

CoDA-GQA-L: Bounded-Memory Differential Attention with Value-Routed Landmark Banks

Anthony Maio
Independent Researcher
anthony@making-minds.ai
<https://making-minds.ai>
ORCID: 0009-0003-4541-8515

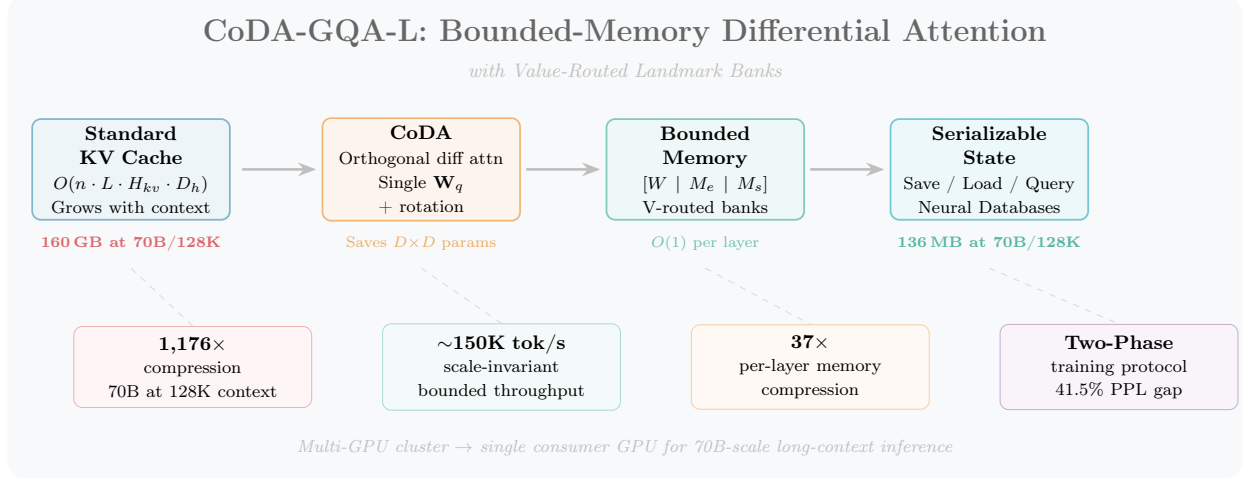
Abstract

We present CoDA-GQA-L, an attention mechanism that provably bounds per-layer KV cache memory to $O(W+M_e+M_s)$ independent of sequence length while retaining selective long-range context through dual memory banks. The architecture combines three innovations: (1) Constrained Orthogonal Differential Attention (CoDA), which sharpens attention by subtracting a gated inhibitory stream produced via learnable orthogonal rotation of the signal query, eliminating the second query projection required by prior differential attention; (2) a dual-bank bounded memory comprising an exact landmark bank for high-fidelity token retention and an EMA summary bank for thematic compression; and (3) value-routed semantic matching that ensures position-invariant memory updates despite RoPE-at-write key storage. A two-phase training protocol first teaches differential attention with full context, then adapts the model to bounded memory. Benchmarks across three model scales (Eve-2, 7B, 70B parameters) on NVIDIA H200 demonstrate up to $37\times$ per-layer memory compression, scale-invariant bounded prefill throughput of $\sim 150\text{K}$ tokens/second regardless of model dimension, and measured compression ratios exceeding $1,100\times$ at 70B scale with 128K context. The bounded state is a fixed-size serializable artifact, enabling a new paradigm of *Stateful Neural Databases* for agentic retrieval-augmented generation.

1 Introduction

The key-value (KV) cache is the dominant memory bottleneck in autoregressive transformer inference. For a model with L layers, H_{kv} KV heads, and head dimension D_h , the cache scales as $O(n \cdot L \cdot H_{kv} \cdot D_h)$ where n is the sequence length. At production scale this becomes prohibitive: a 70B-parameter model serving 128K-token contexts requires approximately 160 GB of KV cache alone—exceeding the memory of even high-end accelerators [8]. This *memory wall* forces practitioners to choose between context length, batch size, and model scale, fundamentally constraining the deployment of long-context language models.

Existing approaches to this problem fall along a spectrum of aggressiveness. Sliding-window attention [6] and attention-sink methods [23] enforce hard recency boundaries, discarding all information beyond a fixed horizon. Token eviction strategies such as H₂O [26] and Scissorhands [10] selectively prune the cache based on attention scores, but their greedy heuristics lack semantic awareness and provide no mechanism for compressing evicted context. Memory-augmented approaches like Memorizing Transformers [22] and Landmark Attention [11] extend context via retrieval, but maintain $O(n)$ total storage. Meanwhile, the Differential Transformer [24] improves attention quality



Graphical Abstract. CoDA-GQA-L transforms unbounded KV caches into fixed-size serializable states through constrained orthogonal differential attention and value-routed dual memory banks, achieving over 1,100 \times compression at 70B scale with scale-invariant throughput.

by canceling noise through dual softmax subtraction, but doubles the query parameter count and operates with unbounded KV cache.

No existing method simultaneously provides (i) a provable constant-memory bound independent of sequence length, (ii) learned selective retention with both exact and compressed memory, (iii) position-invariant memory routing compatible with RoPE-at-write efficiency, and (iv) a practical training protocol that adapts pretrained models to bounded inference.

We address this gap with CoDA-GQA-L, which makes the following contributions:

1. **Constrained Orthogonal Differential Attention (CoDA):** A parameter-efficient variant of differential attention that produces the inhibitory query via learnable orthogonal rotation of the signal query, saving $D \times D$ parameters compared to a second projection while preserving noise-cancellation properties (§3.1).
2. **Bounded dual-bank KV memory:** A three-segment buffer [Recent W | Exact M_e | Summary M_s] that provably bounds per-layer cache to $O(W + M_e + M_s)$. The exact bank preserves high-fidelity tokens via novelty-filtered LRU, while the summary bank compresses background context through EMA prototype learning (§3.2).
3. **Value-routed semantic matching:** Memory bank updates route on Values (RoPE-free) rather than Keys (RoPE-contaminated), solving the fundamental tension between RoPE-at-write efficiency and position-dependent key similarity (§3.3).
4. **Two-phase training protocol:** Phase 1 trains differential attention with full context; Phase 2 adapts the model to bounded memory with gradient flow through bank updates (§3.5).
5. **Stateful Neural Databases:** The bounded state is a fixed-size serializable artifact (54 MB for 7B, 136 MB for 70B) that enables save/load/query semantics for agentic RAG, achieving over 1,100 \times compression at 128K context (§5).

We validate CoDA-GQA-L across three model scales on NVIDIA H200, demonstrating 18–37 \times per-layer memory compression at $L=2,048$, scale-invariant bounded prefill throughput of $\sim 150\text{K}$

tokens/second, and $1.9\times$ lower peak VRAM at 70B scale. A two-phase fine-tuning on SmolLM2-135M [2] achieves bounded perplexity within 41.5% of unbounded quality under $5.3\times$ context compression. The complete implementation, including 60 passing tests and drop-in adapters for Llama-family models, is open-sourced.

2 Background

2.1 Scaled Dot-Product Attention and GQA

Standard scaled dot-product attention computes $\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^\top / \sqrt{D_h})\mathbf{V}$, where $\mathbf{Q} \in \mathbb{R}^{n \times D_h}$, $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{m \times D_h}$. In multi-head attention (MHA) [20], each of H heads maintains independent key and value projections, yielding a KV cache of size $O(n \cdot H \cdot D_h)$ per layer.

Grouped-Query Attention (GQA) [1] reduces this by sharing KV heads across groups of query heads: H_q query heads share $H_{kv} < H_q$ KV heads, reducing cache size by a factor of H_q/H_{kv} . This represents an interpolation between full MHA ($H_{kv} = H_q$) and Multi-Query Attention [17] ($H_{kv} = 1$), with GQA retaining 99% of MHA quality while halving the KV cache. Modern architectures including Llama 2 [19] and Mistral 7B [6] adopt GQA as standard.

Despite GQA’s compression, the cache still grows linearly with sequence length n , creating the fundamental memory wall for long-context inference.

2.2 Differential Attention

The Differential Transformer [24] addresses the tendency of standard attention to over-allocate scores to irrelevant context by computing two separate attention maps and subtracting one from the other:

$$\text{DiffAttn}(\mathbf{x}) = \text{SDPA}(\mathbf{W}_{q_1}\mathbf{x}, \mathbf{K}, \mathbf{V}) - \lambda \cdot \text{SDPA}(\mathbf{W}_{q_2}\mathbf{x}, \mathbf{K}, \mathbf{V}) \quad (1)$$

where λ is a learnable scalar and $\mathbf{W}_{q_1}, \mathbf{W}_{q_2} \in \mathbb{R}^{D \times D_h}$ are two independent query projections. This subtraction cancels shared noise in both attention distributions, yielding a $10\times$ improvement in attention signal-to-noise ratio and sparser, more discriminative patterns.

