Neural Garbage Collection: Learning to Forget while Learning to Reason

Michael Y. Li **Jubayer ibn Hamid**

Emily B. Fox **Noah D. Goodman**

Abstract

Chain-of-thought reasoning has driven striking advances in language model capability, yet every reasoning step grows the KV cache, creating a bottleneck to scaling this paradigm further. Current approaches manage these constraints on the model's behalf using hand-designed criteria. A more scalable approach would let end-to-end learning subsume this design choice entirely, following a broader pattern in deep learning. After all, if a model can learn to reason, why can't it learn to forget? We introduce **Neural Garbage Collection (NGC)**, in which a language model learns to forget while learning to reason, trained *end-to-end* from outcome-based task reward alone. As the model reasons, it periodically pauses, decides which KV cache entries to evict, and continues to reason conditioned on the remaining cache. By treating tokens in a chain-of-thought and cache eviction decisions as discrete actions sampled from the language model, we can use reinforcement learning to jointly optimize how the model reasons and how it manages its own memory: what the model evicts shapes what it remembers, what it remembers shapes its reasoning, and the correctness of that reasoning determines its reward. Crucially, the model learns this behavior entirely from a *single learning signal* — the outcome-based task reward — without supervised fine-tuning or proxy objectives. On Countdown, AMC, and AIME tasks, NGC maintains strong accuracy relative to the full-cache upper bound at a 2–3x compression in peak KV cache size and substantially outperforms eviction baselines. Our results are a first step towards a broader vision where end-to-end optimization drives both capability and efficiency in language models.

1 Introduction

A central lesson of deep learning is that desirable properties emerge from end-to-end optimization pressure rather than being built in manually [35]. Yet, when it comes to making models more efficient, practitioners rely on deep domain knowledge to design architectures [13, 1], systems-level optimizations [5, 19], and inference-time heuristics [21]. We ask whether efficiency itself can be treated as a model capability. Under this perspective, the same end-to-end optimization that drives capability can also drive efficiency, enabling an additional dimension of self-improvement [33] in which models become more capable but also more efficient.

This approach is particularly promising in the chain-of-thought reasoning paradigm [43] that has driven dramatic advances in model capability [28, 15]: by thinking longer before answering, models can solve problems that would otherwise be out of reach. However, these thinking traces rapidly grow the KV cache, creating a bottleneck to further scaling along this dimension. But this cost is not fundamental to reasoning itself — after all, humans reason over long horizons despite severely limited working memory: instead, it is an artifact of how we manage the KV cache. Much of the KV cache consists of transient information: intermediate steps that become irrelevant as reasoning progresses or scratch work on sub-problems where only the answer matters downstream. This suggests that

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

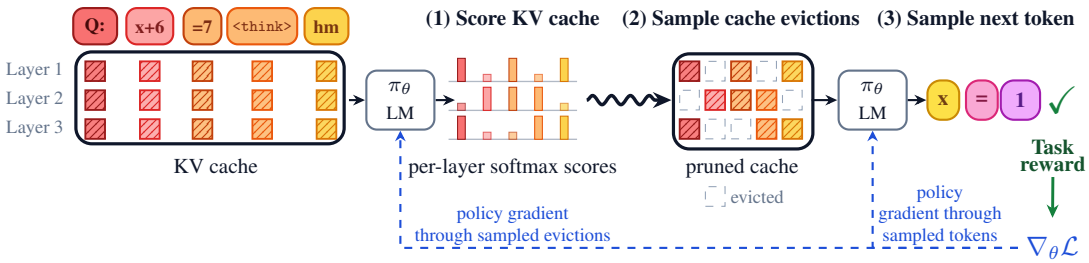

Figure 1: **Neural Garbage Collection: Learning to Forget while Learning to Reason.** As the language model (LM) generates its chain-of-thought, it periodically enters an eviction round: (1) the LM scores KV cache entries via softmax, (2) samples cache evictions using those scores via Gumbel top-k, and (3) samples the next token conditioned only on the pruned cache. Since both tokens and cache evictions are discrete actions sampled from the LM, we can *jointly* train the LM to reason and manage its memory via *end-to-end reinforcement learning* from outcome-based task reward alone (e.g., binary correctness) — no proxy objectives, no supervised finetuning.

actively managing the KV cache during reasoning need not degrade performance, and may enable models to reason over longer horizons than current resource constraints permit.

Cache management, so far, relies on fixed heuristics or proxy objectives. Heuristic eviction policies remove cache entries based on criteria such as attention weights or recency [39, 21]. Compression methods learn compact representations of the model's KV cache using hand-tailored training objectives like reconstruction or distillation losses [45, 25], and summarization-based approaches use elaborate prompts that specify what the model should preserve [36]. These proxy objectives are only indirectly tied to what the model needs to solve a task, and this mismatch is avoidable now that reasoning models are trained end-to-end against outcome-based task reward.

To that end, we introduce **Neural Garbage Collection (NGC)**, in which a language model (LM) learns to forget while learning to reason. As the model generates a chain-of-thought, it periodically pauses, computes a softmax over its KV cache entries, samples which to evict, and continues reasoning conditioned on the pruned cache. Since this eviction decision is discrete, prior work uses differentiable relaxations [20, 27] or separate supervised training for those decisions using auxiliary objectives [6, 3]. Our key insight is that, because we already train reasoning models against task reward with reinforcement learning (RL) — where tokens are discrete actions sampled from the LM — we can cast cache eviction as *another discrete action* sampled from the LM and optimize it within the same framework. A single outcome-based task reward *jointly* trains the model to reason and manage its own memory end-to-end: what the model evicts shapes what it remembers, what it remembers shapes its reasoning, and the correctness of that reasoning trains both the eviction decisions and reasoning tokens that produced it. Our key methodological contribution is to show that both types of decisions can be optimized from a *single learning signal* — the outcome-based task reward — without auxiliary objectives or separate training stages. This follows the *tabula rasa* spirit of AlphaZero [34]: end-to-end optimization pressure alone guides how the model reasons and what it forgets.

We evaluate NGC on Countdown [11, 12], and on math competition benchmarks AMC, and AIME. On Countdown, NGC doubles the accuracy of the next-best baseline (49.6% vs 21.2%) at a 2.4x reduction in peak cache size. We then validate NGC at larger scale by training on DAPO-17k [40]. On mathematical reasoning tasks (AMC, AIME), we show that NGC maintains strong performance at 2-3x reductions in peak KV cache size and outperforms all baseline cache management methods.

2 Problem setting: resource use as a learned capability

Today, practitioners manage resource constraints through design choices that span architecture, systems-level optimizations, and even scaling laws that account for inference time cost [31]. We ask whether efficiency can instead be *learned* by the model through end-to-end training.

