{\mathrm{KL}}\Big(\underbrace{p_{\mathrm{sg}(\theta)}(\cdot \mid z_p)}_{\textit{teacher, stop-grad}} \,\Big\|\, \underbrace{q_\theta(\cdot \mid x_p)}_{\textit{student, with-grad}}\Big). \tag{25}$$

Proof sketch. Add and subtract $\log p_{\mathrm{sg}(\theta)}(y \mid z_p)$ inside the expectation defining L^{sleep} and identify the two resulting expectations as a KL divergence and an entropy; that is equation 24. Differentiating with the target held fixed leaves the score expectation stated, whose estimator variance is its second moment; the bound is finite because q_θ has full support (softmax) and no ratio-based re-weighting inflates the summand. For the last claim, sum equation 24 over $p \in \mathcal{C}$, discard the θ-independent entropy sum, weight the KL sum by λ, add the task loss, and substitute $q_\theta(\cdot \mid x_p) = P_\theta(\cdot \mid \mathbf{p_s}(y_p))_p$ and $p_{\mathrm{sg}(\theta)}(\cdot \mid z_p) = P^{\mathrm{target}}(\cdot \mid \mathbf{p_t}(y_p))$ with the notation of §4. □

The two directions have the same zero set.

Theorem 2 (Zero SDCC residual ⇔ conditioning invariant). *Let* $\varepsilon_p(\theta) := D_{\mathrm{KL}}\big(p_{\mathrm{sg}(\theta)}(\cdot \mid z_p) \,\|\, q_\theta(\cdot \mid x_p)\big)$ *be the per-leaf forward-KL residual that SDCC minimizes. Then*

$$D_{\mathrm{KL}}\big(p_{\mathrm{sg}(\theta)} \,\big\|\, q_\theta\big) = 0 \quad\Longleftrightarrow\quad p_{\mathrm{sg}(\theta)} = q_\theta \;\; \textit{a.e.} \quad\Longleftrightarrow\quad D_{\mathrm{KL}}\big(q_\theta \,\big\|\, p_{\mathrm{sg}(\theta)}\big) = 0, \tag{26}$$

and consequently $\sum_{p \in \mathcal{C}} \varepsilon_p(\theta) = 0$ *if and only if the conditioning invariant equation CI holds at every diverging leaf — in which case the ELBO's reverse-KL slack of Prop. 6 is zero as well.*

[2] The two targets coincide only when $p(y^\star \mid y)$ is y-independent on the support of $p_\theta(\cdot \mid z_p)$, i.e. when the outcome carries no discriminative signal — a degenerate configuration, and not the regime of interest here.

Proof. Both equivalences in equation 26 are Gibbs' inequality: a KL divergence vanishes iff its two arguments agree almost everywhere, and that condition is symmetric in the two arguments even though the two divergences are not. A sum of non-negative terms is zero iff every term is, which lifts the per-leaf statement to $\mathcal{C}$. □

Away from the zero set the two divergences are different functions of θ, and the forward direction is the one that can actually be descended in a single-backward training loop: the frozen teacher supplies the target distribution, the gradient flows only through the student, and the update is an ordinary cross-entropy rather than a $\log(q_\theta/p_\theta)$-weighted score-function estimator whose variance diverges exactly on the tokens the eviction damaged. Full ELBO tightness (Prop. 4) is strictly stronger than equation 26 and is neither targeted nor required.

Existence of a zero-KL solution. The residual can be driven to zero at all only if the model can reproduce its pre-eviction conditional from the compressed prefix alone.

Proposition 8 (Existence of a zero-KL solution)**.** *If model capacity admits $\theta^\star$ with $P_{\theta^\star}(\cdot \mid \mathbf{p_s}(y_p))_p = P_{\theta^\star}(\cdot \mid \mathbf{p_t}(y_p))$ for all $p \in \mathcal{C}$, then the SDCC KL loss attains 0. This holds when the evicted span is redundant with the surviving trunk, and fails when the span carries a strictly necessary bit absent from $\mathbf{H}^{\text{comp}}$, in which case ε_p stays positive and SDCC is bias-dominated relative to the exact methods.*

Proof. Each per-leaf term $\varepsilon_p(\theta)$ is non-negative, so $K(\theta) = \sum_{p\in\mathcal{C}} \varepsilon_p(\theta) \geq 0$ with equality iff student and teacher coincide on every diverging leaf (theorem 2); whenever capacity admits such a $\theta^\star$ the zero set Θ_K is non-empty and the loss attains 0. This is existence only — that SGD reaches it is theorem 4 in section E.6, at rate $\mathcal{O}(1/\sqrt{T})$ to the noise floor and exactly 0 under the decaying-step-size schedule of corollary 3. □

E.4 What the residual KL guarantees

In finite training the residual $\varepsilon_p(\theta)$ is small but nonzero, so what matters is not the zero set of theorem 2 but the chain *small KL ⇒ small total variation ⇒ small deployment deviation ⇒ small policy-gradient bias*. This subsection states that chain once. Its first link is unconditional; only the last requires bounded advantage and score.

Proposition 9 (Behavioral Pinsker bound on the conditioning gap)**.** *For every diverging leaf $p \in \mathcal{C}$,*

$$\big\|p_\theta(\cdot \mid z_p) - q_\theta(\cdot \mid x_p)\big\|_{\text{TV}} \;\leq\; \sqrt{\varepsilon_p(\theta)/2}. \tag{27}$$

Proof. Pinsker's inequality (Tsybakov, 2009) states that $\|\mu - \nu\|_{\text{TV}} \leq \sqrt{D_{\text{KL}}(\mu\|\nu)/2}$ for any two distributions on a common measurable space. Take $\mu = p_{\text{sg}(\theta)}(\cdot \mid z_p)$ and $\nu = q_\theta(\cdot \mid x_p)$, whose divergence is exactly the per-leaf SDCC residual $\varepsilon_p(\theta)$; at convergence the stop-gradient branch equals p_θ and equation 27 follows. No bounded-score, same-advantage or parameterization assumption enters, so the bound is unconditional. Because the harness redelivers z_p at inference, the left-hand side is literally the deployment gap: the behavioral distance between the context the update was computed under and the one the deployed policy is served. □

Assumption 1 (Bounded advantage and context-stable score)**.** *(A1) $\|\hat{A}\|_\infty \leq A_{\max}$; (A2) the same behavioral advantage $\hat{A}(y)$ is used on both $\mathbf{p_t}(y_p)$ and $\mathbf{p_s}(y_p)$; (A3), for the parameter block under study, the score $s_c(y) := \nabla_\theta \log \pi_\theta(y \mid c)$ is uniformly bounded by $2H_{\max}$ and is stable across the paired contexts in the sense that, for $\pi := \pi_\theta(\cdot \mid \mathbf{p_t}(y_p))$ and $\widetilde{\pi} := \pi_\theta(\cdot \mid \mathbf{p_s}(y_p))$, $\mathbb{E}_{y\sim\widetilde{\pi}} \|s_{\mathbf{p_t}}(y) - s_{\mathbf{p_s}}(y)\| \leq 2H_{\max}\|\pi - \widetilde{\pi}\|_{\text{TV}}$; (A4) advantages are averaged over an outer batch.*

Corollary 1 (Conditional gradient-bias bound)**.** *Under assumption 1, the SDCC per-leaf policy-gradient deviation is $\mathcal{O}(\sqrt{\varepsilon_p})$; in aggregate, with $C := 6A_{\max}H_{\max}$,*

$$\Big\|\nabla_\theta \mathcal{L}_{\text{task}}(\theta; \mathbf{H}^{\text{comp}}) - \nabla_\theta \mathcal{L}^\star_{\text{task}}(\theta)\Big\| \;\leq\; C \sum_{p\in\mathcal{C}} \sqrt{\varepsilon_p/2} \;=\; \mathcal{O}\Big(\textstyle\sum_{p\in\mathcal{C}} \sqrt{\varepsilon_p}\Big), \tag{28}$$

where $\mathcal{L}^\star_{\text{task}}$ is the tree-consistent task loss that Logit masks and LogitTree compute exactly. Writing $\varepsilon_{\text{KL}} := \max_p \varepsilon_p$, this is the per-leaf $\mathcal{O}(\sqrt{\varepsilon_{\text{KL}}})$ form quoted in §E.4.

Proof. Sketch: split the two policy gradients into a distribution-difference term and a score-difference term; the boundedness and context-stability clauses of (A3) control them in total variation, which proposition 9 converts into $\sqrt{\varepsilon_p}$.

