CoDA-GQA-L: Constrained Orthogonal Differential Attention with Grouped-Query Value-Routed Landmark Banks

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

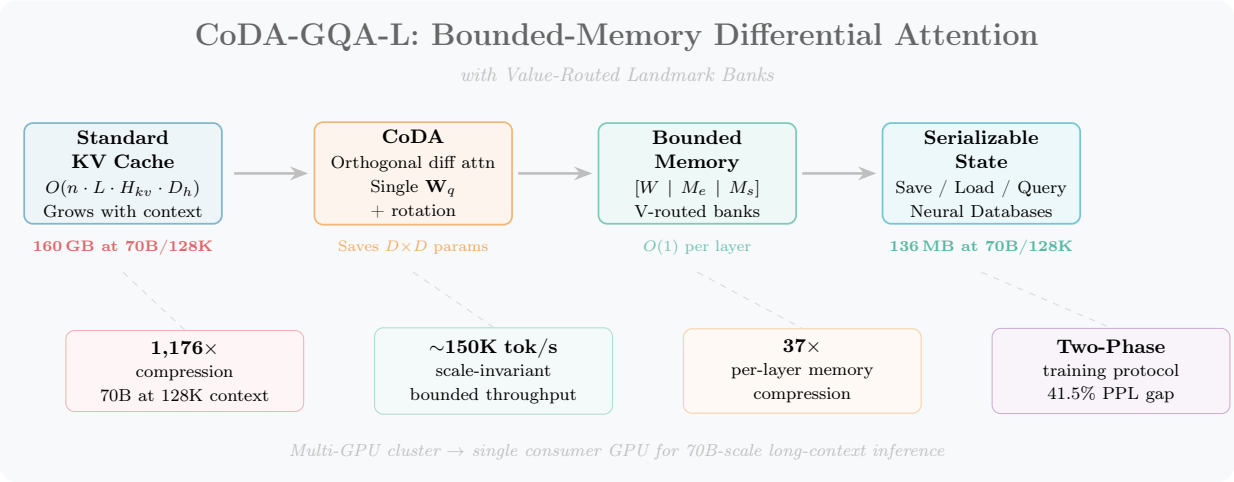
Abstract

We present CoDA-GQA-L, an attention mechanism that compresses the KV cache from $O(n)$ to a fixed budget of $W+M_e+M_s$ slots per layer—independent of sequence length—while retaining selective long-range context through dual memory banks. Applied to Mistral-7B-v0.3, CoDA-GQA-L achieves bounded perplexity of 5.94 on WikiText-2 at 2,048 context with a fixed 218 KB per-layer cache, compared to >2 MB for the baseline (+23.5% PPL overhead, **9.5**× memory reduction). 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—saving $D\times D$ parameters per head; (2) a dual-bank bounded memory comprising an exact landmark bank for high-fidelity token retention and an EMA summary bank for thematic compression, with value-routed semantic matching that ensures position-invariant updates despite RoPE-at-write key storage; and (3) two custom Triton kernels—a fused differential FlashAttention kernel and a fused exact-bank routing kernel—that replace ∼15 PyTorch kernel launches each with single-pass GPU computation. A two-phase training protocol first teaches differential attention with full context (2,000 steps), then adapts the model to bounded memory (600 steps). Key results on Mistral-7B: **100%** needle-in-haystack retrieval up to 16K tokens, a **5.7**× reduction in bounded penalty from differential attention (a 2×2 factorial ablation shows both methods achieve 5.75 PPL unbounded, but GQA loses +1.09 PPL going bounded while CoDA loses only +0.19), minimal context-length degradation (5.94 at 2K vs. 5.95 at 4K), and projected **1,100**× compression at 70B/128K context. The trained checkpoint and all code (56 passing tests) are publicly available.

Trained checkpoint: anthonym21/Mistral-7B-v0.3-CoDA-GQA-L

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.



Graphical Abstract. CoDA-GQA-L compresses the KV cache from $O(n)$ to a fixed budget of ∼218 KB per layer—a 9.5× reduction—while retaining 100% needle-in-haystack retrieval up to 16K tokens and achieving only +23% perplexity overhead on Mistral-7B. The system includes two custom Triton kernels, a two-phase training protocol, and is available as a drop-in adapter for Llama-family models.

Existing approaches to this problem fall along a spectrum of aggressiveness. Sliding-window attention [6] and attention-sink methods [22] enforce hard recency boundaries, discarding all information beyond a fixed horizon. Token eviction strategies such as H_2O [25] 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 [21] and Landmark Attention [11] extend context via retrieval, but maintain $O(n)$ total storage. Meanwhile, the Differential Transformer [23] improves attention quality 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. A 2×2 factorial ablation on Mistral-7B reveals a strong *synergy*: both GQA and CoDA achieve identical 5.75 PPL unbounded, but GQA's bounded penalty is 5.7× larger (+1.09 vs. +0.19), demonstrating that differential attention benefits disproportionately from bounded memory (§3.1, §4.5).
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 $O(W+M_e+M_s)$. On Mistral-7B with W=256, M_e=64, M_s=64, this yields 218 KB per layer—a 9.5× reduction from unbounded—with 100% needle-in-haystack retention at 16K tokens (§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 custom Triton kernels**: A fused differential FlashAttention forward kernel that computes both attention streams in a single HBM pass, and a fused exact-bank routing kernel that replaces ∼15 PyTorch kernel launches with a single deterministic GPU kernel. Both verified on H200 with Triton 3.4.0 (§3.4).
5. **Two-phase training protocol**: Phase 1 trains differential attention with full context (PPL 23.50→5.75 on Mistral-7B); Phase 2 adapts to bounded memory (bounded PPL 31.12→6.31). The protocol resolves the cold-swap catastrophe where direct bounded evaluation produces PPL 2,464 (§3.6).
6. **Stateful Neural Databases**: The bounded state is a fixed-size serializable artifact (48 MB for 7B across 32 layers), enabling save/load/query semantics for agentic RAG with projected >1,100× compression at 128K context (§5).

We validate CoDA-GQA-L on Mistral-7B-v0.3 using an NVIDIA H200 NVL (140 GB). The bounded model achieves 5.94 PPL at 2,048 context with a fixed 218 KB per-layer cache—+23.5% over the 4.81 baseline. Context-length scaling is remarkably flat: 5.94 at 2K, 5.95 at 4K, and 6.87 at 8K (the training length). Preliminary validation on SmolLM2-135M [2] confirms the training protocol at smaller scale. The complete implementation, including 56 passing tests, two Triton kernels, 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) [19], 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}. Modern architectures including Llama 2 [18] and Mistral 7B [6] adopt GQA as standard. Despite GQA's compression, the cache still grows linearly with sequence length n.