We model resource-constrained generation as a sequential decision process of horizon T, in which the model makes two kinds of decisions at each step: a task action a_t (e.g. a generated token) and

a resource-management action u_t (e.g., which KV cache entries to evict). Let $z_t \in \mathcal{Z}$ denote the model's internal computational state at step t, and let $(a_t, u_t) \sim \pi_{\theta,\phi}(\cdot \mid z_t)$ where θ governs task behavior and ϕ governs resource allocation; these parameters can be shared or tied but this is a design choice independent of the general formulation. A trajectory is $\tau = (z_t, a_t, u_t)_{t=1}^{T}$. We optimize expected task reward subject to a resource budget B:

$$\max_{\theta,\phi} \mathbb{E}_{\tau\sim\pi_{\theta,\phi}}[R(\tau)] \quad \text{subject to} \quad \mathbb{E}_{\tau\sim\pi_{\theta,\phi}}[C(z_{1:T})] \leq B, \tag{1}$$

where R is a task reward (e.g., answer correctness) and $C : \mathcal{Z}^T \to \mathbb{R}_{\geq 0}$ measures resource use (e.g., mean fraction of KV cache entries retained over the trajectory); u_t is often discrete, which makes it amenable to RL. We instantiate this framework for KV-cache management next.

3 Neural Garbage Collection

In this section, we describe NGC, our method for training a LM to jointly reason and manage its KV cache via end-to-end RL using purely an outcome-based task reward. The main deviation from standard RLVR is that our action space consists of both eviction decisions and tokens. Our key methodological contribution is to show that both types of decisions can be optimized from a single learning signal — the outcome-based task reward — without auxiliary objectives or separate training stages. Eviction decisions, like reasoning tokens, are rewarded only insofar as they improve task success. We provide an overview of our method in Fig. 1. Due to space limitations, we defer Related Work to Appendix B.

3.1 State space and environment dynamics: grow-then-evict cycle

In this section, we define the *state space* and *environment dynamics* of our RL formulation. In language modeling, the state corresponds to the model's context window—represented by its KV cache—while the environment dynamics dictate how this context evolves over time. In standard RLVR, the LM samples the next token conditioned on all previously generated tokens and the KV cache steadily grows. In NGC, the environment dynamics differ because the model actively modifies its own state space by managing its KV cache, in addition to generating tokens.

Specifically, we introduce a "grow-then-evict" dynamic. Periodically, at a set of discrete eviction steps, the LM selects cache entries to keep. Every δ tokens, an *eviction round* fires: the model produces scores for all cache entries and samples a $(1-\varepsilon)$ fraction of them to keep, where $\varepsilon \in (0, 1]$ is the *eviction rate*, and permanently evicts the rest; we will refer to δ as the *eviction cadence*. Both prefill and generation tokens are eligible for eviction and we evict the same fraction (though not the same entries) in each of the L layers of the transformer. The first eviction round fires when the total number of tokens in the cache including prefill tokens reaches δ. Between eviction rounds, the cache grows as usual as the model generates tokens conditioned on the surviving entries. Under these dynamics, the maximum cache size before an eviction round converges to the steady state value $L\frac{\delta}{\varepsilon}$; this is independent of the number of tokens generated. We formalize this in Section A.2 .

3.2 Extending the action space: block-level evictions via existing attention mechanism

Having formalized the state space, we now describe how we extend the LM's *action space* to include cache evictions. At each eviction round, the action is a subset of cache entries to keep. Language models do not natively support this type of action since their action space consists only of tokens. We describe how we use the transformer's existing attention mechanism to parameterize cache evictions, enabling the model to act on its own KV cache — even though it was never trained, through pre-training or supervised fine-tuning (SFT), to do so.

Parameterizing eviction actions To perform KV cache evictions, the LM must be able to evaluate the utility of its cache entries—a task it was never explicitly trained to perform. Fortunately, the transformer's native attention mechanism is already well-suited for this role. First, after standard pre-training, attention weights naturally encode a prior over which past entries are relevant for ongoing computation. This provides an effective initialization, which has been shown to be essential for stable RL [12]. Second, re-purposing this existing machinery introduces no additional architectural

overhead. We therefore parameterize eviction decisions with the model's attention scores, setting $\phi = \theta$ and introducing no new parameters.

Concretely, at each eviction step, we score the remaining (prefix) keys according to how much they are attended to by the w most recent queries. For layer ℓ, we use the queries $\mathbf{Q}^{(\ell)}$ associated with the w most recent tokens, compute attention weights from these queries to the prefix keys $\mathbf{A} = \mathrm{softmax}\left(\frac{\mathbf{Q}^{(\ell)}\mathbf{K}^{(\ell)\top}_{\text{prefix}}}{\sqrt{d_h}}\right)$. We average across heads and recent queries to obtain a scalar importance score for each prefix key $\psi_t = \frac{1}{H \cdot w}\sum_{h=1}^{H}\sum_{q=1}^{w} A_{q,t}^{(\ell,h)}$.

In our experiments, we use $w = 5$, so scoring incurs only a small memory overhead. This construction is inspired by sparse attention methods [6], which also use attention scores to decide which keys to read when decoding a token. Rather than deciding which entries are read at the current step, NGC uses these scores to decide which entries are *permanently removed*. Moreover, instead of requiring an auxiliary KV cache, used to decide which actual KV cache entries get used for a decode step, we re-purpose the attention mechanism into a parameter-free eviction policy.

Coarsening the action space via block-level evictions Evicting at the individual key level creates a *credit assignment* problem: the marginal contribution of any single key is difficult to isolate. Moreover, the importance of adjacent keys is highly correlated: semantically coherent units such as sub-computations typically occupy contiguous spans of tokens in the chain-of-thought. Consider a set of surviving keys of length T. We *coarsen* the action space by working at the block level and partitioning the T keys into $N = \lceil T/b \rceil$ blocks of contiguous tokens of size b, where the final block may be smaller if $b \nmid T$ (b does not divide T). Let m_t be a mask indicating whether a key t is valid (e.g., non-padding); we omit the layer index for simplicity. We aggregate the per-key scores $\boldsymbol{\psi} \in \mathbb{R}^T$ to block-level scores by averaging over valid keys in a block: $s_j = \frac{\sum_{t \in \mathcal{B}_j} m_t\, \psi_t}{\sum_{t \in \mathcal{B}_j} m_t}$ yielding $\mathbf{s} \in \mathbb{R}^N$, one score per block. This dramatically reduces the size of the action space. While the focus of NGC is on algorithmic choices rather than hardware optimizations, blockwise selection broadly aligns with hardware principles: contiguous memory accesses achieve higher GPU throughput [41], and block-level granularity maps naturally onto paged KV cache systems such as vLLM [19], which partition the cache into pages of contiguous keys.

3.3 Efficiently sampling eviction actions with Gumbel-top-k

We now define a procedure for tractably *sampling eviction actions* for training. KV cache eviction is conventionally performed using deterministic top-k rules [21, 39]. We instead formulate cache eviction as a *stochastic action* sampled from the LM. This formulation enables us to leverage policy gradient methods for unbiased gradient estimates: stochastic sampling with exact log-probabilities is precisely the structure these methods require.

Concretely, we need to be able to sample a size-K subset of kept cache entries in a way that exposes a well-defined log-probability of the selected set. This might seem challenging since subset selection is combinatorial and the partition function is intractable. We formulate this task as sequential sampling without replacement using the *Gumbel-top-k trick* [18, 37]. Sampling can be done in parallel: we sample a size-K subset by adding i.i.d. Gumbel noise to the block logits and keep the K largest perturbed values. The log-probability of the selected blocks $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_K)$ is

$$\log p(\boldsymbol{\sigma} \mid \mathbf{s}) = \sum_{j=1}^{K} \left[s_{\sigma_j} - \log \sum_{t \notin \{\sigma_1, \ldots, \sigma_{j-1}\}} \exp(s_t) \right] \tag{2}$$

where indices range over blocks. Each term naively requires recomputing the partition function over the remaining blocks, i.e., computing a `logsumexp` K times. We avoid this via a prefix sum trick that tracks the cumulative fraction of removed probability mass.