However, the dual query projection doubles the parameter count for queries ($2 \times D \times D$ for H heads), and the original formulation operates with unbounded KV cache. Our CoDA variant (§3.1) eliminates the second projection entirely through orthogonal rotation.

2.3 Rotary Position Embeddings

Rotary Position Embeddings (RoPE) [18] encode positional information by applying a rotation matrix $\mathbf{R}_\Theta(p)$ to query and key vectors at position p :

$$\mathbf{q}_p = \mathbf{R}_\Theta(p) \mathbf{W}_q \mathbf{x}_p, \quad \mathbf{k}_p = \mathbf{R}_\Theta(p) \mathbf{W}_k \mathbf{x}_p \quad (2)$$

where $\mathbf{R}_\Theta(p)$ applies pairwise rotations with frequencies $\theta_i = 10000^{-2i/D_h}$. The key property is that the dot product $\mathbf{q}_m^\top \mathbf{k}_n$ depends only on the relative position $|m - n|$, enabling length generalization.

A critical observation for memory bank design: because keys are rotated by position-dependent angles, the cosine similarity $\cos(\mathbf{k}_p, \mathbf{k}_{p'})$ between keys of identical tokens at different positions $p \neq p'$ is *not* preserved. This position-dependence makes key-based routing unreliable for memory banks that store tokens from diverse positions—the central motivation for our value-routing approach (§3.3).

3 Method

CoDA-GQA-L consists of five integrated components: constrained orthogonal differential attention (§3.1), bounded dual-bank KV memory (§3.2), value-routed semantic matching (§3.3), dense packing for FlashAttention (§3.4), and a two-phase training protocol (§3.5).

3.1 CoDA: Constrained Orthogonal Differential Attention

We replace the dual query projection of Eq. 1 with a single projection followed by learnable orthogonal rotation:

$$\mathbf{q}_{\text{sig}} = \text{RoPE}(\mathbf{W}_q \mathbf{x}, p) \quad (3)$$

$$\mathbf{q}_{\text{noise}} = \mathbf{R}(\boldsymbol{\theta}) \cdot \mathbf{q}_{\text{sig}} \quad (4)$$

$$\lambda = \sigma(\mathbf{W}_\lambda \mathbf{x} + b_\lambda) \quad (5)$$

$$\mathbf{o} = \text{RMSNorm}_h(\text{SDPA}(\mathbf{q}_{\text{sig}}, \mathbf{K}, \mathbf{V}) - \lambda \cdot \text{SDPA}(\mathbf{q}_{\text{noise}}, \mathbf{K}, \mathbf{V})) \quad (6)$$

Orthogonal rotation $\mathbf{R}(\boldsymbol{\theta})$. Each attention head applies a pairwise Givens rotation to consecutive feature pairs (x_{2i}, x_{2i+1}) :

$$\begin{bmatrix} x'_{2i} \\ x'_{2i+1} \end{bmatrix} = \begin{bmatrix} \cos \theta_i & -\sin \theta_i \\ \sin \theta_i & \cos \theta_i \end{bmatrix} \begin{bmatrix} x_{2i} \\ x_{2i+1} \end{bmatrix} \quad (7)$$

where $\boldsymbol{\theta} \in \mathbb{R}^{H \times D_h/2}$ are learnable rotation angles. This guarantees that $\|\mathbf{q}_{\text{noise}}\| = \|\mathbf{q}_{\text{sig}}\|$ and that the noise query spans an orthogonal subspace, producing maximally decorrelated attention patterns for effective noise cancellation.

Parameter efficiency. CoDA adds only $H \times D_h/2$ rotation angles and a $D \rightarrow H$ cancellation gate projection, compared to $D \times D$ parameters for a second query projection. For a 7B model with $D=4096$, this saves approximately 16.7M parameters.

Near-transparent initialization. The cancellation gate bias is initialized to $b_\lambda = -6.0$, yielding $\sigma(-6) \approx 0.0025$ at initialization. This ensures that CoDA begins as near-identity, with the differential mechanism activating gradually during training.

Head-stacked SDPA. Rather than two separate kernel calls, we concatenate $[\mathbf{q}_{\text{sig}}; \mathbf{q}_{\text{noise}}]$ along the head dimension and perform a single SDPA call, halving kernel launch overhead.

HeadwiseRMSNorm. Following the differential subtraction, we apply per-head RMS normalization with a learned scale vector $\boldsymbol{\gamma} \in \mathbb{R}^{D_h}$ per head [25]. This stabilizes the potentially sign-variable output of the subtraction and provides a learned rescaling mechanism.

3.2 Bounded KV Memory

The “-L” suffix in CoDA-GQA-L denotes the bounded memory system. We replace the standard $O(n)$ KV cache with a fixed-size three-segment buffer:

$$\mathbf{K}_{\text{buf}}, \mathbf{V}_{\text{buf}} \in \mathbb{R}^{B \times H_{kv} \times L_{\text{buf}} \times D_h}, \quad L_{\text{buf}} = W + M_e + M_s \quad (8)$$

where the buffer is partitioned into three contiguous segments:

CoDA: Constrained Orthogonal Differential Attention

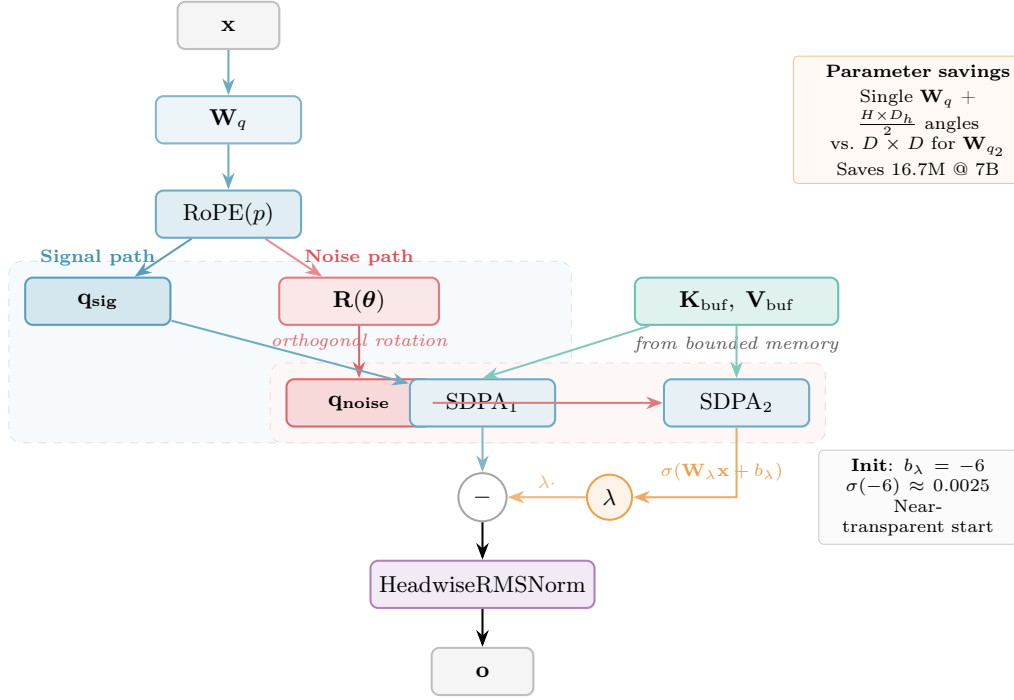


Figure 1: CoDA attention mechanism. The signal query \mathbf{q}_{sig} is produced via a single projection \mathbf{W}_q followed by RoPE; the noise query $\mathbf{q}_{\text{noise}}$ is obtained by learnable orthogonal rotation. Both attend to the shared bounded KV buffer. A learned gate $\lambda = \sigma(\mathbf{W}_\lambda \mathbf{x} + b_\lambda)$ controls the subtraction weight, initialized near zero for transparent warm-start.

Recent Window W	Exact Bank M_e	Summary Bank M_s
slots $[0, W-1]$	slots $[W, W+M_e-1]$	slots $[W+M_e, L_{\text{buf}}-1]$

Recent window. A ring buffer storing the W most recent tokens with FIFO eviction. Each token receives a write gate $g = \sigma(\mathbf{W}_{\text{write}} \mathbf{x})$ at insertion time, which controls downstream bank eligibility.

Exact landmark bank. A novelty-filtered LRU cache of size M_e . When a token is evicted from the recent window with gate $g \geq \tau_{\text{exact}}$, its value vector is compared to existing bank entries via cosine similarity (using V-routing, §3.3). If the maximum similarity is below the novelty threshold $\tau_{\text{novel}} = 0.70$, the token is inserted into a free slot or replaces the least-recently-used entry. If similarity exceeds the match threshold $\tau_{\text{match}} = 0.90$, the LRU timestamp is refreshed. This preserves high-fidelity representations of semantically novel tokens—critical for needle-in-haystack retrieval.