Write $\pi := \pi_\theta(\cdot \mid \mathbf{p_t}(y_p))$, $\widetilde{\pi} := \pi_\theta(\cdot \mid \mathbf{p_s}(y_p))$, $s_\pi(y) := \nabla_\theta \log \pi(y)$, $g := \sum_y \pi(y)\hat{A}(y)s_\pi(y)$, and $\tilde{g}$ analogously with the same $\hat{A}$ by (A2). Then

$$\begin{aligned}\|g - \tilde{g}\| &\le \Big\|\textstyle\sum_y \big(\pi(y) - \widetilde{\pi}(y)\big)\hat{A}(y)\, s_\pi(y)\Big\| + \Big\|\textstyle\sum_y \widetilde{\pi}(y)\hat{A}(y)\big(s_\pi(y) - s_{\widetilde{\pi}}(y)\big)\Big\| \\ &\le 2A_{\max}H_{\max}\,\|\pi - \widetilde{\pi}\|_1 \;+\; A_{\max}\,\mathop{\mathbb{E}}_{\widetilde{\pi}}\big\|s_\pi - s_{\widetilde{\pi}}\big\|. \end{aligned} \tag{29}$$

Total variation is half the ℓ_1 distance, so by (A1), (A3) and proposition 9 the first term is at most $4A_{\max}H_{\max}\sqrt{\varepsilon_p/2}$. By the context-stability clause of (A3), the second is at most $2A_{\max}H_{\max}\|\pi - \widetilde{\pi}\|_{\mathrm{TV}} \le 2A_{\max}H_{\max}\sqrt{\varepsilon_p/2}$. Summing over per-leaf residuals and per-trajectory tokens as in (A4) gives equation 28. □

Remark 1 (Scope of the gradient corollary). proposition 9 is unconditional; only corollary 1 invokes assumption 1, where (A2) is the standard PPO/GRPO advantage-identifiability convention and (A3) is an explicit context-stability assumption on the score. Bounded softmax logits alone do not imply this condition, even for the last layer, because the hidden representation depends on context. For full-transformer parameters it can instead be established from a uniform Lipschitz/Jacobian bound on the backbone. The policy-gradient conclusion is therefore conditional on (A3), whereas the behavioral Pinsker bound is not.

Corollary 2 (Rate to zero bias under training)**.** *Under Assumptions (i)–(iv) of §E.6, Theorem 4 yields* $\min_{t\le T}\mathbb{E}[\varepsilon_p(\theta_t)] = \mathcal{O}(1/\sqrt{T})$*; combined with corollary 1, the deployment policy-gradient bias at the corresponding best iterate decays at the rate* $\mathcal{O}(T^{-1/4})$.

Cost, and where SDCC sits relative to the exact methods. The bound above is what SDCC buys with its cost profile. SDCC requires exactly one backward pass per trajectory against LogitTree's $K+1$; the teacher forwards share KV-cache trunks and are stop-gradient, so they retain no activations. A Logit mask is cheaper still in pure FLOPs — one forward, one backward — but requires attention-kernel surgery and cannot be applied in black-box harnesses where eviction spans are never exposed. The three methods therefore trade off along two independent axes, correctness bias and infrastructure cost: the exact walks have zero bias at heavy cost, while SDCC pays $\mathcal{O}(\sqrt{\varepsilon_{\mathrm{KL}}})$ bias — decaying during training by corollary 2 — for one backward pass and no kernel changes. The claim is not that SDCC dominates the exact methods: where their cost is acceptable, LogitTree and the Logit mask are the correctness ground truth; where it is not, or in black-box regimes, SDCC is the only one of the three that applies.

E.5 LogitTree and SDCC optimize the same objective up to a λ-tunable slack

The exact walks of Section 3 and SDCC of Section 4 charge a diverging leaf $p \in \mathcal{C}$ in visibly different ways: LogitTree places the policy loss on the pre-eviction branch $z_p = \mathbf{p_t}(y_p)$, so its per-leaf loss is an expectation under the teacher distribution $p_\theta(\cdot \mid z_p)$; SDCC keeps the policy loss on the compressed branch $x_p = \mathbf{p_s}(y_p)$, so its per-leaf loss is an expectation under the student distribution $q_\theta(\cdot \mid x_p)$, augmented with a stop-gradient forward KL that penalizes the two distributions for disagreeing (§4). The two constructions therefore differ in *which* branch of $\mathcal{T}$ enters the policy loss — LogitTree scores every branch (Prop. 2); SDCC scores only the deployed branch and charges the others through the KL. Because the KL residual controls the behavioral gap between the two branches (proposition 9), the two losses can be made arbitrarily close by choosing λ; this subsection makes that statement precise as a two-sided bound.

Fix θ and $p \in \mathcal{C}$; write $\varepsilon_p := \varepsilon_p(\theta)$. Following the paper's per-leaf normalization convention, let

$$\mathcal{L}_p^{\mathrm{LT}}(\theta) \;:=\; -\mathop{\mathbb{E}}_{y\sim p_\theta(\cdot|z_p)}\big[\hat{A}(y)\big], \qquad \mathcal{L}_p^{\mathrm{SDCC}}(\theta) \;:=\; -\mathop{\mathbb{E}}_{y\sim q_\theta(\cdot|x_p)}\big[\hat{A}(y)\big] + \lambda\,\varepsilon_p$$

denote the per-diverging-leaf contributions of the two objectives; on non-diverging leaves $z_p = x_p$, both reduce to the identical single-branch term, so a sum-over-$\mathcal{C}$ suffices. $\hat{A}$ is the same behavioral advantage on both branches (Assumption 1, A2), and no rollout re-sampling appears.

Theorem 3 (Two-sided objective equivalence between LogitTree and SDCC)**.** *Under Assumption 1(A1), for every θ, every $p \in \mathcal{C}$, and every $\lambda > 0$,*

$$\left|\mathcal{L}_p^{\mathrm{SDCC}}(\theta) - \mathcal{L}_p^{\mathrm{LT}}(\theta) - \lambda\,\varepsilon_p\right| \;\le\; A_{\max}\sqrt{2\,\varepsilon_p}. \tag{30}$$

Consequently, Young's inequality ($A_{\max}\sqrt{2\varepsilon_p} \le A_{\max}^2/(2\lambda) + \lambda\varepsilon_p$) yields a θ-uniform lower bound and a θ-tracking upper bound,

$$\mathcal{L}_p^{\mathrm{LT}}(\theta) - \frac{A_{\max}^2}{2\lambda} \;\le\; \mathcal{L}_p^{\mathrm{SDCC}}(\theta) \;\le\; \mathcal{L}_p^{\mathrm{LT}}(\theta) + 2\lambda\,\varepsilon_p + \frac{A_{\max}^2}{2\lambda}, \tag{31}$$

and summing over $p \in \mathcal{C}$ gives $\mathcal{L}^{\mathrm{LT}}(\theta) - |\mathcal{C}|A_{\max}^2/(2\lambda) \le \mathcal{L}^{\mathrm{SDCC}}(\theta) \le \mathcal{L}^{\mathrm{LT}}(\theta) + 2\lambda\sum_p \varepsilon_p + |\mathcal{C}|A_{\max}^2/(2\lambda)$. On the zero set $\{\theta : \varepsilon_p(\theta) = 0\ \forall p \in \mathcal{C}\}$ of theorem 2, equation 30 holds with equality at 0, so LogitTree and SDCC coincide exactly as functions of θ; the relaxed form equation 31 still carries its $A_{\max}^2/(2\lambda)$ slack there, Young's inequality being loose at $\varepsilon_p = 0$.

Proof. Rewrite the difference of the two policy-loss terms as a change of measure and apply Hölder:

$$\big(\mathop{\mathbb{E}}_{q_\theta(\cdot|x_p)} - \mathop{\mathbb{E}}_{p_\theta(\cdot|z_p)}\big)[\hat{A}(y)] \;=\; \sum_y \big(q_\theta(y \mid x_p) - p_\theta(y \mid z_p)\big)\hat{A}(y) \;\le\; \|\hat{A}\|_\infty \cdot \|q_\theta - p_\theta\|_1, \tag{32}$$

and analogously for the reverse sign. By Assumption 1(A1) the leading factor is at most $A_{\max}$; total variation is half the ℓ_1 distance and proposition 9 bounds it by $\sqrt{\varepsilon_p/2}$. Together, $|(\mathbb{E}_{q_\theta} - \mathbb{E}_{p_\theta})[\hat{A}]| \le A_{\max}\sqrt{2\varepsilon_p}$. Substituting into $\mathcal{L}_p^{\mathrm{SDCC}} - \mathcal{L}_p^{\mathrm{LT}} = (\mathbb{E}_{p_\theta} - \mathbb{E}_{q_\theta})[\hat{A}] + \lambda\varepsilon_p$ gives equation 30. Young's inequality supplies $A_{\max}\sqrt{2\varepsilon_p} = 2\sqrt{(A_{\max}^2/(2\lambda)) \cdot (\lambda\varepsilon_p/1)} \le A_{\max}^2/(2\lambda) + \lambda\varepsilon_p$; adding and subtracting this on the two sides of equation 30 yields the clean form equation 31. On the zero set every ε_p vanishes, hence $p_\theta(\cdot \mid z_p) = q_\theta(\cdot \mid x_p)$ by theorem 2 and the change-of-measure term equation 32 is 0, so equation 30 is an equality between zeros. □

What the bound says, and where it stops. Read in the maximization convention $J := -\mathcal{L}$, the lower bound of equation 31 becomes $J^{\mathrm{SDCC}}(\theta) - |\mathcal{C}|A_{\max}^2/(2\lambda) \le J^{\mathrm{LT}}(\theta)$: SDCC's objective, offset by a θ-*independent* constant, is a *lower bound on the exact per-branch objective*, so maximizing it raises that bound at every θ and at every residual ε_p, with the offset shrinking as the KL weight λ grows. Equivalently, in the loss convention $\mathcal{L}^{\mathrm{SDCC}}$ plus that constant majorizes $\mathcal{L}^{\mathrm{LT}}$, so minimizing SDCC minimizes an upper bound on the exact walk's loss. The upper bound is looser — it retains a $\lambda\sum_p \varepsilon_p$ term — because SDCC pays the extra KL even when the exact walk does not; on the CI zero set that term is zero, so the two objectives coincide there. The classical VAE correspondence of App. E.2 reappears: the same λ that makes the lower bound tight is the β whose *too-large* regime triggers posterior collapse (Prop. 5), so λ cannot be sent to infinity naively. Two consequences follow. (i) The two objectives share a common global minimizer set on the CI zero set of theorem 2, and their gradients on that set agree (corollary 1 restates the away-from-zero version under the same assumptions); this is the precise sense in which the two objectives are "essentially equivalent". (ii) The gap is *not zero in general*: it scales as $\lambda\varepsilon_p$ on the upper side, which is precisely the main-text trade-off in §4: SDCC buys single-backward compute at a residual that is θ-controlled by the KL weight rather than structurally zero, and this residual is what theorem 4 contracts at $\mathcal{O}(1/\sqrt{T})$.