3.4 Policy optimization: token and eviction policy gradients

With the states, actions, and sampling procedure established, we now formalize the *policy optimization objective*. In standard RLVR, the policy gradient applies solely to the token space. Here, we define

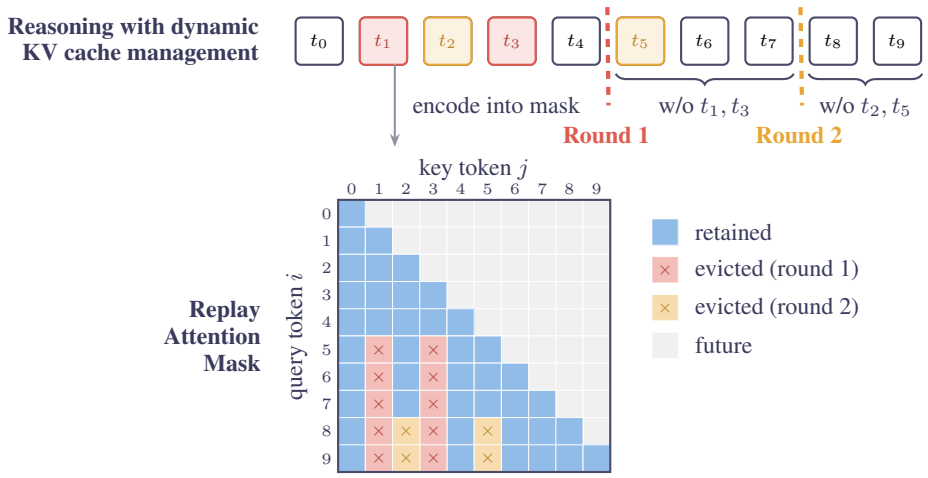

Figure 2: **Replay masks enable efficient end-to-end RL training.** The model evicts cache entries at fixed intervals (**round 1**: t_1, t_3; **round 2**: t_2, t_5), and we encode the resulting visibility patterns into a replay mask: row i records which keys were visible when t_i was generated. A single forward pass over the sampled tokens with this mask reproduces the next-token distributions in parallel.

a unified objective over a *joint* action space consisting of both tokens and discrete cache evictions. Concretely, we detail how the single, outcome-based task reward in RLVR can be used to update the model to jointly optimize both its chain-of-thought reasoning and how it manages its own KV cache.

Given a prompt x, we sample G trajectories $\{\tau_i\}_{i=1}^{G}$ from π_θ, where each trajectory $\tau_i = \{(o_{i,t}, \boldsymbol{\sigma}_{i,t})\}_{t=1}^{|\tau_i|}$ interleaves token generation $o_{i,t}$ with eviction decisions $\boldsymbol{\sigma}_{i,t}$, where $\boldsymbol{\sigma}_{i,t} = \emptyset$ at timesteps t that do not correspond to an eviction round. Each trajectory is scored with a binary task reward $r_i \in \{0, 1\}$ computed from the generated tokens $\{o_{i,t}\}$.

We compute group-normalized advantages $\hat{A}_i = r_i - \frac{1}{G}\sum_{j=1}^{G} r_j$. and optimize π_θ using Dr. GRPO [24, 15]. The only training signal is task accuracy. For the tokens, we have a token-level policy gradient:

$$\mathcal{L}_{\text{token}} = -\mathbb{E}\left[\frac{1}{G}\sum_{g=1}^{G}\sum_{t=1}^{|o_i|}\log \pi_\theta(o_{i,t} \mid \mathcal{C}_{i,t})\,\hat{A}_i\right]. \tag{3}$$

Here, $\mathcal{C}_{i,t}$ denotes the context available at step t of trajectory i (i.e., the surviving KV cache entries).

For the eviction decisions, we have a policy gradient over eviction decisions

$$\mathcal{L}_{\text{mem}} = \sum_{\ell=1}^{L}\mathcal{L}_{\text{mem}}^{\ell}, \qquad \mathcal{L}_{\text{mem}}^{\ell} = -\mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\mathcal{T}_i^{\ell}|}\sum_{t\in\mathcal{T}_i^{\ell}}\log \pi_\theta(\boldsymbol{\sigma}_{i,t}^{\ell} \mid \mathcal{H}_{i,t}^{\ell})\,\hat{A}_i\right] \tag{4}$$

where L is the number of transformer layers, $\mathcal{T}_i^{\ell}$ is the set of eviction rounds at layer ℓ for rollout i that fire before sequence termination, $\boldsymbol{\sigma}_{i,t}^{\ell}$ is the subset of blocks retained at round t, and $\mathcal{H}_{i,t}^{\ell}$ is the context used to make eviction decisions at the ℓ-th layer (layer-ℓ queries and alive keys at round t); with slight abuse of notation, we write $\pi_\theta(\boldsymbol{\sigma}_{i,t}^{\ell} \mid \mathcal{H}_{i,t}^{\ell})$ to denote the probability of the retained subset given by Gumbel top-k in Equation 2. The inner mean averages over eviction rounds within an individual rollout. We use the layer index to emphasize that we make separate eviction decisions for each KV entry in each layer rather than making the same decision for all layers for a given token.

Crucially $\hat{A}_i$ is the *same* for both $\mathcal{L}_{\text{token}}$ and $\mathcal{L}_{\text{mem}}$: both learning to generate reasoning tokens and learning to evict come from the same learning signal. The total objective is:

$$\mathcal{L} = \mathcal{L}_{\text{token}} + \mathcal{L}_{\text{mem}}. \tag{5}$$

Importantly, there is no independent training signal beyond the task reward already present in $\hat{A}_i$. The resource constraint in our setting enters by construction (evicting a fixed amount of cache each

time) rather than through auxiliary losses. Because eviction decisions are scored by the model's own queries and keys, the policy gradient from $\mathcal{L}_{\text{mem}}$ flows into every parameter of the LM. Token generation and cache management are therefore treated as a single policy optimized under a single outcome reward.

3.5 Computing rollout log-probabilities efficiently with replay masks

To compute policy gradients, we need to evaluate the log-probabilities of the rollouts. This section introduces *replay attention masks*, a mechanism to efficiently compute these probabilities.

The key challenge is that because NGC dynamically modifies its own context through cache eviction, it breaks an assumption in standard RLVR: that the model attends to all previous tokens when generating a new token. When the model generates a new token, it does not attend to evicted cache entries. Therefore, computing log-probabilities under a standard causal attention mask would be incorrect and create a form of off-policyness [23]. We use replay attention masks to correctly and efficiently compute the log-probability of the generated tokens. Specifically, we "replay" the rollout log-probabilities under a per-layer attention mask that replicates the exact visibility pattern over past tokens induced by the eviction decisions (Fig. 2). A single forward pass over those tokens with these replay masks gives us the correct log-probabilities. This introduces no additional overhead since standard RL already requires such a forward pass; we store a negligible amount of state (a binary mask per layer and per eviction round).