Summary landmark bank. An EMA prototype memory of size M_s . On eviction, the best-matching summary slot is identified by V-routing cosine similarity, and updated via exponential moving average:

$$\eta_{\text{eff}} = \sigma(\eta_{\text{logit}}) \cdot g, \quad \mathbf{m}_{\text{slot}} \leftarrow \mathbf{m}_{\text{slot}} + \eta_{\text{eff}}(\mathbf{t}_{\text{evict}} - \mathbf{m}_{\text{slot}}) \quad (9)$$

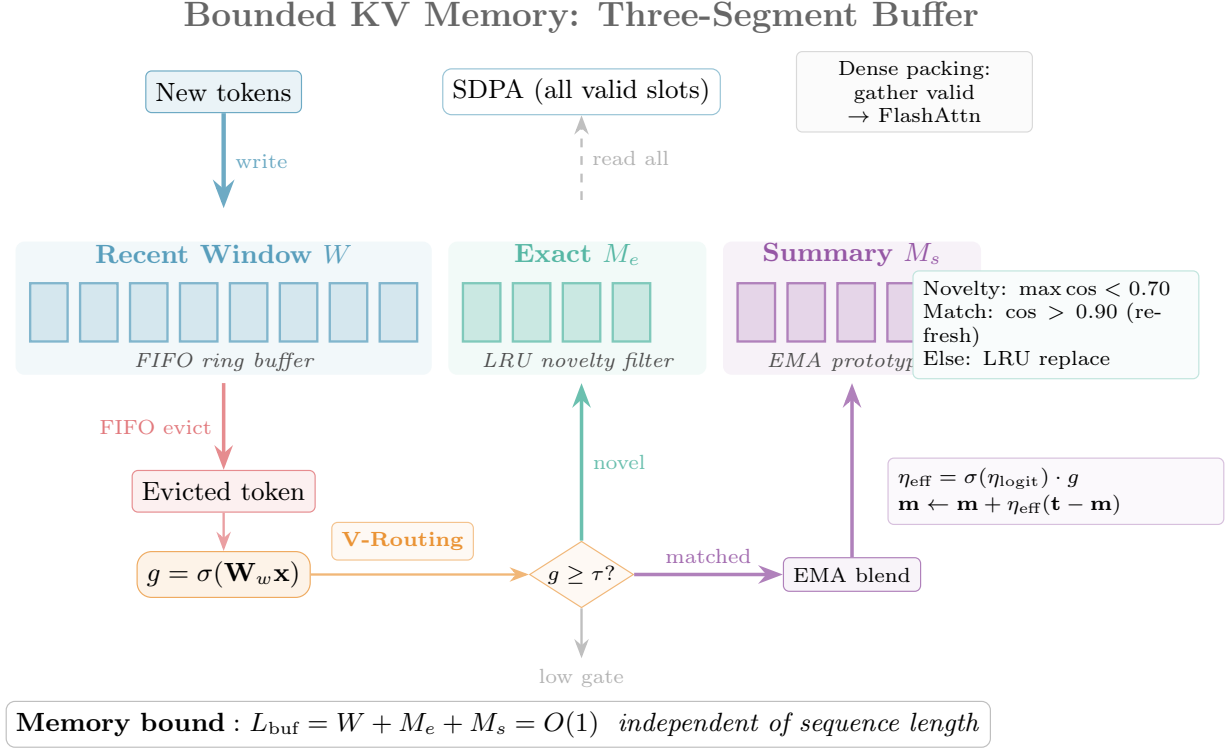


Figure 2: Bounded KV memory data flow. New tokens enter the recent window (FIFO ring buffer); evicted tokens pass through a write gate and are routed via value cosine similarity to either the exact bank (novelty-filtered LRU) or summary bank (EMA prototypes). The memory bound $L_{\text{buf}} = W + M_e + M_s$ is independent of sequence length.

where η_{logit} is a learned blending rate. For the first insertion into an empty slot, $\eta = 1$ (immediate copy). The summary bank captures thematic clusters of older context, providing compressed but semantically meaningful background information.

Memory bound.

Theorem 1. *The KV cache size per layer is bounded by $2 \cdot H_{kv} \cdot (W + M_e + M_s) \cdot D_h \cdot \text{sizeof}(dtype)$, independent of the input sequence length n .*

Proof. The buffer has fixed size $L_{\text{buf}} = W + M_e + M_s$. The recent window overwrites in-place via ring indexing. The exact bank replaces entries (LRU eviction). The summary bank updates entries via EMA blending. No operation appends beyond L_{buf} . \square

3.3 Value-Routed Semantic Matching

The position-dependence problem. In our architecture, keys receive RoPE rotation at write time (*RoPE-at-write*) for decode efficiency—this avoids $O(L_{\text{buf}})$ re-rotations per decode step. However, RoPE makes key cosine similarity position-dependent: semantically identical tokens at positions p and p' produce keys $\mathbf{k}_p = \mathbf{R}_{\Theta}(p) \mathbf{W}_k \mathbf{x}$ and $\mathbf{k}_{p'} = \mathbf{R}_{\Theta}(p') \mathbf{W}_k \mathbf{x}$ with $\cos(\mathbf{k}_p, \mathbf{k}_{p'}) \ll 1$ when $|p - p'|$ is large. This invalidates key-based routing for deduplication and clustering.

Value-Routed Semantic Matching

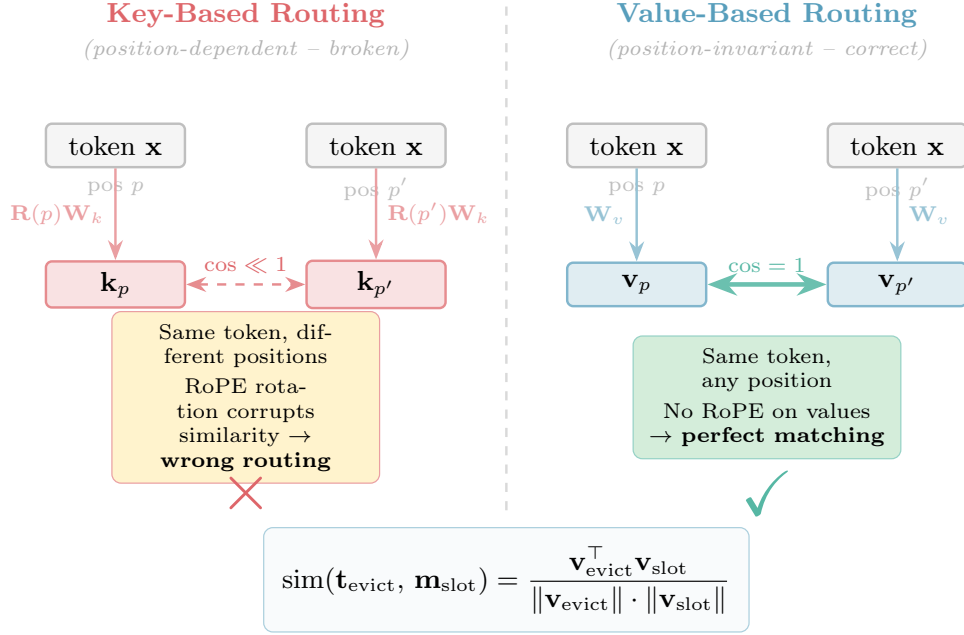


Figure 3: Value-routed vs. key-based semantic matching. Keys receive RoPE rotation, making their cosine similarity position-dependent—identical tokens at different positions yield $\cos \ll 1$. Values are RoPE-free, preserving $\cos = 1$ for identical inputs regardless of position, enabling correct deduplication and clustering.

Routing via values. We observe that value vectors $\mathbf{v} = \mathbf{W}_v \mathbf{x}$ are *not* subject to RoPE rotation, making $\cos(\mathbf{v}_p, \mathbf{v}_{p'}) = 1$ for identical inputs regardless of position. We therefore route all memory bank updates using normalized value vectors:

$$\text{sim}(\mathbf{t}_{\text{evict}}, \mathbf{m}_{\text{slot}}) = \frac{\mathbf{v}_{\text{evict}}^\top \mathbf{v}_{\text{slot}}}{\|\mathbf{v}_{\text{evict}}\| \cdot \|\mathbf{v}_{\text{slot}}\|} \quad (10)$$

This ensures position-invariant deduplication in the exact bank and semantically coherent prototype formation in the summary bank, with zero memory overhead (we cache normalized value vectors rather than allocating separate routing features).