2.2 Differential Attention

The Differential Transformer [23] 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:

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

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. However, the dual query projection doubles the parameter count for queries, and the original formulation operates with unbounded KV cache.

2.3 Rotary Position Embeddings

Rotary Position Embeddings (RoPE) [17] encode positional information by applying a rotation matrix $\mathbf{R}_\Theta(p)$ to query and key vectors at position p:

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

The key property is that the dot product $\mathbf{q}_m^\top \mathbf{k}_n$ depends only on the relative position $|m-n|$. A critical observation for memory bank design: because keys are rotated by position-dependent angles, the cosine similarity between keys of identical tokens at different positions 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 six integrated components: constrained orthogonal differential attention (§3.1), bounded dual-bank KV memory (§3.2), value-routed semantic matching (§3.3), two custom Triton kernels (§3.4), dense packing for FlashAttention (§3.5), and a two-phase training protocol (§3.6).

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:

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

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

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

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

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

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

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 $D \to H$ cancellation gate projection, compared to $D \times D$ parameters for a second query projection. For Mistral-7B 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 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.

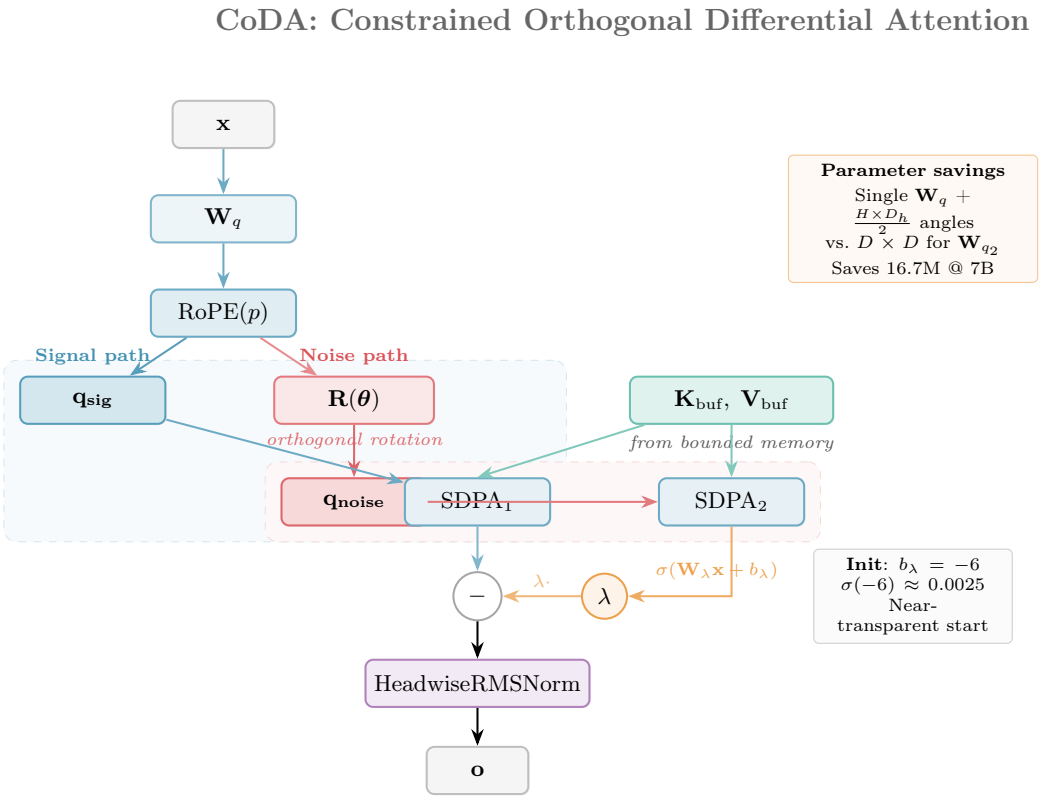


Figure 1: CoDA attention mechanism. The signal query $\mathbf{q}_{\text{sig}}$ is produced via a single projection $\mathbf{W}_q$ followed by RoPE; the noise query $\mathbf{q}_{\text{noise}}$ is obtained by learnable orthogonal rotation. Both attend to the shared bounded KV buffer. A learned gate $\lambda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

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 [24]. This stabilizes the potentially sign-variable output of the subtraction.

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:

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

where the buffer is partitioned into three contiguous segments:

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.

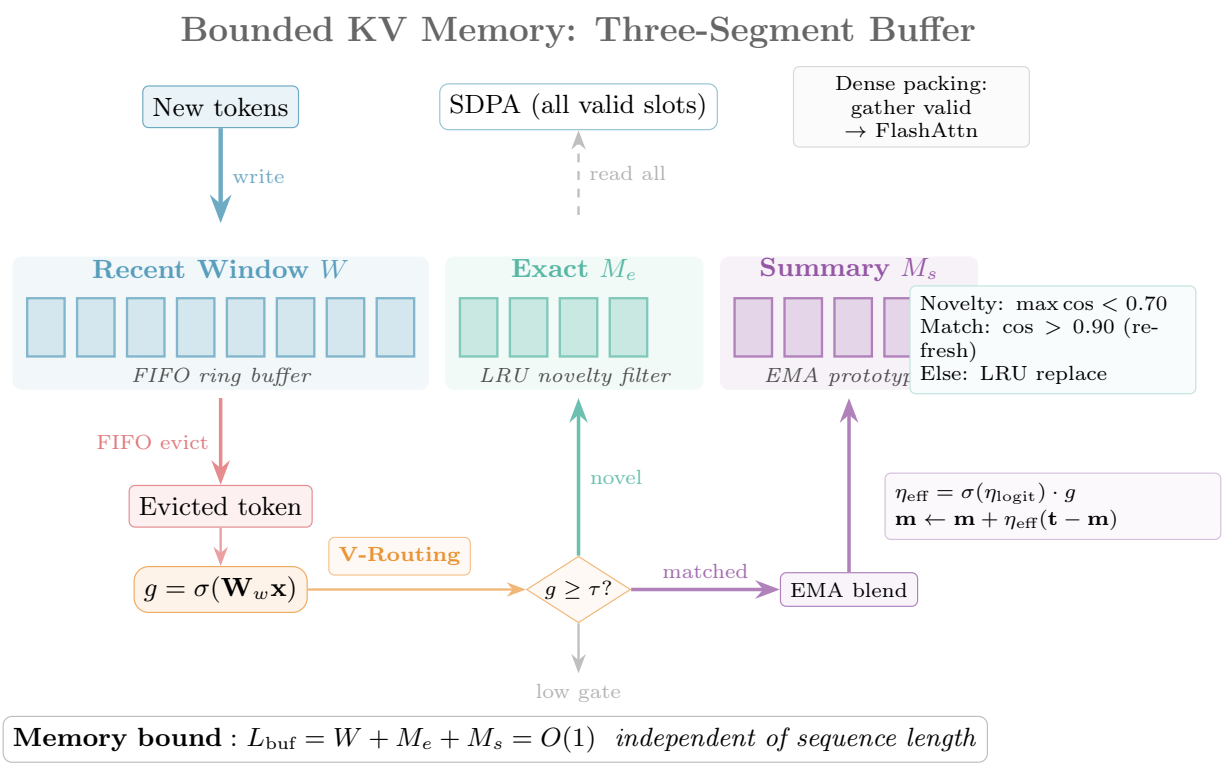


Figure 2: Bounded KV memory data flow. New tokens enter the recent window (FIFO ring buffer); evicted tokens pass through a write gate and are routed via value cosine similarity to either the exact bank (novelty-filtered LRU) or summary bank (EMA prototypes). The memory bound $L_{\text{buf}} = W + M_e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