E.6 Convergence: contraction toward the zero-KL set

Setup and assumptions. With $\theta_t \in \Theta \subset \mathbb{R}^d$ the step-t parameters, $\mathbf{H}^{\mathrm{comp}}$ the compressed student input (§4) and $(\mathbf{H}'_t, \mathbf{H}^{\mathrm{comp}}) \sim \pi$, the task loss is $J(\theta) \triangleq \mathbb{E}_\pi[\mathcal{L}_{\mathrm{task}}(\theta; \mathbf{H}^{\mathrm{comp}})]$ and the SDCC penalty is

$$K(\theta) \;\triangleq\; \mathop{\mathbb{E}}_{\pi}\Big[\textstyle\sum_{t\in\mathcal{C}} D_{\mathrm{KL}}\big(P_{\mathrm{sg}(\theta)}(\cdot \mid \mathbf{H}'_t) \,\big\|\, P_\theta(\cdot \mid \mathbf{H}^{\mathrm{comp}})_t\big)\Big], \tag{33}$$

the stop-gradient making K depend on θ only through the update branch. The composite objective is $F_t = J + \lambda_t K$, with step $\theta_{t+1} = \theta_t - \eta_t(\widehat{\nabla} J + \lambda_t \widehat{\nabla} K)(\theta_t)$ and schedule $\lambda_t = 0$ for $t < T_{\mathrm{warm}}$, $\lambda_t \to \lambda^\star > 0$ after. The post-warm-up Lyapunov function $\Phi \triangleq (J - J^\star) + \lambda^\star K$, $J^\star := \inf_\theta J$, is non-negative, zero at a joint minimizer, and L_Φ-smooth with $L_\Phi \le L_J + \lambda^\star L_K$. We assume throughout ((i)–(iv) are distinct from (A1)–(A4) of Assumption 1):

(i) *Smoothness:* J, K are L_J-, L_K-smooth.
(ii) *Bounded gradient noise:* $\mathbb{E}\,\|\widehat{\nabla} J\|^2 \le \sigma_J^2$, $\mathbb{E}\,\|\widehat{\nabla} K\|^2 \le \sigma_K^2$.
(iii) *Joint realizability:* $\{\theta : K(\theta) = 0\}$ is nonempty (Prop. 8) and meets $\arg\min_\theta J$; fix $\theta^\star$ in the intersection, so $\Phi(\theta^\star) = 0$. This adds to Prop. 8 the capacity condition that conditioning-invariance and task-optimality be jointly attainable.
(iv) *Composite Polyak–Łojasiewicz:* $\|\nabla\Phi\|^2 \ge 2\mu_\Phi \Phi$ near $\theta^\star$ for some $\mu_\Phi > 0$; stronger than PL on K alone, as it excludes cancellations $\nabla J \approx -\lambda^\star \nabla K$ at $\Phi > 0$.

Only (iv) is restrictive: near $\theta^\star$ both Hessians are Fisher matrices, hence PSD, and PSD alone does not imply PL. It holds when Φ is locally strongly convex along descent-relevant directions (e.g. the bounded-score last-layer softmax of Cor. 1); under PSD only, dropping the PL step of Theorem 4 still leaves $\min_t \mathbb{E}\,\|\nabla\Phi(\theta_t)\|^2 \to 0$, forfeiting the linear rate on Φ (hence K) but not convergence to a critical point.

Warm-up. For $t < T_{\mathrm{warm}}$ training is pure task-loss SGD, ε-stationary in $\mathcal{O}(\sigma_J^2/\varepsilon^2)$ steps under (i)–(ii) (Ghadimi & Lan, 2013); as the collapsed "ignore everything" solution is non-stationary for J on non-degenerate tasks, warm-up moves off collapse before the KL is switched on, decoupling cold-start collapse from steady-state alignment. The linear ramp used in practice replaces $\lambda^\star$ by its running average, inflating the constants below by at most 2.

Steady-state contraction. Proofs here are sketches, each naming the standard step it invokes.

Theorem 4 (Steady-state contraction toward zero-KL set)**.** *Under **(i)–(iv)** with μ_Φ the constant of (iv), constant step size $\eta = \min(1/L_\Phi,\, c/\sqrt{T})$, and T post-warm-up SGD steps (distinct from the rollout-end T of §2), let $\Sigma^2 := 2\sigma_J^2 + 2(\lambda^\star)^2\sigma_K^2$. Then*

$$\min_{t\le T} \mathbb{E}\big[\Phi(\theta_t)\big] \;\le\; \Phi(\theta_{T_{\mathrm{warm}}})\big/(\mu_\Phi \eta T) \;+\; \big(L_\Phi \eta/2\mu_\Phi\big)\Sigma^2, \tag{34}$$

an initial-gap decay plus a noise floor. As $\Phi \ge \lambda^\star K$ pointwise, equation 34 over $\lambda^\star$ bounds $\min_{t\le T}\mathbb{E}[K(\theta_t)] = \mathcal{O}(1/\sqrt{T})$ at $\eta = c/\sqrt{T}$.

Proof sketch. L_Φ-smoothness with (ii) gives the standard descent step $\mathbb{E}\,\Phi(\theta_{t+1}) \le \Phi(\theta_t) - \frac{\eta}{2}\|\nabla\Phi(\theta_t)\|^2 + \frac{\eta^2 L_\Phi}{2}\Sigma^2$ for $\eta \le 1/L_\Phi$, using $\|a+b\|^2 \le 2\|a\|^2 + 2\|b\|^2$ rather than assuming the two same-batch estimators are uncorrelated; telescope, divide by T, apply (iv). Only (iv) is non-routine: PL on K alone would *not* suffice, as ∇J and $\lambda^\star\nabla K$ may cancel, while (iii) forces $J - J^\star$ and K to zero together. □

EMA targets. With $\theta^-_{t+1} = \alpha\theta^-_t + (1-\alpha)\theta_{t+1}$ replacing $\mathrm{sg}(\theta_t)$, (θ_t, θ^-_t) is a two-timescale stochastic approximation (Borkar, 1997): differencing the updates and applying discrete Grönwall gives $\mathbb{E}\,\|\theta^-_t - \theta_t\|^2 \le 2\eta_0\Sigma^2/(1-\alpha)$ whenever $\eta_t \le \eta_0$ and $1-\alpha \le \eta_0$. As the target KL is $\mathcal{O}(\|\theta^-_t - \theta_t\|^2)$ near $\theta^\star$ in the Fisher metric, Theorem 4 extends to EMA targets with its noise floor inflated by $\mathcal{O}(\eta/(1-\alpha))$, vanishing as $\eta \to 0$. Composing it with the behavioral Pinsker bound of Prop. 9 turns the KL rate into the deployment TV bias against exact-walk (LogitTree / LogitMask) conditioning, quoted without proof in §E.4.

Corollary 3 (Conditioning bias rate)**.** *Under **(i)–(iv)**, let $\widehat{t} \in \arg\min_{t\le T}\mathbb{E}[\Phi(\theta_t)]$. The corresponding SDCC policy $\pi_{\theta_{\widehat{t}}}$ satisfies*

$$\mathbb{E}\big\|\,\pi_{\theta_{\widehat{t}}}(\cdot\,|\,\mathbf{H}'_t) - \pi_{\theta_{\widehat{t}}}(\cdot\,|\,\mathbf{H}^{\mathrm{comp}})\big\|_{\mathrm{TV}} \;\le\; \mathcal{O}\big(T^{-1/4}\big) + \mathcal{O}\big(\sqrt{\eta\,\Sigma^2/\lambda^\star}\big), \tag{35}$$

an optimization term plus a noise floor, both driven to zero by $\lambda^\star = \Theta(1)$ and $\eta_t = \Theta(1/\sqrt{t})$.

Adding Assumption 1 and replacing Prop. 9 by Cor. 1 lifts equation 35 to a policy-gradient bias rate with the same $\mathcal{O}(T^{-1/4})$ scaling; without that assumption the TV bound still holds unconditionally.