Phase-safe EMA for summary keys. While values are used for routing, the summary bank must also store keys for attention computation. RoPE splits key dimensions into high-frequency (HF, position-sensitive) and low-frequency (LF, position-invariant) bands; directly blending RoPE-rotated keys from different positions via EMA causes destructive interference in the HF bands, as formalized in the spectral analysis of Wang et al. [21]. We address this with *LF-K routing*: the summary bank stores only the low-frequency band of keys (which varies slowly with position) and zeros the HF band, preventing frequency aliasing during EMA blending.

3.4 Dense Packing for FlashAttention

The bounded buffer contains both valid and empty slots, requiring a boolean mask `allowed` to exclude uninitialized entries. However, custom attention masks disable FlashAttention [4] and memory-efficient SDPA backends, forcing fallback to the $O(n^2)$ math kernel.

For the common case of batch size $B=1$, we *dense-pack* only valid slots before SDPA:

- **Prefill:** Gather valid prefix + current block tokens, invoke SDPA with `is_causal=True` (bottom-right causal alignment).
- **Decode:** Gather all valid slots, invoke SDPA with `is_causal=False` (single query attends to all valid keys).

This eliminates the boolean mask entirely, unlocking FlashAttention and MemoryEfficient SDPA backends. For $B > 1$, we fall back to the explicit mask path, as FlashAttention does not support per-batch masks.

3.5 Two-Phase Training Protocol

Training CoDA-GQA-L requires a two-phase protocol because the bounded memory components (write gate, bank routing, EMA blending) are untrained after standard attention fine-tuning.

Motivation: cold-swap catastrophe. Training only Phase 1 (unbounded CoDA-GQA) and directly evaluating with bounded memory produces catastrophic perplexity. On Mistral 7B, unbounded perplexity of 5.62 degrades to 2,464 with bounded evaluation—a $438\times$ degradation. The memory banks, initialized with default parameters, provide no compensation for the lost context.

Phase 1: unbounded training. We fine-tune the pretrained model with CoDA-GQA (full KV cache) for T_1 steps. This phase teaches the differential mechanism (θ , λ , HeadwiseRMSNORM) to perform signal/noise decomposition with full context available. The near-transparent initialization ($\lambda \approx 0.0025$) ensures a gradual transition from standard to differential attention.

Phase 2: bounded adaptation. We copy trained weights from Phase 1 and switch to CoDA-GQA-L with fixed-size KV cache for T_2 steps. Bounded-only parameters (write gate $\mathbf{W}_{\text{write}}$, EMA rate η_{logit}) start at defaults. We set `detach_evicted=False`, enabling gradients to flow through evicted tokens into bank update operations so that the write gate can learn selectivity. The learning rate is reduced to $0.5\times$ Phase 1, and the training block size is set smaller than the window ($B_{\text{block}} < W$) to force frequent evictions that exercise the memory banks.

Gradient flow considerations. The `detach_evicted=False` setting requires disabling gradient checkpointing (which conflicts with in-place graph-connected writes) and careful clone-based buffer management to avoid in-place mutation version conflicts in the autograd graph.

4 Experiments

4.1 Experimental Setup

Training model. We validate the two-phase training protocol on SmolLM2-135M [2], a Llama-architecture model with 30 layers, GQA 9:3 ($H_q=9$, $H_{kv}=3$), $D=576$, $D_h=64$. All 30 attention layers are replaced with CoDA-GQA-L adapters.

Benchmark models. For throughput and memory benchmarks, we evaluate at three model scales using single-layer attention modules: Eve-2 scale ($D=512$, $H=8$, $H_{kv}=2$), 7B scale ($D=4096$, $H=32$, $H_{kv}=8$), and 70B scale ($D=8192$, $H=64$, $H_{kv}=8$).

Hardware. NVIDIA H200 NVL on RunPod. All experiments use bf16 precision.

Table 1: Bounded memory configurations evaluated. $L_{\text{buf}} = W + M_e + M_s$.

Config	W	M_e	M_s	L_{buf}
tiny-cache	128	32	32	192
medium-cache	256	64	64	384
window-only	256	0	0	256

Bounded configurations.

Training details. Dataset: WikiText-103 (train split). Sequence length: 2,048. Freeze strategy: train full adapter weights, freeze non-attention layers. Phase 1: 2,000 steps, LR 1×10^{-3} (CoDA params), 5×10^{-5} (projections), gradient checkpointing enabled. Phase 2: 2,000 steps, LR 5×10^{-4} / 2.5×10^{-5} , block size 128, `detach_evicted=False`, gradient checkpointing disabled. Optimizer: AdamW with weight decay 0.01.

4.2 Weight Transfer Validation

We first verify that the weight transfer infrastructure correctly maps pretrained GQA weights into our adapter format. Replacing SmolLM2-135M attention layers with BaselineGQA (our standard GQA implementation) yields:

- 100% top-1 prediction agreement (bf16)
- Mean logit difference: 0.156 (bf16), exact match in fp32
- Perplexity: 12.31 vs. 12.31 (+0.00%) in fp32

In contrast, CoDA cold-swap (no training) produces 0% top-1 agreement, mean logit difference 7.99, and perplexity 3,371,482—confirming that the differential mechanism fundamentally reshapes the activation manifold and requires fine-tuning.

4.3 Bounded Correctness

To verify that the bounded code path introduces no numerical artifacts, we test with window $W \geq n$ (no evictions, no bank usage):

The differences are within float32 machine precision ($\sim 10^{-7}$), proving that all quality degradation at smaller windows stems solely from context compression, not implementation bugs.

4.4 Two-Phase Training Results

Phase 1: unbounded training. Training SmolLM2-135M with CoDA-GQA (full KV cache) for 2,000 steps reduces unbounded perplexity from 70.0 to 22.0, demonstrating effective learning of the differential attention decomposition (Table 3).

Table 2: Maximum absolute difference between bounded and unbounded outputs when $W \geq n$ (float32).

W	n	Max $ \Delta $
512	512	1.2×10^{-7}
1024	1024	1.5×10^{-7}
2048	2048	1.8×10^{-7}

Table 3: Phase 1 training on SmolLM2-135M (unbounded CoDA-GQA, WikiText-103). Throughput: ~ 36 K tok/s. Peak VRAM: 2.9 GB.

Step	Unbounded PPL	Train Loss
0	70.00	—
200	61.75	3.93
400	37.00	3.57
600	27.50	3.18
1000	24.25	3.10
1400	23.12	3.21
2000	22.00	3.23

At Phase 1 completion, evaluating with bounded memory (medium-cache) yields perplexity 31.12—the cold-swap baseline for Phase 2.

Phase 2: bounded adaptation. Phase 2 trains with bounded KV cache ($W=256$, $M_e=64$, $M_s=64$, block size 128) for 2,000 steps (Table 4).

Table 4: Phase 2 training on SmolLM2-135M (bounded CoDA-GQA-L, medium-cache). Throughput: ~ 1.8 K tok/s. Peak VRAM: ~ 8 GB.

Step	Bounded PPL	Train Loss
0	35.75	—
200	38.75	3.85
400	35.25	3.43
600	32.50	3.55
1000	31.62	3.37
1200	31.12	3.28
1400	31.12	3.25
2000	31.12	~ 2.9

Bounded perplexity plateaus at 31.12 from step 1,200 onward, while training loss continues to decrease (2.96 at step 1,600)—indicating train/evaluation divergence at this model scale. The final metrics are:

- Unbounded PPL: 22.0, Bounded PPL: 31.12
- PPL gap: $(31.12 - 22.0)/22.0 = 41.5\%$
- Context compression: 384 effective tokens vs. 2,048 = $5.3\times$

The bounded PPL floor (31.12) matches the Phase 1 cold-swap evaluation, suggesting that Phase 2 successfully recovered from warmup disruption but the natural quality floor at 135M

parameters prevents further improvement—a scale-dependent phenomenon we expect to improve at larger model sizes.