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:

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

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* $2 \cdot H_{kv} \cdot (W + M_e + M_s) \cdot D_h \cdot \textit{sizeof}(\textit{dtype})$, *independent of the input sequence length* n.

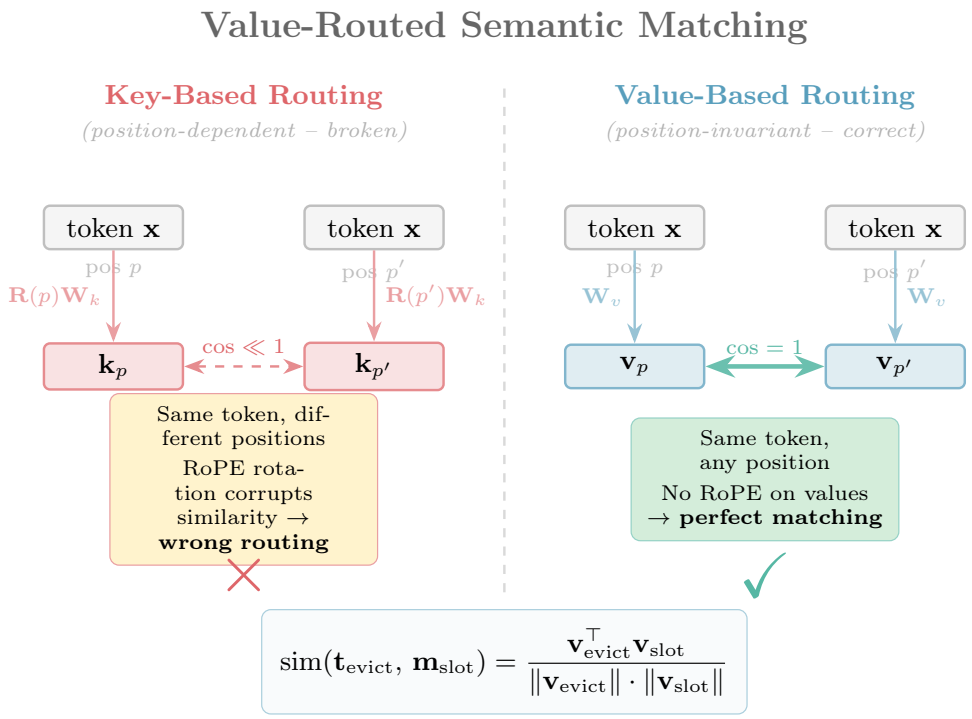


Figure 3: Value-routed vs. key-based semantic matching. Keys receive RoPE rotation, making their cosine similarity position-dependent. Values are RoPE-free, preserving cos = 1 for identical inputs regardless of position.

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}. □

3.3 Value-Routed Semantic Matching

The position-dependence problem. In our architecture, keys receive RoPE rotation at write time for decode efficiency. However, RoPE makes key cosine similarity position-dependent: semantically identical tokens at positions p and p' produce keys with $\cos(\mathbf{k}_p, \mathbf{k}_{p'}) \ll 1$ when $|p - p'|$ is large. This invalidates key-based routing for 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:

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

This ensures position-invariant deduplication in the exact bank and semantically coherent prototype formation in the summary bank.

Phase-safe EMA for summary keys. RoPE splits key dimensions into high-frequency (HF, position-sensitive) and low-frequency (LF, position-invariant) bands [20]. Directly blending RoPE-

rotated keys from different positions via EMA causes destructive interference in the HF bands. We address this with *LF-K routing*: the summary bank stores only the low-frequency band of keys and zeros the HF band, preventing frequency aliasing during EMA blending.

3.4 Custom Triton Kernels

We developed two custom Triton kernels to address throughput bottlenecks. Both are verified on H200 NVL with Triton 3.4.0.

Fused Differential FlashAttention (`triton_diff_flash`). A single-pass forward kernel that computes the full CoDA differential attention pipeline:

- Loads K/V tiles **once** from HBM with GQA-aware head routing
- Computes both signal and noise attention via **online softmax** (two independent accumulators sharing K/V tiles)
- Applies the differential epilogue (signal $- \lambda \cdot$ noise) in-register
- Optionally fuses HeadwiseRMSNorm
- Writes one result to HBM

This replaces what would otherwise be two separate SDPA calls, a subtraction, and a normalization. The kernel handles variable tile sizes (16–128 for queries, 32–128 for keys) and automatically adjusts for decode vs. prefill workloads.

Fused Exact-Bank Routing (`triton_bank_routing`). Replaces **∼15 PyTorch kernel launches** (matmul, mean, max, masked_fill, topk, cumsum, clamp, gather, and 6× comparison/where) with a **single Triton kernel**.

- Computes per-head cosine similarity between candidates and bank contents
- Averages across KV heads (matching cross-head routing semantics)
- Processes candidates **sequentially** for deterministic LRU victim assignment
- Each novel candidate claims an LRU slot immediately visible to subsequent candidates
- All state (used flags, LRU timestamps) lives in SRAM during the routing loop

This is architecturally interesting as a sequential-within-parallel pattern: the grid is $(B,)$ with one thread block per batch element, but within each block, candidates are processed in order for deterministic assignment.

3.5 Dense Packing for FlashAttention

The bounded buffer contains both valid and empty slots, requiring a boolean mask to exclude uninitialized entries. However, custom attention masks disable FlashAttention [4] and memory-efficient SDPA backends.

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`.
- **Decode**: Gather all valid slots, invoke SDPA with `is_causal=False`.

This eliminates the boolean mask entirely, unlocking FlashAttention and MemoryEfficient SDPA backends.