Another technical challenge is ensuring that gradients flow from $\mathcal{L}_{\text{mem}}$; see Fig. 8 for the autograd graph. Eviction decisions are scored using the queries/keys at each eviction round. We cannot simply cache them during rollouts and pass them into Equation 4 since they would be *detached from the autograd graph*; no gradient from $\mathcal{L}_{\text{mem}}$ would reach θ. We compute these inputs in the replay forward pass, so that the values are on the autograd graph. Using these inputs, we compute the eviction log-probabilities via Equation 4, analogously to how we compute per-token log-probabilities. The gradients then flow from $\mathcal{L}_{\text{mem}}$ through the eviction decisions into θ.

3.6 Staircase eviction rate curriculum

Finally, we introduce a curriculum to ensure that our model can be trained stably to both optimize its own reasoning and cache management. During training, we smooth the transition to more aggressive eviction rates by slowly increasing the eviction rate using a staircase curriculum over the retention rate p_0 (i.e., percentage of kept entries). More formally, let $\{\rho_0, \rho_1, \ldots, \rho_K\}$ be a sequence of retention rates with $\rho_0 \geq \rho_1 \geq \cdots \geq \rho_K$, and let Δ denote the number of training steps per stage. At training step t, the current stage index is $\ell = \min(\lfloor t/\Delta \rfloor, K)$. Within stage $\ell < K$, we apply a linear blend toward the next level during the final fraction $\alpha \in (0, 1)$ of the stage. We set $\alpha = 0.6$ in our experiments. Let $s = (t \bmod \Delta)/\Delta$ denote the fractional progress within the stage. Then:

$$p_0(t) = \begin{cases} \rho_\ell & \text{if } s < 1 - \alpha, \\ \rho_\ell + \dfrac{s - (1-\alpha)}{\alpha} \left(\rho_{\ell+1} - \rho_\ell\right) & \text{if } s \geq 1 - \alpha. \end{cases} \tag{6}$$

At the final stage $\ell = K$, we set $p_0(t) = \rho_K$. See Fig. 9 for an illustration of the schedule.

4 Experiments

We begin with a controlled study on Countdown, validating and ablating core design choices, and then evaluate on standard math competition benchmarks.

Model and training recipe. We train DeepSeek-R1-Distill-Qwen-1.5B [15] using on-policy Dr. GRPO. We use block size $b = 32$ and window size $w = 5$. Training hyperparameters are standard; see Section A.1 in the Appendix for details.

Baselines. We compare NGC against four cache eviction baselines applied to a model trained with the full KV cache: **SnapKV** [21] (attention weights), **KeyDiff** [29] (key diversity), **KNorm** [7] (key norm statistics), and **StreamingLLM** [39] (sliding window with attention sinks). For each baseline, we use the same eviction cadence, target retention budget, and maximum response length as NGC.

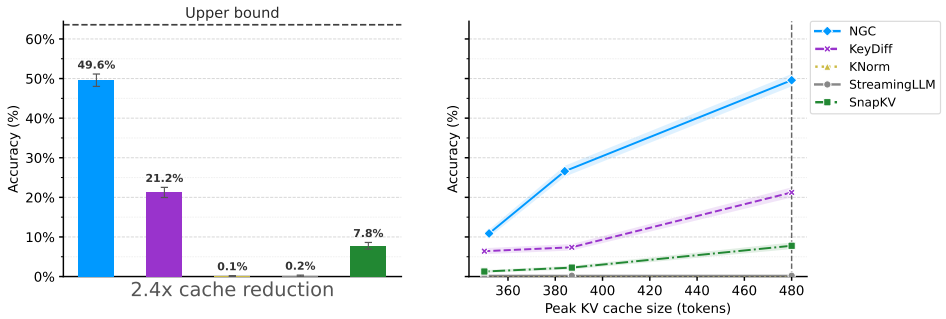

Figure 3: **Comparing NGC against KV cache eviction baselines. (left)** NGC significantly outperforms baseline cache eviction methods. Error bars correspond to 1 SE and we report pass@1 accuracies. All methods evict 50% of entries at each eviction round, corresponding to 2.4x reduction in peak KV cache size. Dashed horizontal line corresponds to a model trained and evaluated with full cache. **(right)** NGC generalizes to larger eviction rates that it was not trained for; the vertical dashed line indicates the minimum cache size during training; NGC Pareto-dominates all other methods.

Metrics. We report accuracy on held-out test problems. During evaluation, we replace Gumbel-top-k sampling with greedy top-k selection, retaining the K highest-scored blocks deterministically. To measure memory savings, we report the **average peak KV cache reduction**: the factor by which a method reduces the **peak KV cache size** relative to no eviction. Let p_i denote the prompt length, c_i^{base} the completion length under the no-eviction baseline, and c_i^{method} the completion length under the method being evaluated. Let $\text{peak}(p, c, \epsilon, \delta)$ denote the maximum number of KV entries in memory during generation for a sequence with prompt length p, completion length c, eviction rate ϵ, and cadence δ. The average peak KV cache reduction is then $\mathbb{E}\left[\frac{\text{peak}(p_i, c_i^{\text{base}}, 0, \delta)}{\text{peak}(p_i, c_i^{\text{method}}, \epsilon, \delta)}\right]$ where the expectation is over prompts. A value of 2 means half the peak memory of the no-eviction baseline is used. We use peak cache size as our key metric since it is an easy to measure resource that bottlenecks inference, while wall-clock time is often confounded by factors like batch effects and system management.

4.1 Controlled analysis on Countdown

Setup. We train DeepSeek-R1-Distill-Qwen-1.5B [15] for 250 steps on Countdown [11], a combinatorial arithmetic task requiring a model to combine randomly drawn numbers via basic operations to reach a target number. We use a maximum completion length of 1024 tokens, eviction cadence 256, 32 unique prompts per update step, and 16 rollouts per prompt. Our training set contains 327,680 problems (3 and 4 input numbers); our test set contains 1024 held-out problems. Countdown is a demanding testbed since DeepSeek-R1-Distill-Qwen-1.5B achieves near-zero accuracy before training; NGC must jointly learn to reason about this task and manage memory from scratch. It has also become a standard setting for evaluating LM reasoning at accessible scales [12]. All methods use a max response length of 1024 at test time.

4.1.1 Main results

NGC outperforms all baselines. We report accuracy on 1024 held-out test problems. In Fig. 3, we compare NGC against several baseline methods at a 50% eviction rate per eviction round (corresponding to 2.4x reduction in peak cache size). NGC achieves 49.6% accuracy, substantially outperforming all heuristic methods. This shows that NGC learns to reason and manage memory jointly, a more challenging task than learning to reason alone.

In Fig. 3 (right), we evaluate NGC across a range of eviction rates more aggressive than those seen during training. NGC pareto-dominates all baselines, and its performance degrades gracefully beyond the training distribution until a very low cache size.