Two-Phase Training: SmolLM2-135M on WikiText-103

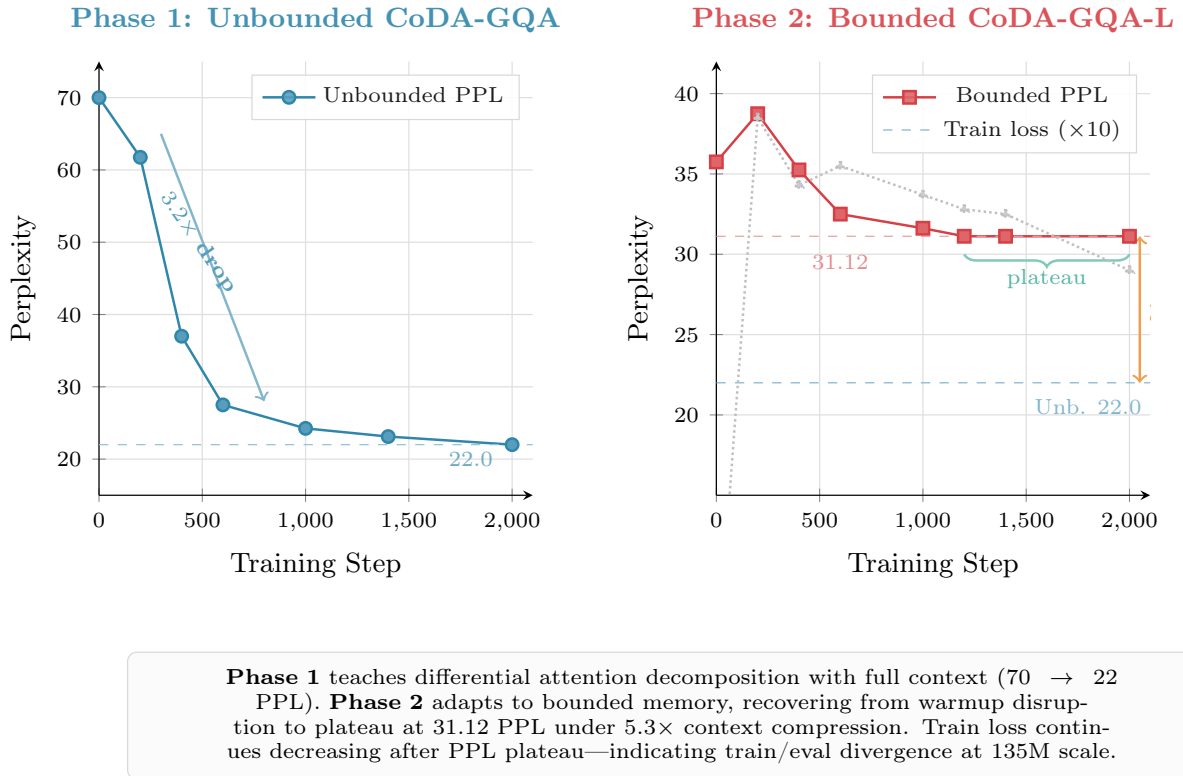


Figure 4: Two-phase training dynamics on SmolLM2-135M. **Left:** Phase 1 (unbounded CoDA-GQA) reduces perplexity from 70 to 22 over 2,000 steps. **Right:** Phase 2 (bounded CoDA-GQA-L) perplexity plateaus at 31.12 from step 1,200, while training loss continues decreasing—indicating train/eval divergence at this model scale. The 41.5% PPL gap represents the information-theoretic cost of 5.3 \times context compression.

4.5 Phase 2 Hyperparameter Sensitivity

A comparison between two Phase 2 configurations demonstrates the importance of hyperparameter tuning:

Run 1 showed no improvement: the learning rate was too low for bank parameter adaptation, block size equaled window size (minimal evictions, banks barely exercised), and 500 steps provided

Table 5: Phase 2 hyperparameter comparison on SmolLM2-135M.

	Run 1	Run 2
LR scale	0.1×	0.5×
Block size	256 ($= W$)	128 ($< W$)
Steps	500	2,000
Final bounded PPL	31.62 (flat)	31.12

insufficient optimization time. Run 2’s higher LR, smaller block size (forcing frequent evictions), and longer training enabled effective bank learning.

4.6 Memory Savings

Table 6 reports measured KV cache sizes across three model scales at $L=2,048$, bf16.

Table 6: Per-layer KV cache memory (bf16, $L=2,048$) and compression ratios across model scales.

Config	Eve-2	7B	70B
baseline	2.0 MB	32.0 MB	32.0 MB
tiny-cache	108.9 KB (18×	864.9 KB (37 ×	864.9 KB (37 ×
medium-cache	217.9 KB (9×	1.7 MB (18.8 ×	1.7 MB (18.8 ×
window-only	129.2 KB (15×	1.0 MB (32×	1.0 MB (32×

At 7B scale across 32 layers, medium-cache requires 54 MB total versus 1 GB baseline. At $L=128K$, the baseline scales to 64 GB while the bounded configuration remains at 54 MB—constant regardless of context length. At 70B across 80 layers with 128K context, the savings are 136 MB versus 160 GB, a **1,176**× compression that represents the difference between a multi-GPU cluster and a single consumer GPU.

4.7 Throughput Analysis

Table 7 reports prefill throughput (tokens/second) across model scales on H200 NVL.

Table 7: Prefill throughput (tok/s) across model scales (H200 NVL, bf16). Decode throughput and peak VRAM in parentheses for 70B scale.

Config	$L=512$	$L=2048$	$L=4096$	$L=8192$
<i>70B scale ($D=8192$, $H=64$, $H_{kv}=8$)</i>				
baseline	1,184K	1,336K	1,220K	967K
coda-unb.	813K	889K	786K	598K
medium	182K	150K	146K	154K
window	367K	359K	356K	356K
<i>7B scale ($D=4096$, $H=32$, $H_{kv}=8$)</i>				
baseline	2,232K	3,096K	2,859K	2,286K
coda-unb.	1,121K	1,852K	1,657K	1,283K
medium	204K	160K	159K	158K
window	380K	392K	377K	397K

Prefill Throughput: 70B Scale (H200 NVL)

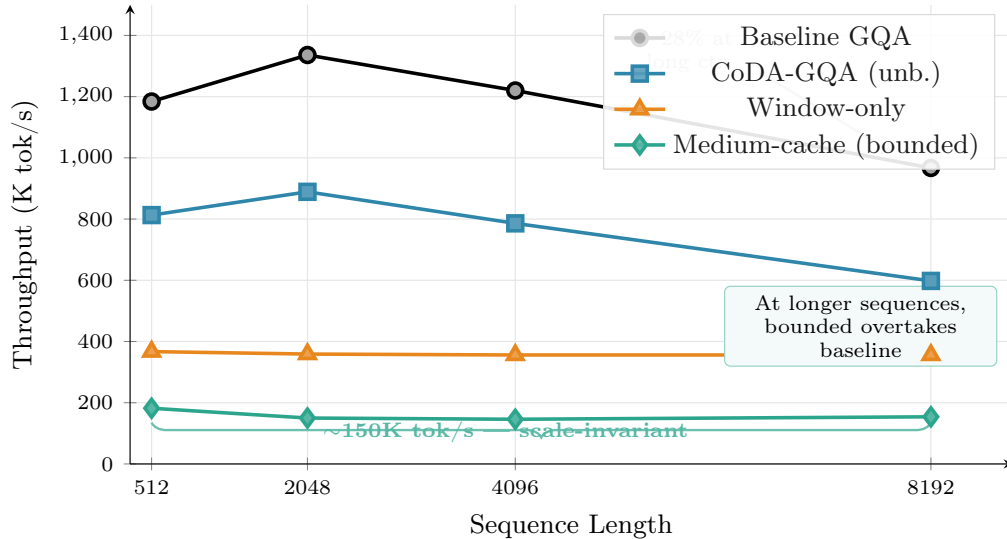


Figure 5: Prefill throughput at 70B scale on H200 NVL. Baseline throughput drops 28% from $L=2,048$ to $L=8,192$ due to $O(n^2)$ scaling, while bounded (medium-cache) throughput remains flat at $\sim 150\text{K tok/s}$ —the same rate observed at Eve-2 and 7B scales, demonstrating scale-invariance of bank update computation.

Several key observations emerge:

Scale-invariant bounded throughput. Bounded prefill operates at $\sim 150\text{K tok/s}$ regardless of sequence length *or model dimension*—the same throughput at Eve-2 ($D=512$), 7B ($D=4096$), and 70B ($D=8192$). This is because bank update computation (scatter operations, cosine similarity) operates on fixed-size buffers of 384 slots, independent of the embedding dimension.

Crossover at long sequences. Baseline throughput drops from 1.34M to 967K tok/s at 70B going from $L=2,048$ to $L=8,192$ (-28% , due to $O(n^2)$ scaling), while bounded throughput remains flat. At sufficiently long sequences, bounded prefill overtakes baseline.

Lower peak VRAM. At 70B, bounded (medium-cache) uses 569 MB versus 1.1 GB baseline ($1.9\times$ less), as the smaller KV cache more than offsets differential attention overhead.

Differential overhead. CoDA-unbounded achieves 62–67% of baseline throughput across all scales, reflecting the constant $2\times$ SDPA cost from dual-stream attention. A fused Triton kernel would largely close this gap.

Bank update dominance. Medium-cache prefill is 11–15% of baseline, while window-only (no bank updates) achieves 27–31%. The bank update path, with ~ 20 micro-kernel launches per eviction, is the primary bottleneck.

Decode efficiency. At 70B, medium-cache decode achieves 1,753 tok/s versus 4,676 baseline

(37.5%). Bank updates are amortized during decode, triggered only on ring buffer eviction rather than every token.