PART III EXPERIMENT SUPPLEMENTS: IMPLEMENTATION, HARNESS KNOBS, METRIC SEMANTICS, TRAINING CURVES

This part supports §5 at the level of detail needed to reproduce every measured entry of Table 1 from a fresh container: the rollout–training contract shared by every cell of the matrix, the three

white-box harnesses (TC-RAG's `pop()`, AgentFold's periodic `fold`, MemexRL's `memory(op)`), the two black-box harnesses (Claude Code, OpenCode), and the five training-method implementations in the SLIME stack (§F); then the reproduction knobs, the full per-harness aggregation of the Q1 offline probe, what `logdiff` does and does not certify under live compression (§G.4), and the training curves with SDCC's eviction-density profile (§G).

F IMPLEMENTATION OF THE HARNESSES AND THE TRAINING METHODS

This appendix documents *how* each of the five harnesses edits the live context at rollout time, and *how* each of the five training methods materializes its training input from the resulting record. It complements §G, which lists the settings needed for reproduction (optimizer schedule, retriever backend, per-method and per-harness settings, evaluation protocol); here the emphasis is on the mechanisms — what data structure each component maintains, what it emits, and where the tree of §2 (formalized in §C) physically resides in the code. All components run inside the SLIME RL stack (Megatron actor + SGLang rollout engine).

Provenance and acknowledgment. Our white-box implementation builds on the SLIME example stack (Zhu et al., 2025), which already provided a TITO-style (Gallouédec & Rasul, 2026) loss hook sketching the per-turn segmented objective—one forward per response segment, scored against a reconstructed rollout-time prefix. We reuse that scaffold and its interfaces, and thank its authors. What this paper adds is the conditioning-tree formulation, the equivalence between explicit branch materialization and the packed Logit mask, the SDCC relaxation, the rollout-side splitting that makes exact replay executable, and the measurements of Table 1—not the idea of replaying tokens under the prefix that produced them.

F.1 THE ROLLOUT–TRAINING CONTRACT

One record per rollout. Every harness, whether white-box or black-box, reduces to one per-rollout record that all five methods consume identically:

$$\underbrace{\mathbf{H}^{\mathrm{comp}}}_{\text{compressed walk}}, \qquad \underbrace{\mathbf{H}_T}_{\text{physical union}}, \qquad \underbrace{\{(J_k,\, E_k)\}_{k=1}^{K}}_{\text{eviction records}},$$

where J_k is the physical position at which the k-th compression fired and E_k is the set of physical positions it removed (Definition 1). The compressed walk $\mathbf{H}^{\mathrm{comp}}$ is exactly the token sequence the rollout engine rendered for the final turn, and $\mathbf{H}_T$ is the union of all tokens that ever existed in the live view. The record travels with each training sample, together with the per-token log-probabilities logged by the rollout engine at decode time (the reference side of the `logdiff` diagnostic of §5.2).

The harness bridge. Each white-box harness implements one bridge adapter with two responsibilities: (i) *execute* the edit inside the rollout loop — rewrite the surviving message list before the next decoding step, so that the next request payload sent to the rollout engine is truly the compressed view; and (ii) *log* the edit as an eviction record (J_k, E_k) in token coordinates of $\mathbf{H}_T$. Because the bridge is the only component that knows the harness's internal semantics (stack, fold, memory store), the downstream loss hooks are fully harness-agnostic: they see only the record above.

F.2 HARNESSES AND METHODS

We now describe, at the level of the mechanism rather than the code, how each editor rewrites the live context and how each method reads the resulting record. The harness and the method are selected by two independent switches: any method runs against any white-box harness without per-cell code divergence, which is the engineering property behind the cross-harness sweep of §5.2.

White-box editors. All three white-box editors follow the *learnable-action* route: wherever the original system specifies a compression operation, we expose it to the policy as a first-class tool call, so the compression decision itself is trainable. They differ in trigger and tree shape. *TC-RAG* (a learnable `pop` over a stack of tool-result envelopes): a `pop` removes the oldest envelope entirely, giving few junctions with heavy spans (mean $|E_k| \approx 156$ tokens; Table 3); it is model-triggered, so an

untrained policy leaves $K=0$ and the tree collapses to its trunk. *AgentFold* (length-triggered folding): when the rendered context exceeds 3,000 tokens the policy summarizes the foldable prefix and the prefix is replaced by that summary in the live view; the summary is on-policy and receives gradients, while the folded prefix is logged as E_k (mean $|E_k| \approx 84$). Being length-triggered, it fires even for an untrained policy, which makes it the natural pilot harness. *MemexRL* (an external key–value store with `memory_offload`/`memory_retrieve`): offload evicts a named span, retrieve re-injects it, and a terminal restore pulls all stored entries back before the answer turn. This yields the richest tree — many small-span junctions (mean $|E_k| \approx 41$) and *non-monotone* spans (a token can leave and re-enter), each event logged so the live view is reconstructible by replay. The row-wise live-set definition $j \in V_i$ in Eq. 7 directly represents these non-monotone spans. Per-turn state transitions are illustrated in Figure 6.

Black-box editors. Claude Code and OpenCode are deployed agents whose context management (sliding window + auto-compact) is internal and unobservable: they surface the per-turn rendered transcript but not eviction spans or summary provenance. Running through a separate agent-platform pipeline, we capture per turn the exact request payload the framework sent to the model and convert it into the same per-rollout record. This requires observing every model request, but not the harness's internal span-level log. Junctions are *detected* as prefix divergences between consecutive payloads (the evicted span is not identifiable, so we store the payloads rather than a token-set), and the teacher prefix is taken to be the previous payload. Payload-level LogitTree evaluates these recorded requests as separate branch sequences, while SDCC uses them as teacher–student pairs; neither requires the hidden eviction spans. The Logit mask, by contrast, requires span-level visibility records. Scoring uses the same EM/format reward as the white-box cells.

The five methods. All five methods are standard SLIME loss hooks reading only the one record above; none modifies the forward pass or the attention kernel. *Naive-Compressed* and *Naive-Full* use the reference policy loss unchanged and differ only in the training sequence: Naive-Compressed trains on $\mathbf{H}^{\mathrm{comp}}$ as rendered (conditioning post-junction tokens on later-created summaries, Pitfall A), while Naive-Full re-injects every evicted span to reconstruct $\mathbf{H}_T$ (conditioning on already-evicted content, Pitfall B); both roll out under live compression, so Table 1 isolates the training-time conditioning choice alone. *LogitTree* splits the rollout at its K junctions into $K+1$ sub-samples, each carrying the live view as it existed during that layer and only that layer's generated tokens; SLIME trains them as independent samples ($K+1$ forward/backward passes) with $c_t^{\mathrm{train}} = c_t^{\mathrm{rollout}}$ by construction, the episode reward inherited unchanged. *The Logit mask* reuses that same branch split but folds it into a single forward: the branches are packed into one sequence and the attention mask of Prop. 3 admits, in each query row, exactly the keys that were live when that token was decoded, with position ids restarted per branch so that RoPE phases match the rollout. *SDCC* runs a student forward on $\mathbf{H}^{\mathrm{comp}}$ (with gradients, the Naive-Compressed compute path) plus one gradient-free teacher forward per junction on the reconstructed pre-eviction prefix $\mathbf{H}'_t$ (Eq. 9), and adds the forward-KL $D_{\mathrm{KL}}(\pi_\theta(\cdot \mid \mathbf{H}'_t) \,\|\, \pi_\theta(\cdot \mid \mathbf{H}^{\mathrm{comp}}[:t]))$ only at diverging-leaf positions, with coefficient λ. The teacher shares the student's weights (no second model in memory), and the direction is fixed — student = compressed view, teacher = pre-eviction view, never the reverse — because the student is scored on the compressed replay while the original live conditional is the stop-gradient target (§4; §E).

Cost and applicability. The three consistency-restoring methods pay for the invariant along different axes (Table 6). LogitTree pays in *compute*: K additional backward passes per trajectory, a 5–20× per-step penalty in deep-search or coding regimes (K = 5–20); it is backbone-agnostic and aligns natively with vLLM/SGLang prefix-sharing. The Logit mask needs one forward and one backward, but pays in *infrastructure*: the model must accept an arbitrary per-row mask, position ids must be reassigned row-wise, and mask equivalence has to be re-checked at serving time. SDCC keeps a single backward pass (though, with $\ell_{K+1} = C$, no smaller forward budget) and needs eviction events only to be detectable, so it is the only one of the three that also runs in the black-box case, at the price of an $\mathcal{O}(\sqrt{\varepsilon_{\mathrm{KL}}})$ correctness residual.

Table 6: The three consistency-restoring methods vs. the two pitfalls, across dimensions that matter for production RL stacks, with the per-rollout token budget of §C folded in: $L = |\mathbf{H}_T|$ is the physical union, $C = |\mathbf{H}^{\text{comp}}|$ the compressed walk, and ℓ_i the live view in force during branch layer i, so $C \le \ell_i \le L$ with $\ell_{K+1} = C$. Tokens are not FLOPs (§C).