4.1.2 Ablations

End-to-end training is essential. To understand the benefits of end-to-end training, we consider two ablations. We first assess whether simply exposing the model to eviction during RL enables the

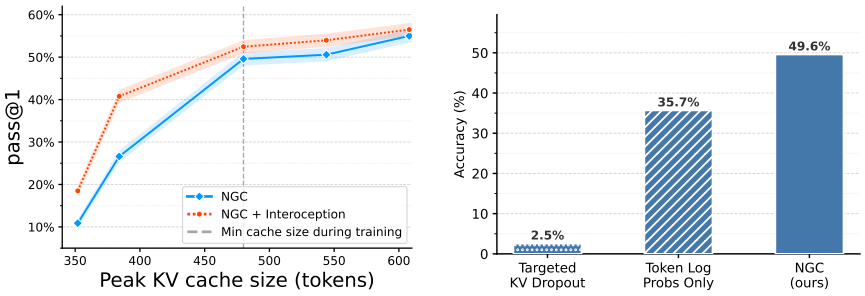

Figure 4: *left:* **Budget-aware interoception.** By training/evaluating the LM with the eviction rate in the prompt, we see more graceful performance degradation as peak cache size decreases and generalization to stricter budgets. At aggressive budgets, accuracy boosts by 8-13%. *right:* **Ablating NGC end-to-end learning.** Targeted KV dropout evicts cache entries during RL but ignores the off-policyness introduced by evictions; token log-probs only corrects for off-policyness but drops the eviction policy gradient term. Both drastically hurt performance.

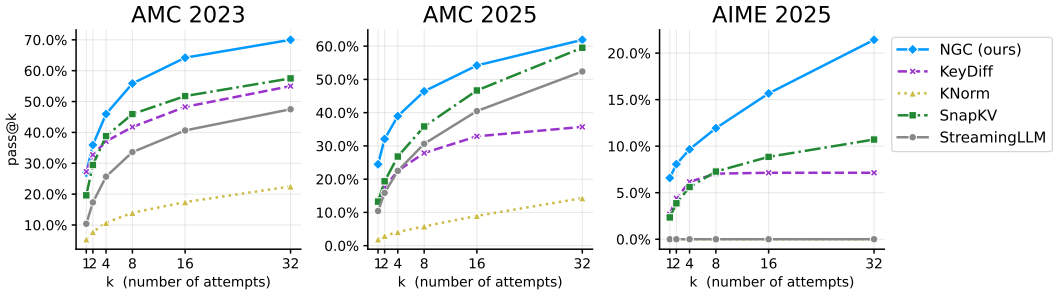

Figure 5: **Pass@k across math reasoning benchmarks.** NGC consistently outperforms all inference-time eviction baselines; all methods evict 50% each eviction round (3-5x peak cache size reduction).

model to handle evictions at test time. **Targeted KV dropout** evicts cache entries during training using NGC's scoring mechanism and applies the same curriculum, but only optimizes Eq. (3) with a standard causal mask—ignoring the off-policyness that eviction introduces (Section 3.5). **Token log-probs only** corrects for off-policyness using replay masks but omits the eviction policy gradient term $\mathcal{L}_{\text{mem}}$; this evaluates whether explicitly training eviction actions end-to-end is important. Fig. 4 (*right*) shows that both significantly hurt performance. We study NGC's training dynamics in Fig. 9.

4.2 Budget-aware Interoception

Intuitively, the LM could benefit from "sensing" its budget constraints before reasoning. We ask whether conditioning the model on the eviction rate improves its generalization to various eviction rates. We call this *budget-aware interoception*. We simply modify the prompt during training and testing: we sample an eviction rate ρ for a group, and append this tag to every prompt in that group: `<eviction_rate>`ρ`%</eviction_rate>`. At inference time, we vary the eviction rate and compare against a standard NGC model trained under identical conditions but without the eviction rate tag.

Results. Fig. 4 (*left*) reports accuracy as a function of the peak cache size. The interoceptive model matches or exceeds standard NGC at eviction rates seen during training, and the gap widens at aggressive eviction rates beyond the training distribution.

4.3 Mathematical reasoning experiments

To test NGC on broader reasoning tasks, we now evaluate on standard math competition benchmarks.

Setup. We train DeepSeek-R1-Distill-Qwen-1.5B on DAPO-17k [40], a large-scale math reasoning dataset spanning a range of problem types and difficulties. We evaluate on AMC 2023, AMC 2025,

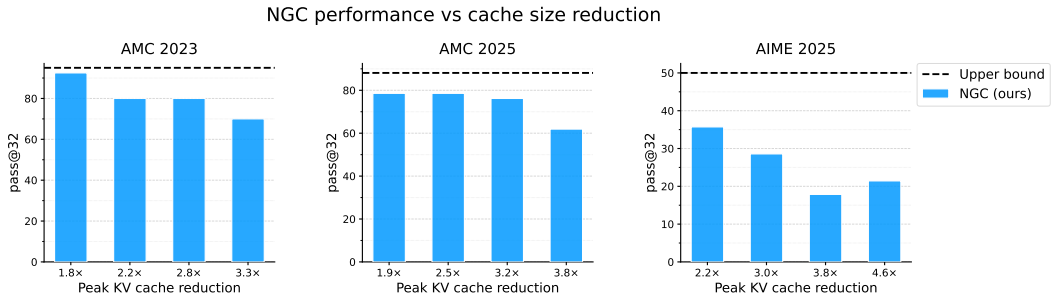

Figure 6: **NGC pass@32 across cache size reductions.** For 2-3x reductions in max cache size, NGC can maintain relatively strong accuracy relative to the no eviction upper bound.

and AIME 2025 to assess whether NGC generalizes beyond this training distribution. We use the same baseline eviction approaches as before on top of a model trained without eviction. We train with an eviction cadence of 350 and a max response length of 1050, using 256 prompts per update step and 8 rollouts per prompt for a total of 469 steps. At test time, we use top_p $= 0.95$, temperature 0.6, and a max response length of 3850 tokens for NGC and baseline methods.

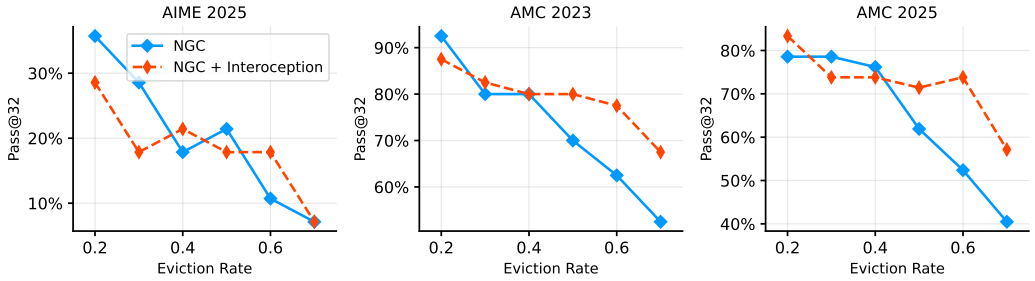

Figure 7: **Budget-aware interoception for math reasoning tasks.** Training and evaluating with budget interoception boosts AMC 2023/2025 performance but does not boost AIME 2025 performance.

Results. Fig. 5 reports pass@k [2] at a 3-5x cache reduction size (corresponding to 50% eviction rate at every eviction round) on AMC 2023, AMC 2025, and AIME 2025. NGC consistently outperforms baselines across all benchmarks. Heuristic baselines are inconsistent across datasets: SnapKV fails on Countdown but performs reasonably on math reasoning tasks. This inconsistency illustrates why a proxy for importance is fragile. NGC's strong performance across tasks shows that end-to-end training allows the model to adapt to tasks. In Fig. 6, we compare performance across cache sizes. At more moderate (2-3x) cache reduction sizes, NGC can come close to matching a model trained with no eviction and evaluated with no eviction (upper bound). We also again experiment with a budget-aware interoception approach. In Fig. 7, we find that this boosts performance at aggressive eviction rates for AMC 2023/2025 but does not provide a strong benefit for AIME 2025.