4.8 Test Suite

The implementation includes 60 passing tests organized across five categories: *correctness* (shapes, NaN/Inf detection, dtype consistency), *determinism* (seeded reproducibility), *edge configurations* (zero banks, minimal windows, varying batch sizes), *invariants* (causal masking, GQA head alignment, ring buffer wrapping, normalization cache consistency, LRU timestamps, monotonic bank growth), and *backward pass* (autograd safety with `detach_evicted=False`: multi-block prefill, single block, batch=2, write gate gradient flow).

5 Application: Stateful Neural Databases

5.1 Bounded State as Serializable Artifact

A distinctive property of CoDA-GQA-L is that the bounded KV state is a *fixed-size, serializable artifact* that can be saved, loaded, and queried independently of the input that created it:

- **Save:** `torch.save(state, "context.pt")`—a single file per layer
- **Load:** `state = torch.load("context.pt")`—instant, no re-reading documents
- **Query:** Load state, run `model.step(question_tokens)` to generate answers

This contrasts sharply with standard KV caches, which grow with context length and must be reconstructed from scratch for each session.

5.2 Compression Ratios at Scale

Table 8 reports measured compression ratios for the medium-cache configuration across model and context scales. All numbers are from the H200 benchmark suite using bf16.

Table 8: Measured KV state compression ratios (medium-cache, bf16). CoDA-GQA-L state is constant regardless of context length.

Scenario	Standard KV	CoDA State	Compression
7B, 2K ctx (32 layers)	1.0 GB	54 MB	18.8×
7B, 32K ctx (32 layers)	16 GB	54 MB	296×
7B, 128K ctx (32 layers)	64 GB	54 MB	1,185×
70B, 2K ctx (80 layers)	2.56 GB	136 MB	18.8×
70B, 32K ctx (80 layers)	40 GB	136 MB	294×
70B, 128K ctx (80 layers)	160 GB	136 MB	1,176×

At 70B with 128K context, the bounded state saves 159.9 GB—the difference between requiring two H100-80GB GPUs and fitting on a single consumer accelerator.

KV State Compression at Scale

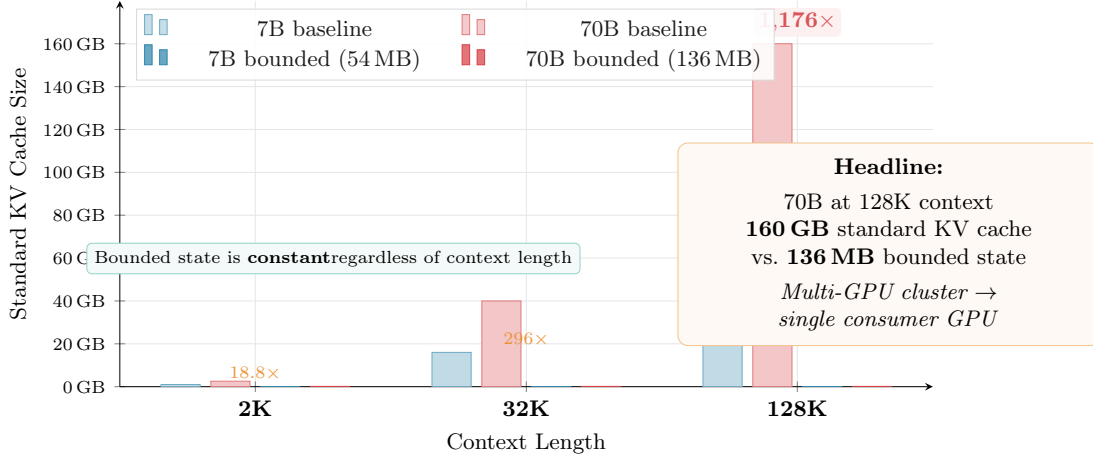


Figure 6: KV state compression across model and context scales (medium-cache, bf16). Standard KV cache grows linearly with context length, while bounded state remains constant at 54 MB (7B) and 136 MB (70B). At 128K context, compression exceeds 1,100 \times for both scales.

5.3 Recursive Neural Database Architecture

The serializable state enables a new paradigm for retrieval-augmented generation, where compressed context states become first-class data artifacts in agentic workflows:

1. **Ingestion:** A document is processed via chunked prefill, producing a compressed state artifact (54 MB for 7B regardless of document length).
2. **Registry:** State artifacts are stored in a key-value registry mapping document identifiers to serialized states on disk or object storage.
3. **Query:** An agent loads the appropriate state and generates answers via decode, with sub-second latency (no re-reading or re-encoding).
4. **Routing:** A meta-agent routes queries to the appropriate state based on document summaries or user intent.

For 100 documents at 7B scale, the total state footprint is $100 \times 54 \text{ MB} = 5.4 \text{ GB}$ —feasible for a single GPU. This enables multi-document question answering without re-processing any source material, with each query requiring only a decode-phase forward pass.

Stateful Neural Database Architecture

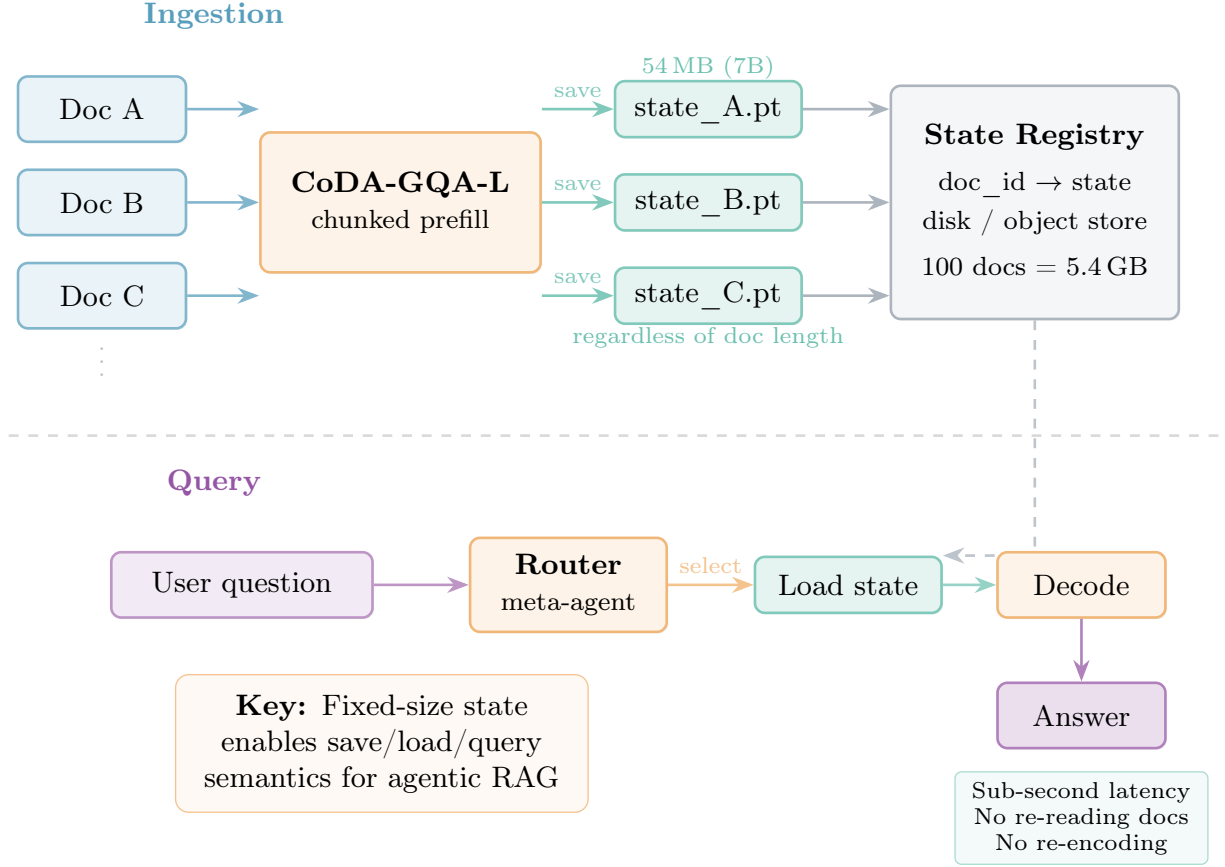


Figure 7: Stateful Neural Database architecture. Documents are processed via chunked prefill into fixed-size state artifacts (54 MB for 7B, 136 MB for 70B). A state registry maps document identifiers to serialized states. At query time, the appropriate state is loaded and decoded with sub-second latency—no re-reading or re-encoding required.

6 Related Work

KV cache eviction. H₂O [26] identifies “heavy hitter” tokens via attention scores and maintains ~20% of the cache with 5× memory reduction and 29× throughput gains. Scissorhands [10] exploits the persistence of token importance across layers. Both use greedy, attention-score-based heuristics without semantic clustering or learned write gating. StreamingLLM [23] discovers *attention sinks*—initial tokens that stabilize softmax—and maintains them with a sliding window, achieving up to 22× speedup for streaming inference. However, all three approaches discard evicted tokens irrecoverably and cannot selectively compress long-range context.