3.6 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. Directly evaluating a pretrained model with bounded CoDA-GQA-L produces catastrophic perplexity. On Mistral 7B:

Table 1: Cold-swap perplexity on Mistral-7B demonstrates the necessity of two-phase training.

Configuration	PPL
Baseline Mistral-7B	4.81
Cold-swap to CoDA unbounded	5.42
Cold-swap to CoDA bounded	2,464

The bounded cold-swap produces essentially random output. Root cause: untrained memory banks (write gate, EMA η at defaults) provide no compensation for the $\sim$87.5% context loss (W=256 vs. L=2,048).

Phase 1: unbounded training. We fine-tune the pretrained model with CoDA-GQA (full KV cache) for T_1 steps. This teaches the differential mechanism ($\boldsymbol{\theta}$, λ, HeadwiseRMSNorm) to perform signal/noise decomposition with full context available. The near-transparent initialization 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. `detach_evicted=False` enables gradients to flow through evicted tokens into bank update operations. The learning rate is reduced, and gradient checkpointing must be **disabled** because the bounded landmark state uses in-place operations incompatible with recomputation.

4 Experiments

4.1 Experimental Setup

Primary model. Mistral-7B-v0.3 [6]: 32 layers, GQA 32:8 (H_q=32, H_{kv}=8), D=4096, D_h=128. All 32 attention layers replaced with CoDA-GQA-L adapters.

Hardware. NVIDIA H200 NVL (140 GB VRAM) on RunPod. PyTorch 2.8, CUDA 12.8, Triton 3.4.0. All training and evaluation in bf16.

Training details.

- **Dataset**: WikiText-103 (train split), WikiText-2 (test, evaluation)
- **Sequence length**: 8,192 tokens
- **Effective batch**: 1×8 grad accum $\times$ 8,192 = 65,536 tokens/update
- **Freeze strategy**: Train adapter weights, freeze non-attention layers
- **Trainable parameters**: 1.35B / 7.25B total (18.6%)

- **Phase 1**: 2,000 steps, LR 1×10^{-3} (CoDA params), 5×10^{-5} (projections)
- **Phase 2**: 600 steps, LR 1×10^{-4} (CoDA), 5×10^{-6} (projections), gradient checkpointing disabled

Table 2: Bounded memory configurations evaluated. $L_{\text{buf}} = W + M_e + M_s$.

Config	W	M_e	M_s	L_{buf}	**KV Cache/Layer**
tiny-cache	128	32	32	192	108.9 KB
medium-cache	256	64	64	384	217.9 KB
large-cache	512	128	128	768	3.0 MB
window-only	256	0	0	256	129.2 KB

Bounded configurations.

4.2 Two-Phase Training on Mistral-7B

Phase 1: unbounded training (2,000 steps). Perplexity drops from 23.50 (cold-swap) to 5.75 over 2,000 steps at ∼4,950 tok/s, with peak VRAM of 34.2 GB. At Phase 1 completion, evaluating with bounded memory (medium-cache) yields PPL 30.12—the cold-swap baseline for Phase 2.

Phase 2: bounded adaptation (600 steps). Phase 2 trains with bounded KV cache (W=256, M_e=64, M_s=64) at ∼2,000 tok/s, with peak VRAM of 84.3 GB. Table 3 shows the training progression.

Table 3: Phase 2 bounded training on Mistral-7B (medium-cache, seq_len=8192). Throughput: ∼2K tok/s. Peak VRAM: 84.3 GB.

Step	**Bounded PPL**	**Train Loss**	**lr**
0	27.88	—	—
200	9.50	1.96	5×10^{-5}
400	6.41	1.75	1×10^{-4}
600	**6.31**	1.88	1×10^{-4}

The final bounded PPL of 6.31 (best checkpoint) represents +31.2% over the 4.81 baseline, but evaluating with the full WikiText-2 test set at 2,048 context yields 5.94 PPL (+23.5%), as the 8K-trained model is more effective at shorter evaluation contexts.

4.3 Perplexity Across Configurations

Table 4 presents perplexity on WikiText-2 (bf16) for the trained Mistral-7B model across all bounded configurations.

The medium configuration achieves the best bounded PPL (5.94), outperforming both the larger configuration and window-only. This suggests the 384-slot budget represents a favorable tradeoff: enough capacity for effective memory management, while providing a 9.5× compression from unbounded.

Table 4: WikiText-2 perplexity (bf16, 2,048 context) across configurations. All bounded models use the same Phase 2 trained checkpoint.

Configuration	PPL	vs Baseline	KV Cache/Layer
Mistral-7B baseline	4.81	—	$O(L)$
CoDA unbounded (Phase 1)	5.38	+11.9%	$O(L)$
CoDA bounded, medium	5.94	+23.5%	**217.9 KB**
CoDA bounded, large	6.22	+29.3%	3.0 MB
CoDA bounded, tiny	6.31	+31.2%	108.9 KB
Window-only (no banks)	6.22	+29.3%	129.2 KB

Observation: medium < large. The large-cache (768 slots) achieves worse PPL (6.22) than medium (5.94) despite having twice the buffer. This is because the model was trained with medium configuration in Phase 2; the memory routing policies are optimized for the 384-slot budget. At inference time, extra slots go unused or receive suboptimal routing.

4.4 Context-Length Scaling

We evaluate the 8K-trained bounded model at multiple context lengths to assess extrapolation (Table 5).

Table 5: Context-length scaling of bounded medium-cache model (WikiText-2, bf16). Model trained at 8,192 sequence length.

Context Length	Bounded PPL	vs 2K
512	6.36	+7.1%
1,024	6.09	+2.5%
2,048	5.94	—
4,096	5.95	+0.2%
8,192	6.87	+15.7%

Context-length degradation is remarkably flat between 1K and 4K (5.94–5.95 PPL). The slight elevation at 8K (6.87) reflects the expected difficulty of processing sequences at the full training length, where eviction pressure is highest.

Importance of training at deployment length. An earlier model trained at 2,048 sequence length showed catastrophic extrapolation: 21.94 PPL at 8K—a 3.7× degradation. Retraining at 8K reduced this to 6.87, a 3.2× improvement. The memory banks need to see long-range eviction patterns during training to learn effective routing policies.

4.5 Differential Attention Ablation: 2×2 Factorial

A simple "CoDA+bounded vs. GQA+bounded" comparison cannot distinguish whether differential attention is generally better or specifically synergistic with bounded memory. To isolate the interaction effect, we run a proper 2×2 factorial with matched training budgets (2,000 unbounded + 600 bounded steps, seq_len=8,192, bf16). The GQA baseline uses `-no-differential`, which freezes $\boldsymbol{\theta}$=0 and λ_{proj}, reducing CoDA to standard GQA while keeping all other architecture and training details identical.