5 Conclusion & Limitations

We introduced Neural Garbage Collection, a framework in which a language model jointly learns to reason and manage its own KV cache from outcome-based task reward alone, with no auxiliary objectives, distillation targets, or supervised stages. NGC outperforms cache-eviction baselines across Countdown and competition mathematics, maintaining strong accuracy at 2–3x peak cache reductions. More broadly, NGC shows that that efficiency itself can be treated as a capability acquired under end-to-end optimization pressure, rather than a property engineered into the system by hand. The same machinery that already trains models to reason extends naturally to decisions about a model's own resource use.

Our study is scoped to a clean setting where we have verifiable tasks with unambiguous outcome rewards. The 1.5B-parameter model reflects our compute constraints but also affords the kind of

careful controlled study that is harder to conduct at frontier scale. Extending NGC to softer reward signals and to larger models are natural next steps.

Acknowledgments and Disclosure of Funding

This work was supported in part by ONR Grant N00014-22-1-2110, NSF Grant 2205084, and the Stanford Institute for Human-Centered Artificial Intelligence (HAI). EBF is a Biohub, San Francisco, Investigator.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[2] Mark Chen et al. Evaluating large language models trained on code, 2021.

[3] Yu Chen, Runkai Chen, Sheng Yi, Xinda Zhao, Xiaohong Li, Jianjin Zhang, Jun Sun, Chuanrui Hu, Yunyun Han, Lidong Bing, Yafeng Deng, and Tianqiao Chen. Msa: Memory sparse attention for efficient end-to-end memory model scaling to 100m tokens, 2026.

[4] Tri Dao and Albert Gu. Transformers are ssms: generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

[5] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[6] DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025.

[7] Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective l_2 norm-based strategy for kv cache compression, 2024.

[8] Aniket Didolkar, Nicolas Ballas, Sanjeev Arora, and Anirudh Goyal. Metacognitive reuse: Turning recurring llm reasoning into concise behaviors, 2025.

[9] Sabri Eyuboglu, Ryan Saul Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Ruoyu Liu, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study. In *The Fourteenth International Conference on Learning Representations*, 2026.

[10] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.*, 23(1), January 2022. ISSN 1532-4435.

[11] Kanishk Gandhi, Denise H J Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah Goodman. Stream of search (sos): Learning to search in language. In *First Conference on Language Modeling*, 2024.

[12] Kanishk Gandhi, Ayush K Chakravarthy, Anikait Singh, Nathan Lile, and Noah Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective STars. In *Second Conference on Language Modeling*, 2025.

[13] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First Conference on Language Modeling*, 2024.

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

[15] Daya Guo et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, page 633–638, 2025.

[16] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the Third Conference on Uncertainty in Artificial Intelligence*, 1987.

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

[18] Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The Gumbel-top-k trick for sampling sequences without replacement. In *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2019.

[19] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

[20] Adrian Łańcucki, Konrad Staniszewski, Piotr Nawrot, and Edoardo Ponti. Inference-time hyper-scaling with KV cache compression. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026. URL https://openreview.net/forum?id=8ZiElzQxf1.

[21] 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 *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[22] Falk Lieder and Thomas L. Griffiths. Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources. *Behavioral and Brain Sciences*, 43:e1, 2020. doi: 10.1017/S0140525X1900061X.

[23] Jiacai Liu, Yingru Li, Yuqian Fu, Jiawei Wang, Qian Liu, and Yu Shen. When speed kills stability: Demystifying RL collapse from the training-inference mismatch, 2025. URL https://richardli.xyz/rl-collapse.

[24] Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective, 2025.

[25] Giovanni Monea, Yair Feldman, Shankar Padmanabhan, Kianté Brantley, and Yoav Artzi. Breadcrumbs reasoning: Memory-efficient reasoning with compression beacons, 2025.

[26] Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[27] Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo M. Ponti. Dynamic memory compression: retrofitting llms for accelerated inference. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[28] OpenAI. Openai o1 system card, 2024. URL https://arxiv.org/abs/2412.16720.

[29] Junyoung Park, Dalton Jones, Matthew J Morse, Raghavv Goel, Mingu Lee, and Christopher Lott. Keydiff: Key similarity-based KV cache eviction for long-context LLM inference in resource-constrained environments. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2026.

[30] Stuart Russell and Eric Wefald. *Do the right thing: studies in limited rationality*. MIT Press, Cambridge, MA, USA, 1991. ISBN 0262181444.

[31] Nikhil Sardana, Jacob Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws, 2025.

[32] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[33] David Silver and Richard S. Sutton. Welcome to the era of experience. https://storage.googleapis.com/deepmind-media/Era-of-Experience%20/The%20Era%20of%20Experience%20Paper.pdf, 2025. Preprint.

[34] David Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404.

[35] Richard S. Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, March 2019. Blog post.

[36] Vivek Vajipey, Aditya Tadimeti, Justin Shen, Ben Prystawski, Michael Y. Li, and Noah Goodman. Simple, scalable reasoning via iterated summarization. In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025. URL https://openreview.net/forum?id=uhZLKclfGB.

[37] Tim Vieira. Gumbel-max trick and weighted reservoir sampling, 2014. Blog post.

[38] Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

[39] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024.

[40] Qiying Yu et al. Dapo: An open-source llm reinforcement learning system at scale, 2025.

[41] Jingyang Yuan et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, July 2025.

[42] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

[43] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[44] Eric Zelikman, Georges Raif Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah Goodman. Quiet-STar: Language models can teach themselves to think before speaking. In *First Conference on Language Modeling*, 2024.

[45] Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast KV compaction via Attention Matching, 2026. URL https://arxiv.org/abs/2602.16284.

A Technical appendices and supplementary material

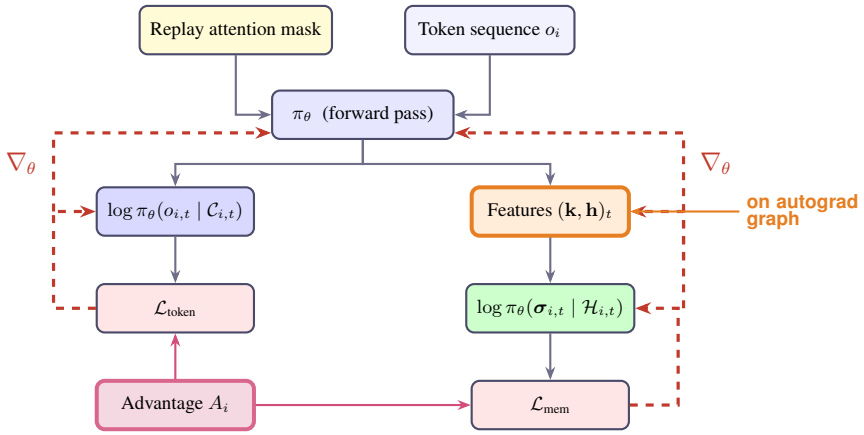

Figure 8: **Replay masks route gradients through eviction decisions.** We perform a forward pass over the tokens with π_θ using replay attention masks (see Figure 2). Both losses share the same group-normalized advantage A_i (purple) but differ in their log-probabilities: $\mathcal{L}_{\text{token}}$ uses the per-token next-token log-probs, while $\mathcal{L}_{\text{mem}}$ uses the Gumbel-top-k log-prob of the eviction decisions. Crucially, since the inputs that are used to make eviction decisions $(\mathbf{k}, \mathbf{h})_t$ are collected during this forward pass, they are on the autograd graph. Therefore, gradients from $\mathcal{L}_{\text{mem}}$ flow back into θ and, as a consequence, eviction decisions are treated as a *native* action of the model that is optimized end-to-end just like tokens in the chain-of-thought.