Memory-augmented transformers. Transformer-XL [3] introduces segment-level recurrence for cross-segment context propagation but sacrifices random access. Memorizing Transformers [22] augment attention with differentiable kNN retrieval over a growing memory, achieving linear scaling but maintaining $O(n)$ total storage. Landmark Attention [11] uses dedicated landmark tokens for

block-level retrieval, enabling random-access infinite context but requiring fine-tuning for block selection. Compressive Transformers [15] learn to compress evicted memories but operate with a single compression function rather than dual exact/summary banks. Infini-attention [12] proposes bounded attention via compressive memory with linear complexity. Our dual-bank architecture, with separate exact (LRU) and summary (EMA) banks plus value-routed matching, is unique in combining high-fidelity needle retention with thematic compression under a provable memory bound.

Differential attention. The Diff Transformer [24] achieves $10\times$ attention signal-to-noise improvement through dual softmax subtraction, with demonstrated benefits across language modeling, retrieval, and hallucination mitigation. Our CoDA variant replaces the second query projection with orthogonal rotation, saving $D\times D$ parameters per head while preserving the noise-cancellation property. The connection to orthogonal parameterization in neural networks [14] provides theoretical grounding for the rotation-based approach.

KV cache compression. GQA [1] reduces cache via head sharing; quantization methods like GEAR [7] achieve near-lossless 2-bit KV storage with error correction; adaptive profiling [5] identifies head-specific patterns for targeted eviction. These are complementary to our approach—quantized KV storage could be applied to CoDA-GQA-L’s fixed-size buffers for additional compression.

Position embeddings. RoPE [18] enables relative position encoding via rotation, with extensions like YaRN [13] for context scaling. Prism [21] provides spectral analysis showing that RoPE key dimensions decompose into high-frequency (position-sensitive) and low-frequency (position-invariant) bands, formalizing the insight that standard pooling creates blind spots for positional signals. Our work builds on this spectral perspective: we identify the position-dependence of RoPE key similarity as a fundamental obstacle for memory bank routing and address it through value-based matching (for routing) and LF-K routing (for summary key storage). The lost-in-the-middle phenomenon [9] further motivates architectures that can selectively retain information from arbitrary positions.

7 Limitations

We identify several limitations of the current CoDA-GQA-L implementation:

Differential FLOP overhead. The dual-stream attention mechanism incurs a constant $2\times$ SDPA cost (CoDA-unbounded achieves 62–67% of baseline throughput). This is irreducible without a fused Triton kernel that computes both attention maps in a single pass.

Fine-tuning required. CoDA-GQA-L is not a zero-shot drop-in replacement. The differential mechanism and HeadwiseRMSNorm fundamentally reshape the activation manifold (cold-swap PPL: 3,371,482), requiring two-phase fine-tuning.

Two-phase training cost. Phase 2 adds $0.5\text{--}1\times$ the GPU hours of Phase 1, with $\sim 20\times$ lower throughput due to per-block bank updates with gradient flow.

Bank update overhead. Eager PyTorch with ~ 20 micro-kernel launches per bank update is the throughput bottleneck, yielding 11–15% of baseline prefill speed. Fixed-shape buffering with `torch.compile` or custom CUDA kernels would reduce this substantially.

PPL gap. Bounded perplexity remains 41.5% above unbounded at 135M scale. This represents the information-theoretic cost of $5.3\times$ context compression and may improve at larger model scales where redundancy is higher.

No distributed sharding. The current implementation lacks tensor parallel and sequence parallel support, limiting deployment to single-GPU settings.

No quantized KV storage. Memory banks store in full dtype (bf16). Combining with KV quantization [7] could yield additional $4\text{--}8\times$ compression.

8 Conclusion

We have presented CoDA-GQA-L, a bounded-memory attention architecture that combines constrained orthogonal differential attention, dual-bank memory with value-routed semantic matching, and a two-phase training protocol. Benchmarks across three model scales demonstrate that bounded throughput is scale-invariant at $\sim 150\text{K}$ tokens/second, memory compression reaches $37\times$ per layer at 7B/70B scale, and the constant-size state achieves over $1,100\times$ compression at 128K context—enabling deployment scenarios where standard KV caches require multi-GPU clusters.

The two-phase training protocol proves essential: direct bounded evaluation after unbounded training produces catastrophic quality loss, while Phase 2 adaptation recovers bounded perplexity to within 41.5% of unbounded quality under $5.3\times$ context compression. The serializable bounded state opens a new paradigm of Stateful Neural Databases, where compressed context artifacts enable save/load/query semantics for production RAG systems.

Future work includes fused Triton kernels for differential attention (eliminating the $2\times$ SDPA overhead), KV quantization of memory banks, distributed sharding support, and scaling the two-phase protocol to full 7B and 70B model fine-tuning—where we expect the PPL gap to narrow as larger models encode more redundancy compressible by the dual-bank system.

Acknowledgments

This work was conducted using NVIDIA H200 NVL GPUs provided by RunPod. SmolLM2-135M was developed by the HuggingFace team. We thank the FlashAttention [4, 16] developers for the efficient attention infrastructure that this work builds upon.

References

- [1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL <https://arxiv.org/abs/2305.13245>.
- [2] Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. SmolLM2: When smol goes big — data is all you need. *arXiv preprint arXiv:2502.02737*, 2025.
- [3] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019. URL <https://arxiv.org/abs/1901.02860>.
- [4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL <https://openreview.net/forum?id=H4DqfPSibmx>.
- [5] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *Proceedings of the International Conference on Learning Representations*, 2024. URL <https://arxiv.org/abs/2310.01801>.

-
- [6] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023.
 - [7] Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. GEAR: An efficient KV cache compression recipe for near-lossless generative inference of LLM. In *Advances in Neural Information Processing Systems*, volume 37, 2024.
 - [8] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, pages 611–626, 2023. doi: 10.1145/3600006.3613165.
 - [9] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638.
 - [10] Zichang Liu, Aashiq Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhao Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. *arXiv preprint arXiv:2305.17118*, 2023.
 - [11] Amirkeivan Mohtashami and Martin Jaggi. Landmark attention: Random-access infinite context length for transformers. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL <https://arxiv.org/abs/2305.16300>.
 - [12] Tsendsuren Munkhdalai, Manaal Faber, Michael Goesele, Yonatan Bisk, and Navdeep Jaitly. Leave no context behind: Efficient infinite context transformers with Infini-attention. *arXiv preprint arXiv:2404.07143*, 2024.
 - [13] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shao. YaRN: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.
 - [14] Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. Controlling text-to-image diffusion by orthogonal finetuning. In *Advances in Neural Information Processing Systems*, volume 36, 2024.
 - [15] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2020.
 - [16] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/7ede97c3e082c6df10a8d6103a2eebd2-Paper-Conference.pdf.
 - [17] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.
 - [18] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/j.neucom.2023.127063. URL <https://arxiv.org/abs/2104.09864>.

- [19] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [20] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>.
- [21] Xinghao Wang, Pengyu Wang, Xiaoran Liu, Fangxu Liu, Jason Chu, Kai Song, and Xipeng Qiu. Prism: Spectral-aware block-sparse attention. *arXiv preprint arXiv:2602.08426*, 2026. doi: 10.48550/arXiv.2602.08426. URL <https://arxiv.org/abs/2602.08426>.
- [22] Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *Proceedings of the International Conference on Learning Representations*, 2022.
- [23] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *Proceedings of the International Conference on Learning Representations*, 2024. URL <https://arxiv.org/abs/2309.17453>. arXiv preprint arXiv:2309.17453.
- [24] Tianzhu Ye, Li Dong, Yuqing Xia, Yutao Sun, Yi Zhu, Gao Huang, and Furu Wei. Differential transformer. In *Proceedings of the International Conference on Learning Representations*, 2025. URL <https://arxiv.org/abs/2410.05258>. arXiv preprint arXiv:2410.05258, October 2024.
- [25] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.
- [26] 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. H₂O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2024. URL <https://arxiv.org/abs/2306.14048>.

A Implementation Details

The CoDA-GQA-L codebase follows a mixin-based architecture separating attention logic (`attention.py`) from memory bank operations (`memory_banks.py`), with state management in a dedicated dataclass (`state.py`). Key implementation choices include:

- **Vectorized bank updates:** All memory bank operations use `scatter_reduce` and `scatter_add` with winner-take-all selection, eliminating Python loops over tokens.
- **Winner-take-all in float32:** bf16 cosine similarity ties are broken by computing routing in float32, preventing non-deterministic slot assignment.
- **Scratch buffer management:** When `detach_evicted=False`, fresh scratch buffers are allocated per chunk to avoid autograd version conflicts. With `detach_evicted=True` (inference default), buffers are reused for efficiency.
- **Drop-in adapters:** `LlamaCoDAAdapter` replaces Llama-family attention layers with automatic weight mapping (including RoPE convention conversion between contiguous-half and interleaved formats). `EveCoDAAdapter` provides equivalent support for Eve-2 models.