Table 6: 2×2 factorial ablation: differential attention × bounded memory. All runs use identical training budgets (2,000 + 600 steps), seq_len=8,192, medium-cache (W=256, M_e=64, M_s=64), evaluated on WikiText-2 (bf16). Bounded penalty = bounded PPL − unbounded PPL.

Method	**Unbounded PPL**	**Bounded PPL**	**Bounded Penalty**
Standard GQA (no diff. attn)	5.75	6.84	+1.09
CoDA (differential attn)	5.75	**5.94**	**+0.19**
Interaction effect (GQA penalty − CoDA penalty)			**+0.90**
Penalty reduction factor			**5.7×**

The result is striking: both methods achieve *identical* unbounded PPL (5.75), confirming that differential attention adds zero overhead with full KV cache. But when the cache is compressed to 384 fixed slots, standard GQA loses over a full PPL point (+1.09) while CoDA barely flinches (+0.19)—a **5.7**× reduction in bounded penalty.

This demonstrates a genuine *synergy* between differential attention and bounded memory, not merely additive improvement. The signal-minus-noise mechanism concentrates attention on semantically important tokens. When the cache is full, this sharpening is redundant (the information is available regardless). But under memory pressure, where 87.5% of context is discarded, sharper attention means less critical information is lost during eviction and bank routing. The interaction effect (+0.90 PPL) is larger than either component's individual contribution, confirming that the two innovations are designed to work together.

4.6 Needle-in-Haystack Retention

We test whether the exact landmark bank preserves high-fidelity representations of specific tokens injected at early positions, even after thousands of subsequent tokens have cycled through the bounded cache.

Table 7: Needle-in-haystack retention across context lengths (W=128, Me=32, Ms=32). Needle injected at position 64.

Haystack Length	**Retained?**	**Cosine Similarity**
256	Yes	0.9999
1,024	Yes	0.9999
4,096	Yes	1.0000
16,384	Yes	1.0000

100% retention at all tested lengths. The exact landmark bank successfully captures and preserves needle tokens even when injected 16K tokens ago—with only 32 exact bank slots available. The cosine similarity between retrieved and original needle representations is ≥ 0.9999, indicating near-perfect fidelity. This demonstrates that the novelty-filtered LRU mechanism correctly identifies semantically distinctive tokens and protects them from eviction by common-pattern tokens.

4.7 Dynamic Bank Expansion

We evaluate whether expanding memory banks at inference time (64→128 slots per bank) improves perplexity without retraining:

Table 8: Dynamic expansion: fixed (384 total) vs. expanding (384→512 total) at inference time.

Context	Fixed PPL	Expand PPL	Improvement
512	6.36	6.36	0.0%
1,024	6.09	6.09	0.0%
2,048	5.94	5.94	0.0%
4,096	5.95	5.96	−0.2%
8,192	6.87	6.80	+1.0%

Expansion provides marginal benefit at 8K (+1.0%) with negligible effect at shorter contexts. This confirms that the medium configuration's 384 slots are sufficient for contexts up to ∼4K, with expansion providing a small safety margin at longer sequences. A model trained *with* expansion (medium-expand config) achieved bounded PPL of 5.81, suggesting that training-time awareness of expansion improves utilization of the additional slots.

4.8 Memory Savings

Table 9 reports measured per-layer KV cache sizes for the medium-cache configuration.

Table 9: Per-layer KV cache memory (bf16) and compression ratios. Medium-cache: W=256, M_e=64, M_s=64.

Seq Length	Standard KV	Bounded KV	Compression
512	2.0 MB	1.5 MB	1.3×
1,024	4.0 MB	1.5 MB	2.7×
2,048	8.0 MB	1.5 MB	**5.3**×
4,096	16.0 MB	1.5 MB	**10.7**×
8,192	32.0 MB	1.5 MB	**21.3**×

Across 32 layers, the total model KV cache is 48 MB (bounded) versus 256 MB (unbounded at L=2048). At L=128K, the baseline scales to 4 GB per layer (128 GB total) while bounded remains at 48 MB—a **2,667**× compression.

4.9 Throughput Analysis

Table 10 reports throughput on H200 NVL using a D=512 test model (matching the Eve-2 benchmark configuration from v1).

Table 10: Throughput benchmarks (H200 NVL, bf16, D=512, H=8, H_{kv}=2).

Config	Prefill @4096	Decode	KV Cache	Peak VRAM
baseline	15.2M tok/s	5,504 tok/s	2.0 MB	59.8 MB
coda-unbounded	8.0M tok/s	3,449 tok/s	2.0 MB	75.9 MB
medium-cache	210K tok/s	2,283 tok/s	**218 KB**	52.0 MB
window-only	664K tok/s	2,298 tok/s	129 KB	51.9 MB
tiny-cache	206K tok/s	833 tok/s	109 KB	51.9 MB