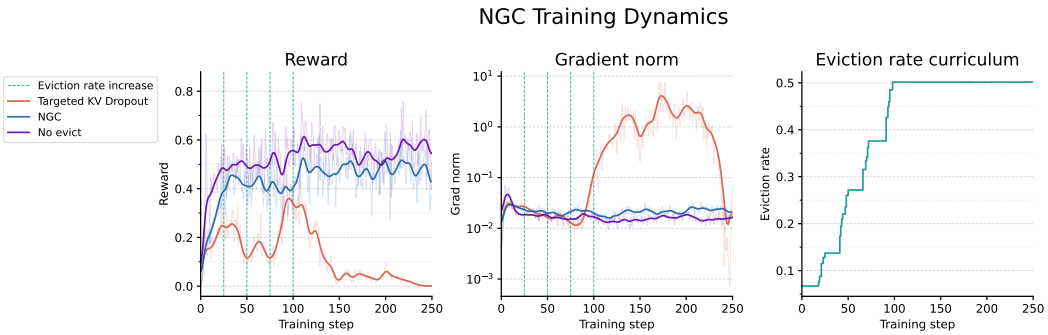

Figure 9: **NGC training dynamics on Countdown.** (**left**) Reward over training: NGC improves steadily but more slowly than the no eviction model, but eventually reaches a similar level of accuracy. Targeted KV Dropout collapses after around step 100 due to off-policyness. (**center**) Targeted Dropout exhibits exploding gradient norms coinciding with its reward collapse, whereas NGC and the no-eviction baseline remain stable throughout training. (**right**) The staircase eviction rate curriculum, shared across both NGC and targeted KV dropout, which gradually increases the percentage of KV cache evicted to 50% over $\sim$100 steps. Vertical dashed teal lines in the left and center panels mark the steps where the eviction rate begins to increase.

A.1 Hyperparameters

Min response length penalty. Completions whose total length $T_i = |\text{prompt}_i| + |\text{completion}_i|$ falls before the first eviction round do not participate in an eviction round and carry no signal. For the `DAPO-17k` experiments, we found that it can be helpful to set $r_i = 0$ if the response length $T_i < L$.

Table 1: Countdown training hyperparameters.

Parameter	**Value**
Maximum response length	1024 tokens
Sampling temperature	0.9
Top-k	50
Optimizer	AdamW
Adam parameters (β_1, β_2)	$(0.9, 0.95)$
Adam ϵ	1×10^{-15}
Gradient norm clipping	1.0
Learning rate scheduler	Constant
Learning rate	5×10^{-6}
KL penalty coefficient	0.0

Table 2: DAPO-17k training hyperparameters.

Parameter	Value
Maximum response length	1050 tokens
Sampling temperature	1.0
Top-k	disabled
Optimizer	AdamW
Adam parameters (β_1, β_2)	$(0.9, 0.999)$
Adam ϵ	1×10^{-8}
Gradient norm clipping	1.0
Learning rate scheduler	Constant
Learning rate	5×10^{-6}
KL penalty coefficient	0.0

A.2 Cache size converges to a constant under grow-then-evict dynamics

Proposition 1 (Steady-state maximum cache size under periodic eviction)**.** *Consider a cache that undergoes periodic eviction as follows: every δ newly generated tokens, an eviction round keeps a $(1-\varepsilon)$ fraction of the current cache entries in each layer and permanently removes the remaining ε fraction, where $\varepsilon \in (0, 1]$. Assume the prefill length is less than δ.*

Let c_t denote the cache size in a single layer *immediately before eviction round t. Then the sequence $\{c_t\}$ converges to the unique fixed point*

$$c^\star = \frac{\delta}{\varepsilon}.$$

Therefore, since we keep the same fraction in each layer, the total KV cache size across all L layers immediately before an eviction round converges to

$$C^\star = Lc^\star = L\frac{\delta}{\varepsilon}.$$

Proof. By construction, immediately after eviction round t, the cache size in a single layer is $(1-\varepsilon)c_t$. Before the next eviction round, the model generates δ new tokens, so we have the following recursion

$$c_{t+1} = (1-\varepsilon)c_t + \delta.$$

Since $\varepsilon \in (0, 1]$, we have $|1-\varepsilon| < 1$, so this recursion is a contraction and therefore converges to a unique fixed point. The fixed point condition gives $c_{t+1} = c_t = c^\star$

$$c^\star = (1-\varepsilon)c^\star + \delta.$$

Rearranging,

$$\varepsilon c^\star = \delta, \qquad \Rightarrow \qquad c^\star = \frac{\delta}{\varepsilon}.$$

Since each of the L layers has the same cache size under these dynamics, the total KV cache size immediately before eviction is

$$C^\star = L\frac{\delta}{\varepsilon}.$$

as desired. □

A.3 Implementation Details

We maintain a custom fork of the Huggingface implementation of Qwen2 as well as a custom fork of the `GRPOTrainer` from TRL [38]. We describe some of the key modifications below.

Per-Layer Attention Masks. Because different layers keep different subsets of KV entries after eviction, a single attention mask shared across layers is insufficient, for applying the replay mask mechanism described in Section 3.5. The standard Huggingface transformers library does not support this. We extend the transformer decoder layers to accommodate layer-level attention masks. We store a list of per-layer attention masks inside a modified `DynamicCache` object (`attention_mask_list`); at each forward pass, we detect whether we have a layer level mask and, if so, use it in the appropriate decoder layer.

Reconstructing Replay Attention Masks from Retention Decisions. To re-run the model under fixed eviction decisions during the replay pass, we must reconstruct the exact (per-layer level) attention masks that would have been in effect at each token position during the original rollout. Given per-layer retention decisions $\{d_r^{(\ell)} \in \{0,1\}^{B\times|\mathcal{A}_r|}\}_{r=0}^{R-1}$, where B is the batch size, $R = \lceil T/L \rceil - 1$ is the number of eviction rounds, T is the total sequence length, L is the fixed eviction period (denoted δ in the main text), and $\mathcal{A}_r$ denotes the alive set at round r, we construct a dense Boolean mask $M^{(\ell)} \in \{0,1\}^{B\times T\times T}$ independently for each layer ℓ as follows.

The sequence is partitioned into $\lceil T/L \rceil$ contiguous blocks, where block r spans global token indices $[rL,\ \min((r+1)L,\ T))$. Within each block, queries attend causally to earlier tokens in the same block via the standard lower-triangular mask. The alive set is initialized as $\mathcal{A}_0 = \text{block}_0$ and updated after each eviction round: applying $d_r^{(\ell)}$ to $\mathcal{A}_r$ yields the retained set

$$\texttt{kept}_r^{(\ell)} = \big\{ i \in \mathcal{A}_r : d_{r,i}^{(\ell)} = 1 \big\}, \tag{7}$$

after which the alive set for the next round is $\mathcal{A}_{r+1} = \texttt{kept}_r^{(\ell)} \cup \text{block}_{r+1}$. All queries in block $r+1$ attend to every token in $\texttt{kept}_r^{(\ell)}$, in addition to causally attending within block $r+1$ itself. After iterating over all R block boundaries, the diagonal of $M^{(\ell)}$ is set to 1 to ensure every token attends to itself. The resulting masks $\{M^{(\ell)}\}_\ell$ are injected into our modified `DynamicCache`, so that the replay forward pass attends to exactly the same context at each layer as existed during the original rollout.