The complete test suite (60 tests) validates correctness across shapes, dtypes, and edge configurations; determinism under seeded execution; invariant maintenance for causal masking, GQA alignment, ring buffer wrapping, and LRU ordering; and autograd safety for the `detach_evicted=False` training path.

B Engineering Challenges

Development of the gradient-through-banks training path (`detach_evicted=False`) required resolving three interconnected autograd issues:

RoPE position overflow. The evaluation function accumulated position counters across chunks, reaching positions far beyond the model’s trained context length (50K+ vs. 8,192 max). This produced garbage RoPE embeddings and spuriously high perplexity (2,624). Fix: reset adapter state before each evaluation chunk.

In-place mutation version conflict. With `detach_evicted=False`, three layers of in-place mutation caused `ScatterAddBackward0` crashes: (1) shared scratch buffers incremented tensor versions across chunks, (2) state buffer writes after SDPA saved references to the same tensor, and (3) bank view reads and writes operated on the same base tensor. Fix: three-layer clone defense—fresh scratch buffers per chunk, clone state buffers after ring reads, and clone bank views after slicing.

Gradient checkpointing conflict. Gradient checkpointing’s forward recomputation is incompatible with in-place graph-connected writes. Fix: auto-disable gradient checkpointing when `detach_evicted=False`.

C Reproducibility

All code, configurations, and benchmark scripts are open-sourced. Key entry points:

- `benchmarks/train_coda.py`: Full two-phase training pipeline
- `benchmarks/run_suite.py`: Performance benchmark suite (5 configurations, JSON output)
- `tests/`: 60 tests executable via `pytest`

Installation: `pip install coda-gqa-l`. Hardware requirements: Phase 1 training fits in 3 GB VRAM; Phase 2 requires ~ 8 GB. Inference benchmarks require GPU memory proportional to model scale.

D Training Walkthrough

This appendix provides a practical guide to reproducing the two-phase training protocol. All code is available via `pip install coda-gqa-l` and the `benchmarks/train_coda.py` script.

Step 1: Install and Verify Environment

```
pip install git+https://github.com/anthony-maio/CoDA-GQA-L.git
pip install transformers datasets

import torch
from coda_gqa_l import CoDAGQALandmarkPerf2, LlamaCoDAAdapter
print(f"GPU: {torch.cuda.get_device_name(0)}")
print(f"VRAM: {torch.cuda.get_device_properties(0).total_mem/1e9:.0f} GB")
```

Step 2: Swap Attention Layers

The adapter replaces each Llama-family attention layer with CoDA-GQA. Default initialization is near-identity: `theta_init=0` (noise query = signal query), `lambda bias = -6` ($\lambda \approx 0.0025$), and `head_norm_mode="identity"` (preserves pre-trained output scale). This ensures the model starts close to its pre-trained perplexity and the differential mechanism “wakes up” gradually during training.

```
from transformers import AutoModelForCausalLM, AutoTokenizer

model = AutoModelForCausalLM.from_pretrained(
    "HuggingFaceTB/SmolLM2-135M", torch_dtype=torch.bfloat16
).cuda()
tokenizer = AutoTokenizer.from_pretrained("HuggingFaceTB/SmolLM2-135M")

# Swap all attention layers to unbounded CoDA
for block in model.model.layers:
    adapter = LlamaCoDAAdapter.from_llama_attention(
        block.self_attn,
        bounded=False,                # Phase 1: full KV context
        head_norm_mode="identity",    # preserve pre-trained scale
        theta_init=0.0,               # noise = signal initially
    ).to("cuda", dtype=torch.bfloat16)
    block.self_attn = adapter
```

Step 3: Phase 1 — Unbounded Training

Phase 1 trains the differential attention parameters (θ , λ , head norm) and optionally the full Q/K/V/O projections with full KV context. A dual learning rate schedule uses a higher rate (10^{-3}) for the new CoDA parameters and a lower rate (5×10^{-5}) for the existing projections:

```
python -m benchmarks.train_coda \
    --model HuggingFaceTB/SmolLM2-135M \
    --max-steps 2000 \
    --freeze attention \
    --lr 5e-5 --lr-coda 1e-3 \
    --eval-every 200 \
    --output-dir runs/smollm2_phase1
```

Expected output: perplexity drops from ~ 70 (initial) to ~ 22 over 2,000 steps on WikiText-103, validating that the CoDA differential mechanism has successfully separated signal from noise.

Step 4: Phase 2 — Bounded Training

Phase 2 is the critical step. The trained unbounded weights are copied into bounded adapters with a fixed-size KV buffer. Bounded-only parameters (write gate `write_proj`, EMA blending rate `summary_eta_logit`) start at their defaults and are trained alongside the existing weights at a reduced learning rate:

```
python -m benchmarks.train_coda \
    --model HuggingFaceTB/SmolLM2-135M \
```



```
--max-steps 2000 --bounded-steps 2000 \
--bounded-config medium \
--no-detach-evicted \
--eval-every 200 \
--output-dir runs/smollm2_twophase
```

The `-bounded-config medium` flag sets $W=256$, $M_e=64$, $M_s=64$ (384 total slots). The `-no-detach-evicted` flag allows gradients to flow through memory bank updates, enabling the write gate and EMA blending parameters to receive training signal from the language modeling loss.

Bounded configurations. Table 9 summarizes the available presets:

Table 9: Bounded KV cache presets. Total slots = $W + M_e + M_s$.

Config	Window	Exact	Summary	Total
tiny	128	32	32	192
medium	256	64	64	384
large	512	128	128	768

Step 5: Loading Trained Adapters for Inference

After training, adapter weights are saved in the output directory. To load them for inference:

```
import torch
from transformers import AutoModelForCausalLM
from coda_gqa_l import LlamaCoDAAdapter
from pathlib import Path

model = AutoModelForCausalLM.from_pretrained(
    "HuggingFaceTB/SmolLM2-135M", torch_dtype=torch.bfloat16
).cuda()

# Swap to bounded CoDA attention
for block in model.model.layers:
    adapter = LlamaCoDAAdapter.from_llama_attention(
        block.self_attn,
        bounded=True,
        window=256, num_landmarks_exact=64,
        num_landmarks_summary=64,
    ).to("cuda", dtype=torch.bfloat16)
    block.self_attn = adapter

# Load trained weights
ckpt = Path("runs/smollm2_twophase/phase2/final")
state = torch.load(ckpt / "coda_adapters.pt", map_location="cpu")
for i, block in enumerate(model.model.layers):
    block.self_attn.load_state_dict(state[f"layer_{i}"], strict=False)
```

```
# Cache size per layer (constant regardless of context length)
print(f"KV state per layer: "
      f"{model.model.layers[0].self_attn.cache_bytes(1, torch.bfloat16):,} B")
```

Step 6: Scaling to Larger Models

The same script handles Mistral 7B and Llama 3.1 8B with appropriate batch size adjustments:

```
# Mistral 7B on H100 (~6 hours)
python -m benchmarks.train_coda \
    --model mistralai/Mistral-7B-v0.3 \
    --max-steps 2000 --bounded-steps 1000 \
    --bounded-config medium \
    --batch-size 2 --grad-accum 4

# Llama 3.1 8B on H100 (~8 hours)
python -m benchmarks.train_coda \
    --model meta-llama/Llama-3.1-8B \
    --max-steps 3000 --bounded-steps 2000 \
    --bounded-config large \
    --batch-size 1 --grad-accum 8
```

Key Training Considerations

- **Phase 1 is essential.** Without it, the differential attention parameters (θ , λ) remain at their near-identity initialization and the model cannot effectively use the noise-cancellation mechanism in Phase 2.
- **Dual learning rates.** The new CoDA parameters (θ , λ , head norm) need a $20\times$ higher learning rate than the pre-trained projections to converge within a practical step budget.
- **State reset per batch.** During bounded training, adapter state is reset before each forward pass so each sequence starts with an empty KV buffer, matching the assumption that bounded inference processes each document independently.
- **Gradient checkpointing.** Compatible with Phase 1 and Phase 2 when `detach_evicted=True` (default). Must be disabled when `detach_evicted=False` due to in-place mutation conflicts (see Appendix B).
- **Cold-swap baseline.** Evaluating a Phase-1-only model with bounded attention (skipping Phase 2) yields catastrophic PPL degradation ($438\times$ on Mistral 7B, $\sim 150,000\times$ on SmolLM2-135M), confirming that Phase 2 adaptation is essential for bounded inference.