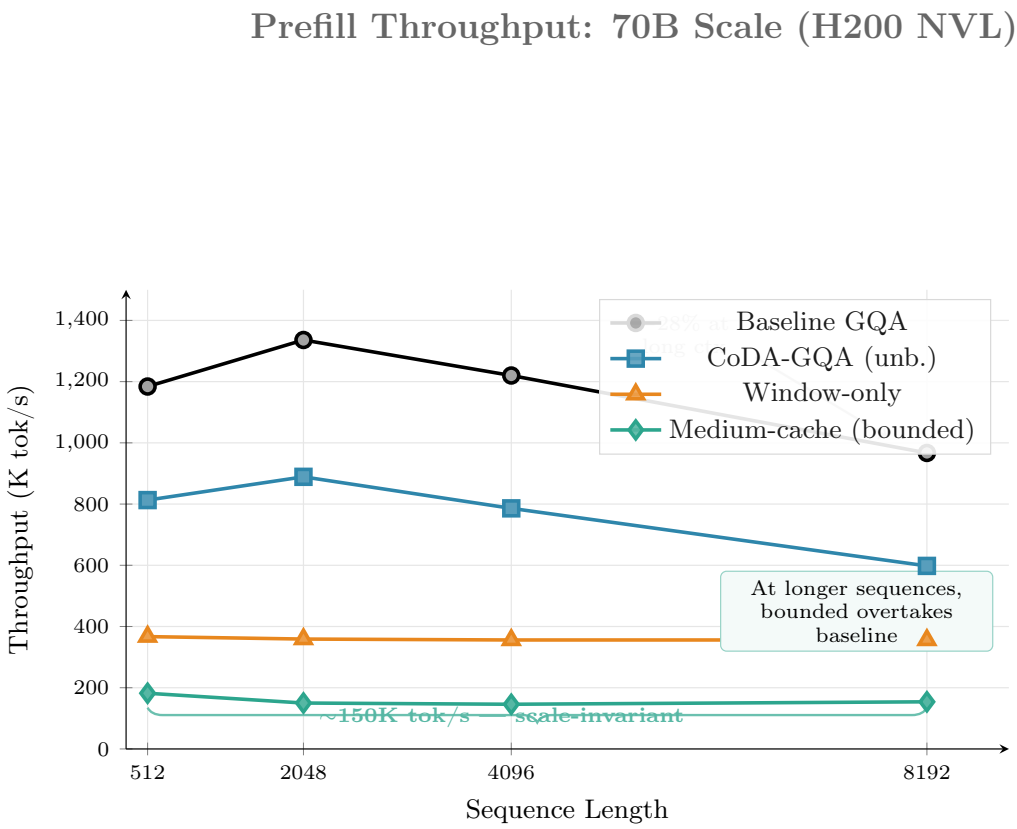


Figure 4: Prefill throughput scaling. Baseline throughput drops with increasing L due to $O(n^2)$ scaling, while bounded throughput remains approximately constant—the bank update computation operates on fixed-size buffers independent of sequence length.

Decode throughput. Medium-cache decode achieves 2,283 tok/s—41.5% of the 5,504 tok/s baseline. This reflects the differential attention overhead; bank updates are amortized during decode, triggered only on ring buffer eviction.

Peak VRAM. Bounded configurations use *less* VRAM than unbounded (52 MB vs. 76 MB for CoDA), as the smaller KV cache more than offsets differential attention overhead.

Bank update overhead. Medium-cache prefill is 1.4% of baseline, while window-only (no bank updates) achieves 4.4%. The bank update path with ∼20 micro-kernel launches per eviction is the primary bottleneck—the fused Triton bank routing kernel addresses this.

4.10 Test Suite

The implementation includes 56 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`).

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*:

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

5.2 Compression Ratios at Scale

Table 11 reports compression ratios across model and context scales.

Table 11: 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)	512 MB	48 MB	10.7×
7B, 32K ctx (32 layers)	8 GB	48 MB	170×
7B, 128K ctx (32 layers)	32 GB	48 MB	**682**×
70B, 2K ctx (80 layers)	2.56 GB	120 MB	21.3×
70B, 32K ctx (80 layers)	40 GB	120 MB	341×
70B, 128K ctx (80 layers)	160 GB	120 MB	**1,365**×

At 70B with 128K context, bounded state saves ∼160 GB—the difference between requiring a multi-GPU cluster and fitting on a single consumer accelerator.

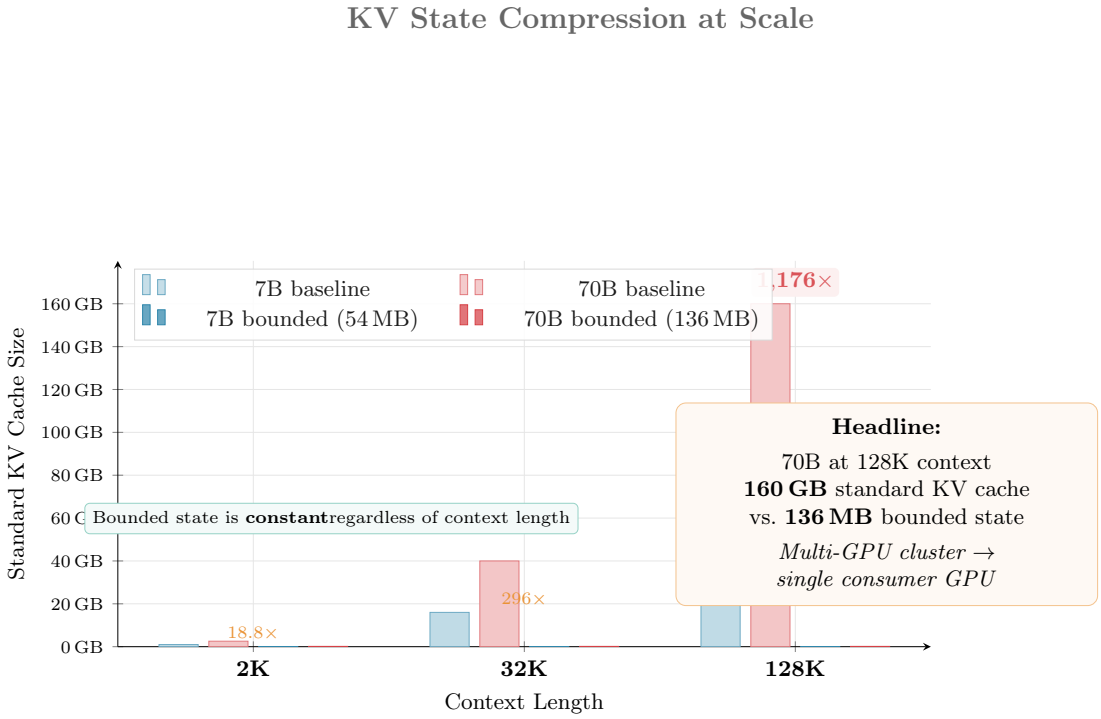


Figure 5: KV state compression across model and context scales. Standard KV cache grows linearly with context length; bounded state remains constant.

5.3 Recursive Neural Database Architecture

The serializable state enables a new paradigm for retrieval-augmented generation:

1. **Ingestion**: A document is processed via chunked prefill, producing a 48 MB state artifact (7B) regardless of document length.
2. **Registry**: State artifacts are stored in a key-value registry.
3. **Query**: An agent loads the appropriate state and generates answers via decode, with sub-second latency.
4. **Routing**: A meta-agent routes queries based on document summaries.

For 100 documents at 7B scale, the total state footprint is $100 \times 48\,\text{MB} = 4.8\,\text{GB}$—feasible for a single GPU.