Dimension	Naive-Comp / Naive-Full	Logit mask	LogitTree	SDCC
Restores conditioning invariant	×	✓ (dense only)	✓ (any backbone)	✓ up to $\mathcal{O}(\sqrt{\varepsilon_{\text{KL}}})$
Forward passes / trajectory	1	1	$K{+}1$	$\le K{+}1$ (K stop-gradient)
Backward passes / trajectory	1	1	$K{+}1$	1
Tokens per trajectory	C / L	L (packed)	$\sum_{i=1}^{K+1} \ell_i \ge L$	$\le C + \sum_{i=1}^{K} \ell_i$
Custom attention kernel required	no	*yes*	no	no
Tree-structured data pipeline	no	no	*yes*	no (leaf mask only)
Position-id reassignment required	no	*yes* (row-wise)	no	no
Needs eviction spans exposed	no	*yes*	*yes* (segmented)	no (divergence suffices)
vLLM / SGLang alignment	native	×	✓	native

G Experiment Details

This appendix is designed to let an external reader reproduce every entry of table 1 and table 9. The training stack is SLIME (Megatron actor + SGLang rollout engine); we give the macroscopic setup below, with pinned versions and every non-default setting shipped in the supplementary code.

G.1 Experimental Settings Details

Compute environment. All runs use the SLIME RL framework (MEGATRON-LM actor + SGLANG rollout engine) on 80 GB A100- or H800-class accelerators with CUDA/PyTorch and FlashAttention; pinned versions and the full dependency list ship with the supplementary code.

Optimizer and schedule. Adam (β_1=0.9, β_2=0.98) with constant learning rate 1×10^{-6}, weight decay 0.01 and gradient clip 1.0; GRPO with group size G=16 samples per prompt, clip range ε=0.2 (low side) / 0.28 (high side) and KL coefficient β=0.001 (low-variance estimator). For SDCC, the consistency coefficient λ ramps linearly from 0 to 0.1 over the first 20% of update steps and stays constant thereafter; we deliberately do not tune this schedule per editor, so that no reported gain is an artifact of hyperparameter sweeping. The headline sweep uses a fixed step budget per cell, whereas the convergence campaign trains each cell to a reward plateau (early stopping once the trailing reward mean stops improving), capped at 200 rollouts.

Rollout. At most T=30 tool-call turns with top-k=3 retrieved chunks per call, rollout temperature 1.0, n=16 samples per prompt and 4 prompts per rollout, giving 64 trajectories per rollout, which are consumed in 8 optimizer steps of 8 trajectories each; advantages are normalized within each 16-sample group (GRPO default), so the group, not the optimizer step, sets the baseline.

Retriever and search backend. All rollouts — during both training and evaluation — issue live web searches rather than querying a local index. Web search is served by the online DashScope text-search API, which returns up to 10 ranked results per query, and full-page evidence is extracted by a two-stage pipeline that fetches the retrieved pages with Firecrawl and summarizes them with Qwen3-Turbo. Because the same endpoint serves training and evaluation, the inference-time search distribution matches the training-time one.

Training corpus. The training corpus pools REDSEARCHER (Chu et al., 2026) and the ASEARCHER agentic-search training set into 81,638 composite QA instances, normalized to a uniform question–answer schema. Its composite, multi-hop questions typically require at least two retrieval rounds, giving the model-triggered editors (TC-RAG `pop`, MemexRL `memory_offload`) and AgentFold's length-triggered fold the opportunity to actually fire (see the eviction-density profile in Table 10).

Evaluation benchmarks. Each trained checkpoint is evaluated on seven heterogeneous held-out open-domain QA benchmarks (38,270 questions per checkpoint in total), all run through the full

agentic loop (multi-turn search, live compression by the cell's harness): NQ (Kwiatkowski et al., 2019) (3,610 questions, single-hop factoid), TRIVIAQA (Joshi et al., 2017) (11,313, single-hop trivia), HOTPOTQA (Yang et al., 2018) (7,405, two-hop), 2WIKIMULTIHOPQA (Ho et al., 2020) (12,576, structured multi-hop), MUSIQUE (Trivedi et al., 2022) (2,417, 2–4-hop compositional), BAMBOOGLE (Press et al., 2023) (125, hand-curated compositional), and FRAMES (Krishna et al., 2025) (824, multi-constraint retrieval-and-reasoning).

Harness knobs. The three white-box harnesses are selected by a single harness switch. All three compress *live*, inside the rollout turn loop — the surviving messages are rewritten before the next decoding step, and the training and evaluation code consume the same compressed view — and all three emit the same eviction-record contract ($\{J_k, E_k\}$), so the downstream loss hook is harness-agnostic. **MemexRL** is model-triggered, via a `memory_offload` / `memory_retrieve` tool pair over a session-local store with a terminal restore before the answer turn (§F.2); **TC-RAG** is model-triggered, keeping a stack of tool-result envelopes with a learnable pop that evicts the oldest and no rule-based schedule (§F.2); **AgentFold** is length-triggered, folding the prefix into a self-generated summary once the live context exceeds 3,000 tokens — the only rule-based trigger, and hence the highest-density editor at cold start (Table 10; §F.2). Complete mechanism descriptions — action spaces, triggers, and what each editor stores and restores — are in §F.2. For the two black-box harnesses (Claude Code, OpenCode) the rollout channel is supplied by a separate integration pipeline outside this codebase; we consume its outputs in the same per-rollout record format, so the SDCC / LogitTree loss hooks are unchanged (§F.2), and the per-cell scoring uses the same internal EM/format reward as the white-box cells, making the black-box column directly comparable.

Evaluation protocol. The reward (M1) is Exact-Match against the gold answer with a 0.2 format bonus for correctly tagged but factually wrong answers (matching Jin et al., 2025); Pass@1 (M2) is measured on the held-out test split with a single rollout at temperature 0; drift (M3) is $D_{\mathrm{KL}}\big(\pi_\theta(\cdot \mid \mathbf{H}'_t) \,\|\, \pi_{\theta,\text{train}}(\cdot \mid c_t^{\text{train}})\big)$ at inference time, averaged over editor activations.

G.2 Q1 OFFLINE PROBE: FULL PER-HARNESS AGGREGATION

The per-token distributions in Figure 2 are plotted from the same 32 synthetic 3-turn rollouts per harness (untrained Qwen3-4B, forward passes only). Table 7 reports the full per-harness aggregation behind that panel: sample counts, means, and majority-sign fractions for both directions of the pitfall Δ, together with the theoretical prediction each row must satisfy for the invariant of §C.4 to hold. Every harness meets the prediction in both directions. The reading is qualitative: leaves are clustered within rollouts and prompts, so the per-token means and $|\Delta|$ magnitudes here should be interpreted as sign-and-scale indicators rather than as unit-independent population estimates. We do not attach confidence intervals to the means for that reason; instead we report the majority-sign fraction per harness as the primary read. The qualitative pattern (aggressive-eviction editors amplify Naive-Comp, high-payload editors amplify Naive-Full-leak) is discussed in the prose of §5.2.

Table 7: **Q1: the two pitfalls are real and directionally distinct.** Untrained Qwen3-4B, 32 synthetic rollouts per harness. Both pitfalls exhibit the predicted sign on all three white-box harnesses. Δ_{comp}: TC-RAG/AgentFold evict most aggressively (larger $|\Delta|$); MemexRL exposes the most leaves ($n_{\text{comp}} = 4000$) because its two-step offload delay keeps each evicted span alive for two physical steps. n_{full} is identical across harnesses (3072) because all three evict before the answer turn. Figure 2 plots the underlying distributions.

	Δ_{comp} (Naive-Comp vs. teacher)			Δ_{full} (Naive-Full-leak vs. current view)		
Harness	n	mean	frac. < 0	n	mean	frac. > 0
MemexRL	4000	-1.62	0.60	3072	$+0.68$	0.56
TC-RAG	928	-3.96	0.78	3072	$+0.24$	0.71
AgentFold	928	-3.81	0.75	3072	$+0.34$	0.55
Predicted		< 0 (under)	$\to 1$		> 0 (over)	$\to 1$