Note that we need 2D attention masks rather than a broadcastable 1D key-side mask. Standard causal masking is compactly representable because visibility is monotonic in query position — once a key becomes visible, it stays visible. Eviction breaks this monotonicity: a key evicted at round r is visible to queries before round r but invisible to queries after, so visibility is neither a function of key position alone (unlike a padding token) nor monotone in query position.

A.4 Compute

The Countdown experiments were run on a node of H100s/A100s. The DAPO experiments were run on a node of B200s.

B Related work

Language Model Self-Improvement. A growing body of work explores language models that self-improve. STaR [43] demonstrated that LMs can bootstrap reasoning ability by fine-tuning on self-generated chains-of-thought filtered by correctness. Subsequent work extended this paradigm to more general latent reasoning and iterative self-improvement, including Quiet-STaR [44], ReST [14], and Self-Rewarding Language Models [42]. The reinforcement learning from verifiable rewards (RLVR) paradigm [15, 28] showed that self-improving reasoning scales: models trained on outcome supervision can acquire sophisticated reasoning capabilities. NGC is inspired by this line of work, but targets a different dimension of self-improvement: rather than improving task performance, the model learns to *improve its own efficiency*. This reframes efficiency as a capability: models typically become more capable at the cost of being slower and more expensive to run, whereas a model that improves its efficiency, in principle, becomes *cheaper* as it improves. Crucially, many forms of resource management (e.g.,eviction, routing) naturally take the form of discrete decisions, making them directly amenable to the same reinforcement learning with verifiable rewards (RLVR) framework used to train modern reasoning models.

Resource Rationality, Bounded Rationality, Metacognition. A long tradition in cognitive science and AI studies intelligent behavior under computational constraints, viewing agents as trading off

task performance against the cost of computation [16, 30]. Resource rationality [22] formalizes this perspective by treating cognition as the optimization of task performance subject to resource costs. This work has primarily used resource rationality as a normative framework to explain behavior, rather than as a training objective to produce it in language models. A related tradition studies *metacognition*—the ability of an agent to monitor and regulate its own cognitive processes. Recent work explores metacognitive prompting and *metacognitive reuse*, where models reflect on prior reasoning traces to extract reusable strategies [8]. NGC teaches a model to regulate its own *current* cognitive state by deciding what information to retain under a memory budget. This makes NGC a form of online, prospective metacognition: the model learns not just to reason, but to manage the memory on which its reasoning depends.

Sparse attention, alternative architectures, and conditional computation. Several lines of work improve efficiency by changing how transformers use compute. Sparse attention methods reduce the cost of attending over long contexts by selecting only a subset of keys per query [6, 3]. These methods use attention scores to determine which entries are *read* at a given step, but they do not reduce the size of the cache itself; memory still grows without bound. NGC uses a related scoring mechanism but with fundamentally different semantics: its decisions determine which entries are removed indefinitely, permanently reshaping the context available to future tokens, and providing a practical mechanism for keeping the KV cache constant size. A related line of work uses *conditional computation* to improve efficiency by routing tokens to only a subset of model parameters, as in mixture-of-experts (MoE) architectures [32, 10]. These models show that routing decisions can themselves be learned, but they target a different resource and training regime: MoEs sparsify *parameter compute* through architectural routing whereas NGC sparsifies *memory* during inference by deciding what information to retain under an explicit budget. State-space models [13, 4] achieve efficiency through different computational primitives, but require pre-training from scratch and cannot fully leverage the thriving hardware and software ecosystem specific to transformers. By contrast, NGC requires no modification to the base transformer architecture, re-purposing the existing parameters to perform eviction, and can be applied directly during post-training.

Most crucially, what distinguishes NGC from prior approaches is *how* its discrete decisions are trained. In methods such as DeepSeek Sparse Attention [6], the indexer, which selects which tokens the model attends to, and the main model are not trained under a unified objective. Instead, the indexer is first warm-started to imitate dense attention via a KL-divergence loss against the attention scores, and its input is then explicitly detached from the computational graph throughout sparse training: the indexer continues to receive signal only from the KL-divergence term, while the main model is optimized only by the language modeling loss. The indexer is a separate module trained by the designer to recognize importance according to a proxy criterion, then installed into the system while the main model optimizes around it. NGC instead trains eviction under a unified objective: the same reward signal that shapes reasoning also shapes how it remembers. Moreover, NGC does not require a separate warm-up stage. Likewise, conditional-computation methods such as mixture-of-experts learn routing decisions, but their routers are typically trained with standard backpropagation-based objectives and load-balancing schemes rather than from downstream task reward [10]. In contrast, NGC treats discrete choices that modulate memory use as a first-class action and optimizes them with reinforcement learning.

KV cache compression. KV cache compression methods reduce memory by evicting or compressing cache entries during generation. Static methods apply fixed eviction rules: retaining only the most recent tokens [39] or tokens with high attention scores [21]. More recent methods use richer proxies for importance such as diversity [29] or optimization objectives to compress the cache [9, 26, 45]. Breadcrumbs [25] and Memento [17] compress the KV cache during reasoning, but via a fundamentally different mechanism. Breadcrumbs trains a compressed student policy to match an uncompressed teacher via token-level KL divergence, distilling from the teacher's rollouts. Memento teaches models to produce compressed summaries via SFT on reasoning traces interleaved with summaries generated by a strong teacher prompted with hand-designed rubrics. Both methods inject inductive bias about what good compression looks like—Breadcrumbs via the frozen teacher's reasoning style, Memento via the prompt author's rubrics—rather than letting task reward determine what to remember. Dynamic Memory Compression and Sparsification also involve learnable KV compression [20, 27] but deal with non-differentiability using a stochastic reparameterization trick and use a joint distillation and auxiliary loss. NGC jointly learns how to manage memory and reason,

end-to-end from outcome-based task reward alone using reinforcement learning. This requires no annotation pipeline, no teacher model, no supervised data, and no hand-designed objectives.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We carefully explain the setting and scope of our method in both the intro and abstract and carefully distinguish this from the broader vision of learned efficiency that motivates this work.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We explicitly have a section to discuss limitations.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We provide a numbered and correctly referenced Proposition in the Appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide detailed hyperparameters in the experimental sections of the paper and detailed implementation details both in the main text and in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We will release code for accepted version of the paper.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We carefully describe the data splits, sources, optimizers in both the main text and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Whenever it is possible to compute errorbars, we include them in our plots.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)

- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We disclose this in the Appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We read the Code of Ethics and abide by them.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Our research is basic science and we do not feel there are any direct societal implications that must be specifically highlighted here.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The models are trained using already available open-source models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the creators of the models and datasets appropriately.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not release new assets with the submission version of this paper.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We did not perform human experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: We did not perform human experiments.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.