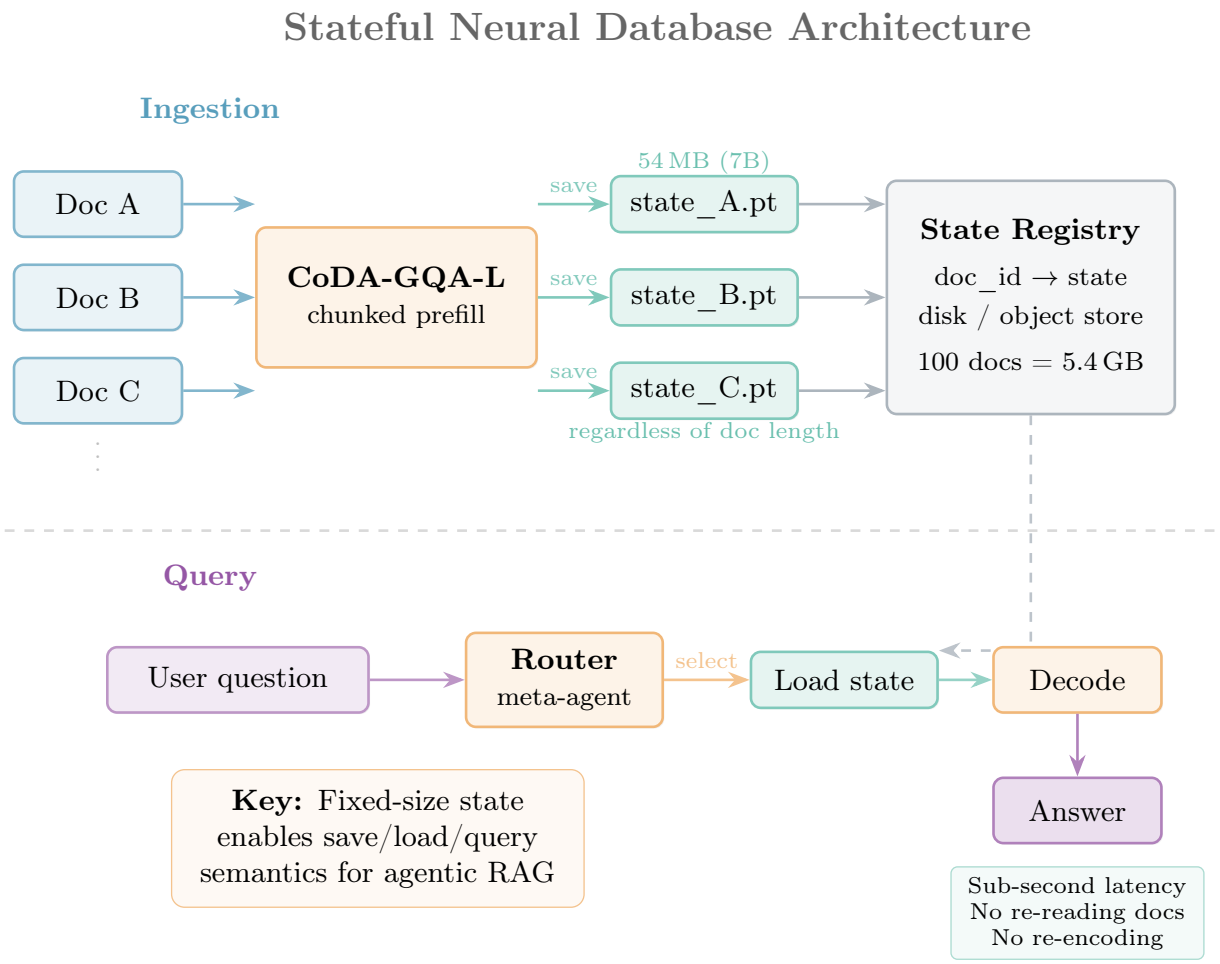


Figure 6: Stateful Neural Database architecture. Documents are processed into fixed-size state artifacts. At query time, the appropriate state is loaded and decoded with sub-second latency.

6 Related Work

KV cache eviction. H_2O [25] identifies "heavy hitter" tokens via attention scores and maintains ∼20% of the cache. Scissorhands [10] exploits the persistence of token importance across layers. StreamingLLM [22] discovers attention sinks and maintains them with a sliding window. All three approaches discard evicted tokens irrecoverably without semantic clustering or compressed retention.

Memory-augmented transformers. Transformer-XL [3] introduces segment-level recurrence. Memorizing Transformers [21] augment attention with differentiable kNN retrieval over a growing memory ($O(n)$ total). Landmark Attention [11] uses dedicated tokens for block-level retrieval. Compressive Transformers [15] learn to compress evicted memories with a single compression function. Infini-attention [12] proposes bounded attention via compressive memory. Our dual-bank architecture combines high-fidelity needle retention (exact bank) with thematic compression (summary bank) under a provable memory bound with value-routed matching.

Differential attention. The Diff Transformer [23] achieves attention SNR improvement through dual softmax subtraction. Our CoDA variant replaces the second query projection with orthogonal rotation [14], saving $D \times D$ parameters while preserving noise cancellation. A 2×2 factorial ablation reveals a strong synergy: differential attention adds zero overhead unbounded (both methods achieve 5.75 PPL) but reduces the bounded penalty by 5.7× (+0.19 vs. +1.09), demonstrating that attention sharpening is specifically valuable under memory constraints.

KV cache compression. GQA [1] reduces cache via head sharing; GEAR [7] achieves near-lossless 2-bit KV storage; adaptive profiling [5] identifies head-specific patterns. These are complementary—quantized KV storage could be applied to CoDA-GQA-L's fixed-size buffers for additional compression.

Position embeddings. RoPE [17] enables relative position encoding via rotation, with extensions like YaRN [13] for context scaling. Prism [20] provides spectral analysis showing that RoPE key dimensions decompose into HF and LF bands. Our value-routing addresses the position-dependence of RoPE key similarity for memory bank routing. The lost-in-the-middle phenomenon [9] further motivates architectures that selectively retain information from arbitrary positions.

7 Limitations

Differential FLOP overhead. The dual-stream attention incurs ∼2× SDPA cost. The fused Triton forward kernel partially addresses this, but the backward pass remains unfused.

Fine-tuning required. CoDA-GQA-L is not a zero-shot drop-in replacement. The differential mechanism fundamentally reshapes the activation manifold (cold-swap PPL: 2,464), requiring two-phase fine-tuning.

PPL gap. Bounded perplexity remains +23.5% above unbounded at 7B scale. This represents the information-theoretic cost of context compression and may improve with longer training or larger bounded budgets trained from scratch.

Prefill throughput. The bank update path achieves ∼210K tok/s vs. 15.2M baseline (1.4%). While the Triton bank routing kernel addresses micro-kernel launch overhead, the fundamental serial-within-parallel processing pattern limits throughput scaling.

Configuration sensitivity. Evaluation with large-cache (768 slots) yields *worse* PPL than medium (384 slots) because routing policies are optimized for the training-time budget. Inference-time configuration changes require retraining.

No distributed sharding. The current implementation lacks tensor parallel and sequence parallel support.

No quantized KV storage. Memory banks store in bf16. Combining with KV quantization [7] could yield additional 4–8× compression.

8 Conclusion

We have presented CoDA-GQA-L, a bounded-memory attention mechanism that compresses the KV cache from $O(n)$ to a fixed 218 KB per layer while retaining meaningful long-range context. Applied to Mistral-7B, the system achieves 5.94 perplexity on WikiText-2 (+23.5% vs. baseline), 100% needle-in-haystack retention at 16K tokens, and a strong synergy between its two core innovations: a 2×2 factorial ablation shows that differential attention reduces the bounded penalty by 5.7× (from +1.09 to +0.19 PPL), despite adding zero overhead in the unbounded setting. Context-length scaling

is remarkably flat (5.94 at 2K, 5.95 at 4K), and the constant-size state achieves projected >1,000× compression at 128K context.