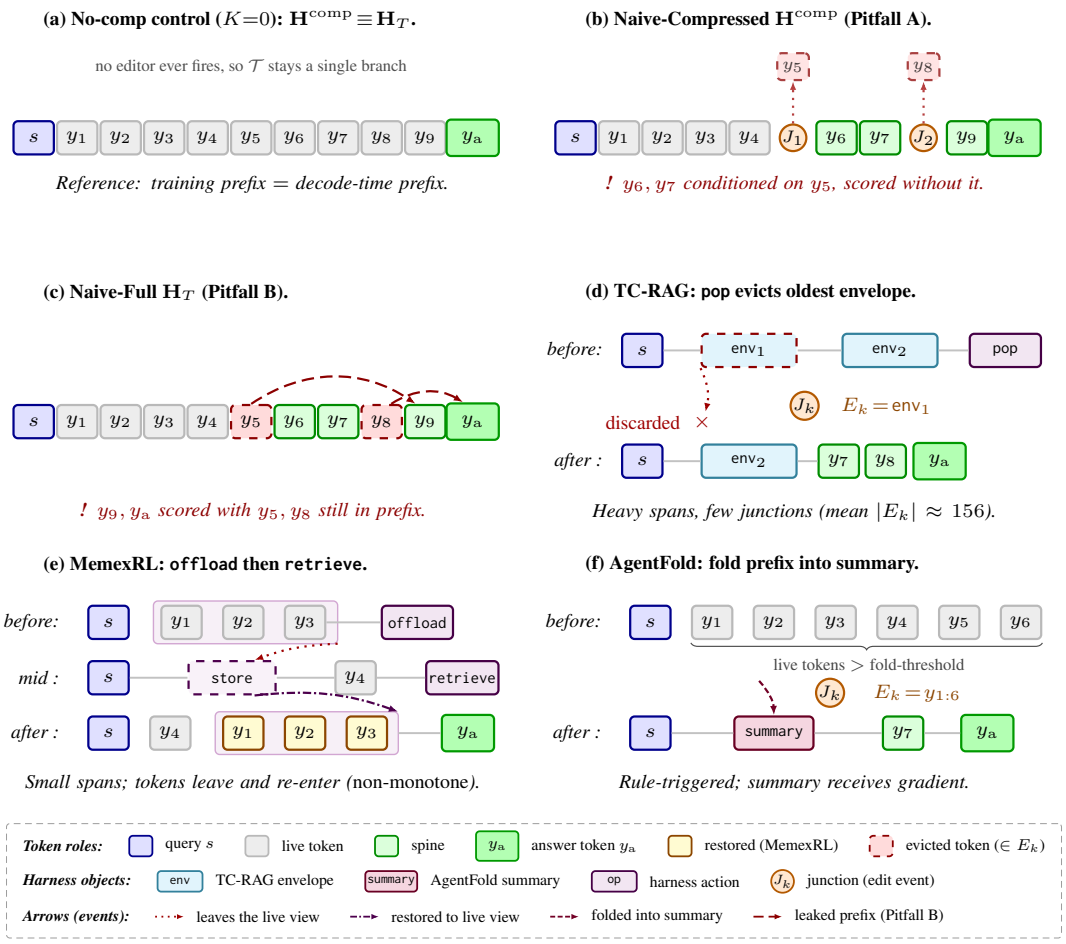

Figure 6: **Six cases on the shared token-node vocabulary of Fig. 2**, whose running example and subscripts these panels reuse; J_3 falls inside the abridged tail y_a. Panels split **by role, not by row**: Part I **(a)–(c)** are training-input recipes on one rollout, exposing the Pitfall A "too-short" and Pitfall B "too-long" mismatches; Part II **(d)–(f)** draw one live compression event per harness (before→after; J_k is generic, indices are panel-local). See §F.2 for harness code.

Table 8: Method inputs from the single adapter record. CC_TOKENS = compressed walk, UN_TOKENS = union.

Method	Student input	Teacher forwards?	Extra bookkeeping
Naive-Compressed	$\mathbf{H}^{\text{comp}}$	no	—
Naive-Full	$\mathbf{H}_T$	no	—
LogitTree (ours)	per-segment slices	no	K segment boundaries
Logit mask (ours)	$\mathbf{H}_T$	no	Logit attn + hole-free posids
SDCC (ours)	$\mathbf{H}^{\text{comp}}$	K (no-grad)	K reconstructed teacher prefixes

G.3 Method-input recipe under the same adapter record

All five methods construct their training input from the same per-rollout adapter record $(\mathbf{H}^{\text{comp}}, \mathbf{H}_T, \{J_k, E_k\}_{k=1}^{K})$, plus a No-comp (Search-R1) control that disables the editor (so $\mathbf{H}^{\text{comp}} \equiv \mathbf{H}_T$ and its drift is the numerical floor). Table 8 summarizes the input, teacher-forward count, and extra bookkeeping. All five methods roll out under *live* compression; Naive-Full re-injects the evicted text into the student input.

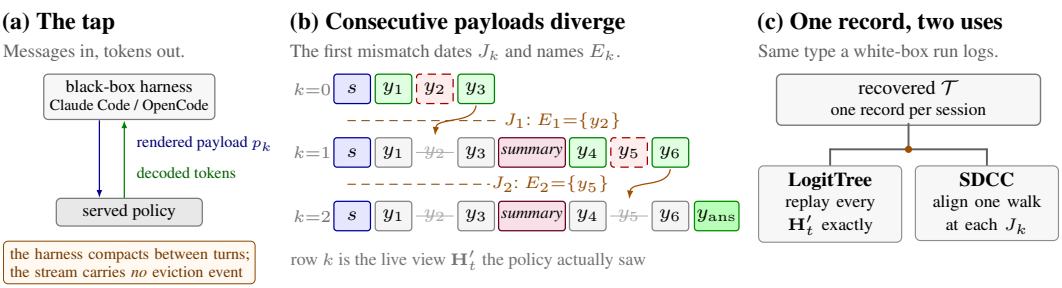

Figure 7: **Recovering the trajectory tree from a black-box harness. (a)** At the model boundary the only observable is one (rendered payload, decoded tokens) pair per turn: the harness compacts its own context between turns and puts no eviction event on the wire. **(b)** Consecutive payloads are laid on a single column grid in the token vocabulary of Figure 2, whose colour key applies here unchanged; the one addition is the purple slot, text the harness wrote for itself, as in Figure 6. What turn k decoded (green) reappears as carried context (grey) in the next payload *except* where the harness dropped it, and a dropped slot leaves no box at all. The first position at which two consecutive payloads disagree therefore dates the junction J_k and names the evicted set E_k (orange band and arrow), so both are recovered from the message stream alone, with no span-level log and no cooperation from the harness. **(c)** Each row of (b) is a live view $\mathbf{H}'_t$, so the recovered $\mathcal{T}$ has the same type as the tree a white-box harness logs, and the LogitTree and SDCC loss hooks consume it unchanged (§F.2).

G.4 What `logdiff` certifies under live compression

Recall the three diagnostics of §5.1: recorded-token `logdiff`, SDCC's training KL $\mathrm{KL}_{\mathrm{SDCC}}$, and agentic EM. This subsection expands on `logdiff`, which compares, per response token, the training-side re-forward under the method's training conditioning with the log-probability recorded by the rollout engine at generation time. Under the live-compression protocol, the latter is conditioned on the *compressed view that the model actually saw at that turn* — so the two sides are directly comparable, and `logdiff` is a faithful per-step measure of the training–rollout conditioning gap for the recorded token. As a diagnostic, it certifies conditioning fidelity of the training forward, not end-task correctness; the efficacy verdict is delivered by EM. The residual gap Naive-Compressed incurs is now precisely identified: a token emitted at turn k was sampled under the turn-k view, which still contained material that a *later* eviction removed; re-forwarding the final compressed walk therefore conditions turn-k tokens on a history under which they were never sampled. LogitTree (K-forward on exact per-turn prompt token ids) and the Logit mask (one packed forward realizing the same conditioning) eliminate this gap *by construction*, and their observed `logdiff` indeed returns to (or within noise of) the no-compression baseline in every cell of Table 1. SDCC deliberately keeps the cheap Naive-Compressed forward (and thus shares its elevated diagnostic on mismatched batches) and closes the gap through the training-KL objective; its progress is read off $\mathrm{KL}_{\mathrm{SDCC}} = \mathrm{KL}(\pi_{\mathrm{teacher}}(\cdot \mid z_p) \,\|\, \pi_{\mathrm{train}}(\cdot \mid x_p))$ (the training loss itself, §4) plus downstream EM. We therefore read the matrix as: `logdiff` ranks conditioning fidelity of the training forward (LogitTree = Logit = baseline $\ll$ Naive-Compressed, by construction for the two exact walks), while SDCC-KL and EM rank end-task correction quality.

G.5 Training curves

This subsection plots the training dynamics behind the grand matrix. The white-box curves come from the 4B RL runs on the pooled training corpus, one curve per method×harness cell; the editor-free control enters this subsection only as a numeric reference in the text. Figure 9 additionally reports reward traces from the Claude Code and OpenCode black-box runs. The curves diagnose optimization behaviour and are not themselves results: Table 1 remains the sole source of final numbers.