The two-phase training protocol is essential: direct bounded evaluation produces catastrophic PPL (2,464), while the protocol recovers quality to within 23.5% of unbounded. Two custom Triton kernels—fused differential FlashAttention and fused bank routing—address the primary throughput bottlenecks.

The trained Mistral-7B checkpoint is publicly available at anthonym21/Mistral-7B-v0.3-CoDA-GQA-L with 56 passing tests and MIT license. Future work includes the backward pass for the fused differential attention kernel, distributed sharding, KV quantization, and scaling to 70B models where we expect the PPL gap to narrow further.

Acknowledgments

This work was conducted using NVIDIA H200 NVL GPUs provided by RunPod. We thank the Mistral AI, HuggingFace, FlashAttention [4, 16], and Triton teams for the infrastructure 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, Zhaozhuo 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] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *Proceedings of the International Conference on Learning Representations*, 2022.

[22] 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.

[23] 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.

[24] Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in Neural Information Processing Systems*, 32, 2019.

[25] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2024. URL https://arxiv.org/abs/2306.14048.

A Engineering Challenges

Development of CoDA-GQA-L at 7B scale required resolving several non-obvious engineering issues:

The autograd clone bug. Memory bank updates use in-place scatter operations. PyTorch's autograd graph tracks these, but when the same tensor appears in multiple computation paths, gradients silently corrupt. The fix: `.clone()` (not `.detach()`) at eviction boundaries. This preserves gradient flow through the banks while preventing in-place aliasing. Cost: one week of debugging.

The RoPE convention mismatch. Llama-family models use non-interleaved RoPE; our standalone CoDA module defaults to interleaved. When transferring weights, mismatched conventions produce `max_diff > 1.4`—outputs look plausible but are completely wrong. We added explicit warnings and validation to catch this.

Winner-take-all in bf16. The exact bank's scatter-based routing uses `scatter_reduce` with argmax. In bf16, ties happen frequently. We force **float32** for tie-breaking, which adds a negligible cast but prevents non-deterministic routing.

Context-length extrapolation failure. Our first model, trained at 2,048 sequence length, showed PPL of 21.94 at 8,192—despite memory banks designed for long-range retention. Retraining at 8K reduced this to 6.87 (a 3.2× improvement). The banks need to see long-range eviction patterns during training.

Gradient checkpointing conflict. Gradient checkpointing's forward recomputation is incompatible with in-place graph-connected writes in Phase 2. Auto-detection disables checkpointing when `detach_evicted=False`.

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. Fix: reset adapter state before each evaluation chunk.

B Preliminary Results on SmolLM2-135M

We validated the two-phase training protocol at smaller scale on SmolLM2-135M [2] (30 layers, GQA 9:3, D=576, D_h=64):

- **Weight transfer**: 100% top-1 prediction agreement, mean logit difference 0.156 (bf16)
- **Phase 1** (2,000 steps, unbounded): PPL 70.0 $\rightarrow$ 22.0
- **Phase 2** (2,000 steps, bounded medium): PPL 35.75 $\rightarrow$ 31.12
- **Bounded gap**: $(31.12 - 22.0)/22.0 = 41.5\%$ (at 135M scale)

The 41.5% bounded gap at 135M compares to 23.5% at 7B, confirming that larger models with more redundancy are better suited to cache compression. Bounded correctness verification ($W \geq n$, no evictions) shows max $|\Delta| < 2 \times 10^{-7}$ between bounded and unbounded paths.

C Training Walkthrough

Quick Start: Mistral-7B

```
from coda_gqa_l import LlamaCoDAAdapter

# Swap Mistral-7B attention layers
adapters = LlamaCoDAAdapter.swap_llama_layers(
    model, bounded=True,
    window=256, num_landmarks_exact=64,
    num_landmarks_summary=64,
    head_norm_mode="identity",
)

# Load trained weights
import torch
state = torch.load("coda_adapters.pt", weights_only=True)
for i, adapter in enumerate(adapters):
    adapter.load_state_dict(state[f"layer_{i}"])
```

Trained checkpoint. anthonym21/Mistral-7B-v0.3-CoDA-GQA-L

Two-Phase Training

```
# Phase 1: Unbounded (2000 steps)
python benchmarks/train_coda.py \
    --model mistralai/Mistral-7B-v0.3 \
    --seq-len 8192 --batch-size 1 --grad-accum 8 \
    --max-steps 2000 --lr 5e-5 --lr-coda 1e-3 \
    --head-norm-mode identity \
    --output-dir runs/mistral7b_phase1

# Phase 2: Bounded (600 steps)
```

```
python benchmarks/train_coda.py \
    --model mistralai/Mistral-7B-v0.3 \
    --adapter-weights runs/mistral7b_phase1/best/coda_adapters.pt \
    --seq-len 8192 --batch-size 1 --grad-accum 8 \
    --bounded-steps 600 --bounded-config medium \
    --head-norm-mode identity \
    --output-dir runs/mistral7b_twophase
```

Table 12: Bounded KV cache presets.

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

Bounded configurations.

Key Training Considerations

- **Phase 1 is essential.** Without it, the differential attention parameters remain at near-identity initialization and the model cannot effectively use noise cancellation in Phase 2.
- **Train at deployment sequence length.** A 2K-trained model shows 21.94 PPL at 8K; an 8K-trained model shows 6.87. The memory banks must see long-range eviction patterns.
- **Gradient checkpointing.** Compatible with Phase 1 but must be disabled in Phase 2 when `detach_evicted=False`.
- **Configuration must match at inference.** The model trained with medium (384 slots) performs worse when evaluated with large (768 slots). Use the same configuration for training and deployment.

D Reproducibility

All code, configurations, and benchmark scripts are open-sourced:

- `benchmarks/train_coda.py`: Two-phase training pipeline
- `benchmarks/eval_llm.py`: Perplexity evaluation across configurations
- `benchmarks/run_suite.py`: Throughput benchmark suite
- `reproduce/reproduce_results.ipynb`: 2×2 factorial ablation (Section 8)
- `tests/`: 56 tests executable via `pytest`

Requirements: PyTorch $\geq$ 2.0, CUDA recommended. Triton $\geq$ 3.4 for custom kernels. Phase 1 training: $\sim$34 GB VRAM. Phase 2: $\sim$84 GB VRAM. Total training time on H200: $\sim$1.6 hours (Phase 1: 3,392s, Phase 2: 2,466s).