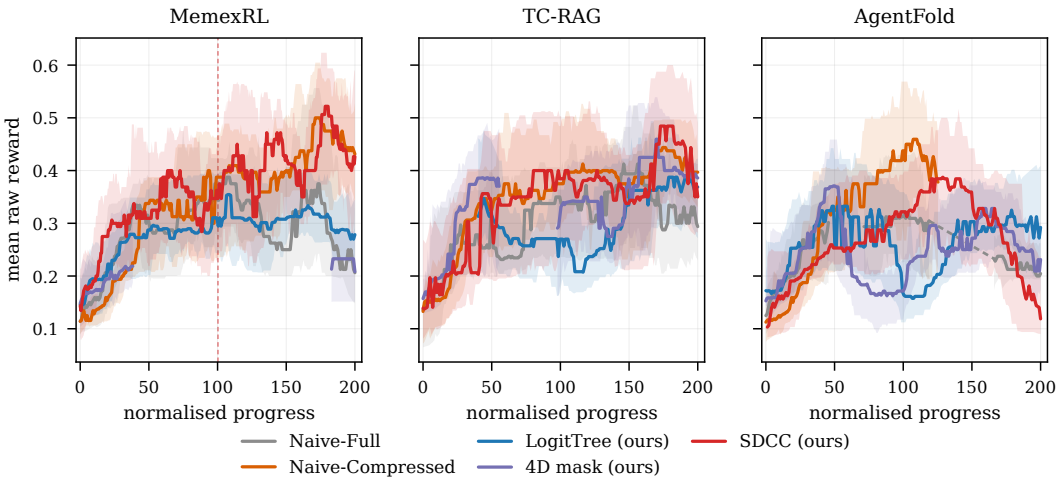

Figure 8: **Mean raw reward over training, by harness.** The axis is normalised progress: each cell's iterations are rescaled by $x = 200\,\mathrm{iter}/\mathrm{iter}_{\max}$, with $\mathrm{iter}_{\max}$ is that cell's last logged iteration, so iteration 0 sits at $x = 0$. One x-unit is therefore a different number of iterations in each curve. Thick lines are a centred rolling median of `rollout/raw_reward` over a ± 15-iteration window, taken on the true iteration grid before the rescale; the band is that window's interquartile range.

Reward. Figure 8 shows all fifteen white-box cells rising over the windows it draws, by between 0.07 and 0.39 from first drawn point to peak, with the per-iteration signal noisy enough that these curves cannot separate methods by eye. We deliberately draw no "best" marker: the logged windows differ in length across cells, and the rescaled axis aligns their endpoints rather than removing that difference, so any endpoint contrast would in part reflect unequal training length rather than method. Note also that `rollout/rewards` is a post-normalization quantity, not a reward: its per-cell median lies between -7.5×10^{-3} and -1.0×10^{-8}, i.e. centred on zero rather than on any reward level, so it appears nowhere in this figure.

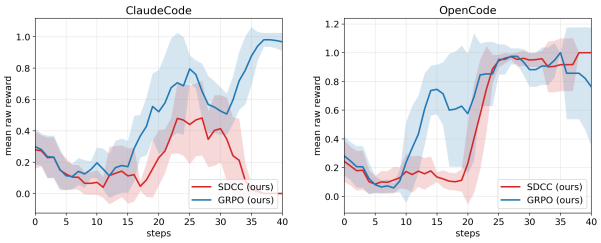

Figure 9: **Black-box reward, first 40 steps.** Mean reward for SDCC (red) and GRPO (blue); bands show run variability.

Figure 9 separately reports the black-box reward traces for Claude Code and OpenCode over their first 40 optimizer steps. These panels use their native step grids rather than the white-box normalized-progress axis. Their noisy, short windows and overlapping variability bands make them descriptive training traces, not a reward-based method ranking.

Conditioning drift. Section G.4 states what `logdiff` certifies; this paragraph states how we measure it. Every number below is a pooled median over *steady* steps, defined by one rule applied uniformly to every cell: within each launch we discard the first 20 optimizer steps and retain only launches with at least 10 steps remaining. The discard is necessary because the statistic reads more than an order of magnitude high until the trainer and the rollout engine have re-aligned; conditioning on the within-launch step index removes most but not all of that, and we report the residual rather than claim it away.

Training length is a second confound, and it is the one that decides the ordering. Because the editor-free control covers iterations up to 58, a median taken over each cell's own window would compare cells trained for different lengths. We therefore fix the comparison window in advance to the window the control covers — steady steps at iteration ≤ 58, against the control's median there (0.0138, n=448). Ten of the fifteen cells sit on the floor at 0.93–1.04× the control; Naive-Full's two eviction-heavy cells sit just above it, both at 1.19×; and three cells are elevated: Naive-

Table 9: **Q2: 5-method comparison, Qwen3-4B / MemexRL.** Drift from the live-compression MemexRL cell; Search-R1 drift from the editor-free control (numerical floor). EM columns are seven-bench macro-EM under the agentic-eval protocol (§G.1); † rows are grand-matrix same-source runs (cross-harness EM in Table 1).

Method	EM ↑	Drift ↓	Cost (× base)
Search-R1 (no-comp.)	$23.0^{\dagger}$	0.0135	1.05
Naive-Compressed	$28.9^{\dagger}$	0.0237	1.00
Naive-Full	$32.1^{\dagger}$	0.0163	1.05
LogitTree (ours)	$45.9^{\dagger}$	**0.0133**	4.20
Logit mask (ours)	$33.4^{\dagger}$	0.0140	1.35
SDCC (ours)	$43.1^{\dagger}$	0.0165	1.55

Compressed×AgentFold $2.17\times$, Naive-Compressed×TC-RAG $2.69\times$ and SDCC×AgentFold $4.51\times$. That is the ordering the main text predicts — the two eviction-heavy harnesses separate Naive-Compressed from the floor and MemexRL does not ($0.99\times$) — and SDCC is elevated by construction, since it keeps the cheap compressed forward and corrects through its KL term. Restricting to the eleven cells that cover the window to within a single iteration leaves the floor band at 0.94–$1.04\times$ and moves no cell across a group boundary.

On the cold-start MemexRL cells — where compression barely fires — all five methods stay within 0.0115–0.0151 over their full windows (Naive-Compressed 0.0133), bracketing the control's 0.0138, exactly as the eviction-density argument of Section G.6 requires. One caution on reading these numbers against the rest of the paper: a median over per-iteration medians, a median pooled over a cell's steady steps, and the drift column of Table 1 are three different estimators of the same quantity, none numerically interchangeable with another. Only the pooled form is used here.

G.6 SDCC TRAINING DYNAMICS AND EVICTION-DENSITY PROFILE

Q2 headline 5-method table. Table 9 provides the 5-method drift/EM comparison on the low-eviction MemexRL anchor cell at 4B; the drift column is measured under the live-compression protocol, and all six daggered EM rows are populated from the same seven-bench agentic runs that supply Table 1. The three ours rows (LogitTree/LogitMask/SDCC) already show the drift ordering; cross-harness EM support appears in Table 1: on the eviction-heavy harnesses, LogitMask matches or exceeds Naive-Compressed EM (ahead 34.2 vs. 33.2 on AgentFold, ahead 37.2 vs. 36.7 on TC-RAG) while its drift stays at the no-compression floor (0.017–0.021 vs. Naive-Compressed's 0.054–0.058), so ours does not trade EM for drift consistency. MemexRL is the cold-start regime ($8/256$ sessions evict), so Naive-Comp drift here is at the *floor* of the eviction-density scaling reported in Table 1 (main text).

Selectivity of the SDCC regularizer under live compression. Figure 10 tracks the on-line evolution of the SDCC regularizer on the AgentFold cell — the eviction-heavy editor (length-triggered fold at 3,000 tokens; compression ratio 0.06–0.15), and hence the cell in which the KL contract has the most work to do. Four regularities emerge.

(i) The KL fires sparsely and precisely. $\mathrm{KL}_{\mathrm{SDCC}}$ is non-zero on slightly over half of the logged optimizer steps and exactly zero elsewhere, and the non-zero steps are exactly those whose batches contain live folds. Conditioning on the gate separates the diagnostic by a factor of 5.6 in median: steps with $\mathrm{KL}_{\mathrm{SDCC}} > 0$ have median `logdiff` 0.081 (IQR 0.058–0.125), whereas steps with $\mathrm{KL}_{\mathrm{SDCC}} = 0$ sit at the no-compression baseline, median 0.015 (IQR 0.014–0.017). The regularizer engages at mismatched positions and only at mismatched positions — the per-leaf gating of §4 operating as designed.

(ii) The contract stays armed for the duration of the run rather than saturating or dying: the diverging-leaf set is non-empty on a majority of micro-batches throughout, so the KL term never becomes structurally inert.

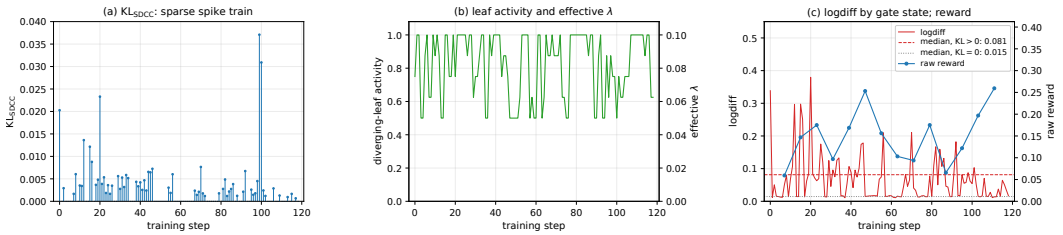

Figure 10: **SDCC dynamics on Qwen3-4B × AgentFold under live compression.** (a) $\mathrm{KL}_{\mathrm{SDCC}}$ is a sparse spike train, non-zero on slightly over half of the logged optimizer steps and exactly zero elsewhere. (b) Diverging-leaf activity and the effective KL weight λ. (c) `logdiff` rises on eviction batches (median 0.081) and sits at the no-compression baseline otherwise (median 0.015); mean reward drifts upward.

(iii) The SDCC KL residual does not grow over the run, while $\mathrm{KL}_{\mathrm{ref}}$ against the reference model grows smoothly from 0.002 to ≈ 0.03: the policy moves but does not detach from the reference, and no run-away is observed without an entropy bonus.

(iv) Mean raw reward rises over the same window in which the training-KL residual does not grow — the training-time consistency signal and the eval-time task signal are not in tension — which motivates the head-to-head against Naive-Compressed in Table 9: absent SDCC's regularizer, the same policy optimizes against physical logits it never trains on. The load-bearing part of that comparison is structural rather than numerical: the two arms share the same policy-loss code path modulo the KL term, i.e. Naive-Compressed $=$ SDCC at $\lambda = 0$ (§5.2). Magnitudes here come from one editor and are not offered as cross-method effect sizes.

This cell is cold-start: the untrained 4B policy triggers evictions in only a fraction of sessions, so the KL magnitudes are small in absolute terms. The point established here is not the magnitude but the *selectivity* — $\varepsilon_{\mathrm{KL}}$ is confined to exactly the diverging leaves, so the $\mathcal{O}(\sqrt{\varepsilon_{\mathrm{KL}}})$ bound of §E.4 is non-vacuous from the first steps of training. Figure 10 plots the co-evolving signals: (a) the sparse KL spike train; (b) the diverging-leaf activity and the effective KL weight; (c) `logdiff` spiking on eviction batches and returning to baseline elsewhere, with reward drifting up.

Per-method mechanism activation under live compression. Table 11 reports, for each method, the run-mean of the quantity that its own corrective mechanism controls, per white-box editor. This is not an outcome (EM) table: it establishes that each mechanism engages at 4B scale under *live*, model-triggered compression, and quantifies how strongly. Two things stand out.

(i) For every editor, each corrective method's own channel is non-zero: LogitTree splits each rollout into $\bar{K} = 3.2$–4.1 segments, the LogitMask packer activates on 1.2–1.4 sub-samples per micro-batch, and SDCC's leaf gate is armed on a large majority of micro-batches. The Naive rows have no such channel by construction.

(ii) SDCC's KL magnitude tracks the editor's eviction density: $\approx 10^{-5}$ on MemexRL (8/256 sessions evict at cold start), 1.1×10^{-4} on TC-RAG, and 2.9×10^{-3} on AgentFold (132/960 sessions fold; compression ratio 0.10). The correction therefore scales precisely with how much the editor actually rewrites — consistent with the per-leaf gating derivation of §4.

Persistence of Pitfall A under training. A natural hope is that the naive-compressed drift will self-correct as GRPO adapts the policy to the compressed prefixes. The AgentFold cell gives no sign of that correction: over the logged window the per-third mean of `logdiff` moves $0.069 \to 0.045 \to 0.064$, and the count of eviction-heavy steps (`logdiff`> 0.05) does not fall — the mismatch neither shrinks nor is absorbed, because its source (folds keep firing on new rollouts) remains stationary. On the same cell, LogitTree's `logdiff` stays pinned at 0.013–0.015 in every third: the exact correction removes the drift at its source rather than asking the optimizer to absorb it. This window is early-training and therefore low-eviction; at the argmax-reward iteration of the full run the same Naive-Compressed $\times$ AgentFold cell reads 0.366 (Table 1). The persistence claim is thus made at the *conservative* end of the density range.

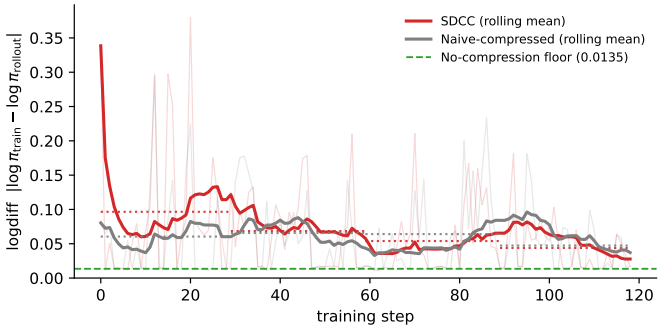

Figure 11: **SDCC's conditioning gap decreases over the logged training window; Naive-Compressed's does not.** Per-step `logdiff` on Qwen3-4B × AgentFold under live compression. SDCC (red) trends toward the no-compression floor (green dashed, 0.0135); Naive-Compressed (grey) drifts without directional trend; LogitTree stays pinned at 0.013–0.015.

SDCC's `logdiff` decreases over the logged window. SDCC shares Naive-Compressed's inexpensive student forward, so both begin with an elevated diagnostic — but only SDCC includes a mechanism that should *shrink* it. Figure 11 tests this on the same eviction-heavy AgentFold cell: SDCC's mean `logdiff` falls by 55% from the start to the end of the logged window and does so monotonically across successive quarters, whereas Naive-Compressed's shows no comparable directional trend. SDCC starts 60% *above* Naive-Compressed (its leaf-gated KL initially perturbs exactly the mismatched positions) and ends *below* it, still descending toward the no-compression floor of 0.0135 at the end of the logged window. This is one editor at one scale; it is the qualitative counterpart of the ordering predicted in the main text — LogitTree/LogitMask own the floor by construction, SDCC approaches it by optimization, and Naive-Compressed has no route there at all — and not a trend estimate across seeds.

Table 10: **Live eviction-density profile per editor.** Trigger and density statistics; drift from the Naive-Compressed cell; KL from the SDCC cell (mean $\times 10^{-3}$). Both diagnostics scale monotonically with eviction density.

Editor	Trigger	eviction density sess. w/ evict	eviction density ratio	Naive-Comp mean drift	SDCC KL $\times 10^{-3}$
MemexRL	model tool (`memory_offload`)	8/256	0.012	0.024	0.01
TC-RAG	model tool (`pop`)	17/320	0.024	0.039	0.11
AgentFold	length fold (3k tokens)	132/960	0.104	0.059	2.86

Table 11: **Per-method mechanism activation under live compression.** Run-mean signature channels per white-box editor. `seg` = LogitTree segments per rollout; `pack` = LogitMask packed sub-samples per micro-batch; KL = SDCC leaf-gated KL ($\times 10^{-3}$); `act` = fraction of micro-batches with a non-empty diverging-leaf set. − = channel not applicable to that method.

Method	Channel	MemexRL	TC-RAG	AgentFold
Naive-Full	—	—	—	—
Naive-Compressed	—	—	—	—
LogitTree (ours, K-forward)	`seg`	4.10	3.90	3.23
Logit mask (ours, packed)	`pack`	1.44	1.28	1.16
SDCC (ours)	KL / `act`	0.01 / 0.99	0.11 / 0.87	2.86 / 0.82

Compression frequency and depth. The depth column of Table 1 and its companion factor $\text{freq} = \Pr[c_i > 0]$ are both measured per episode over each cell's full training run; their product is the single compression rate a one-number summary would report. We keep freq out of the main table

because it is not one quantity across columns: AgentFold folds on a length threshold, so its freq counts episodes that outgrow 3k rendered tokens, whereas TC-RAG and MemexRL evict only when the policy emits `memory_offload` or `pop`, so there freq counts a policy decision and not a length — comparable down a column but not across. depth has one meaning everywhere. Across the fifteen trained cells freq varies by more than an order of magnitude while depth varies by under $2\times$, so the product is close to a monotone rescaling of frequency and inherits its harness-dependence, whereas the two factors reported apart show what it hid: whichever editor fires removes about three fifths of the rendered context. Since depth conditions on the episodes that evict, it is N/A, not 0, for the no-compression control.

Eviction-density profile across editors. The three white-box editors span two orders of magnitude in live eviction density (Table 10): MemexRL and TC-RAG are *model-triggered* (the policy must decide to call `memory_offload` / `pop`), so from a cold start the untrained 4B model rarely invokes them, whereas AgentFold is *length-triggered* and fires on every long rollout, folding physical contexts of up to 121k tokens into ≤ 5.2k-token logical views. The low-density cells serve as in-run negative controls: as eviction density falls, Naive-Compressed's drift and SDCC's KL both collapse toward the no-compression baseline — confirming that the pitfall is compression-driven, not an artifact of harness dispatch.

H Implementation note: branch-replicated packing vs. the physical-union view

The main text (§3, Eq. 7) presents the Logit mask as a per-row visibility schedule over the physical union $\mathbf{H}_T$, in which each token appears once and different queries see different subsets of the same hidden states. Our implementation instead follows LogitTree's branch decomposition: it materializes one *copy* of each shared-trunk token per branch, so that every copy carries a hidden state computed from its branch-local causal prefix alone. The copies are then packed into a single sequence with a *block-diagonal* attention mask that prevents cross-branch interaction; position ids restart per branch.

Under dense softmax attention and the five conditions of Proposition 3, the two constructions produce identical per-target logits and gradients: the physical-union view is the row-wise characterization of what the block-packed execution computes. We retain the physical-union formulation in the main text because it gives a compact, closed-form mask definition (Eq. 7) and a direct proof path (§D.2); the branch-replicated form is what the training loop executes.

The token budget in Table 6 reports L (the physical union length) as a lower bound; the actual packed length is $N_{\mathrm{packed}} = \sum_{\pi} \ell_{\pi} \geq L$ due to trunk replication, and Table 1's "max tok." column reflects the true packed input size.