

Decoupled Bottleneck Attention:

Low-Rank Semantic Routing as Structural Regularization

Daniel Owen van Dommelen

Independent Research

theapemachine@gmail.com

January 2026

Abstract

Standard Multi-Head Attention suffers from *attention drift*: high-dimensional heads overfit to spurious context correlations, causing repetition loops. We propose **Decoupled Bottleneck Attention (DBA)**, separating *semantic routing* from *positional geometry* and constraining the semantic path to low rank ($d_{\text{sem}} = 8$ per head). This bottleneck forces the model to discard distractors and focus on the immediate instruction—a **structural regularizer** against context hallucination.

We train 1B-parameter models on FineWeb-Edu and compare standard attention ($d = 64$ per head) against DBA variants. At 10k steps, baseline achieves loss 3.36 while DBA achieves 3.38—near-parity in convergence. DBA reduces training memory by 27% and maintains 32% higher throughput after warmup. Critically, behavioral probes reveal that the baseline model enters pathological repetition loops on in-context copy tasks (outputting “Red Green Blue...” when asked to copy “A7 B4 C9 D2”), while the aggressively compressed DBA variant **correctly recalls the target content**.

We argue that, for autoregressive language modeling, *high-rank semantic routing is often unnecessary and can be harmful*: excessive bandwidth can encourage spurious context matching, distracting the model from the current instruction. By bottlenecking only the semantic path while preserving positional geometry, DBA reframes efficient attention not merely as a memory technique, but as an architectural bias that can improve behavioral reliability.

Keywords: Transformer, attention mechanism, low-rank, attention drift, context hallucination, structural regularization, KV-cache, memory efficiency, robustness

1 Introduction

Modern Transformer architectures [21] achieve remarkable performance across language modeling, translation, and reasoning tasks. However, in autoregressive language modeling we argue that **high-rank semantic routing is often unnecessary and can be harmful**: allocating large bandwidth to token–token matching can encourage the model to attend broadly to context patterns rather than the immediate instruction.

When a 64-dimensional attention head looks back at context, it can see *everything* with high fidelity. This bandwidth can allow the model to attend to irrelevant tokens—few-shot examples, repeated patterns, distractor content—causing it to *overfit to context noise* rather than focus on the current input. We call this observed failure mode **attention drift**. Our experiments provide evidence consistent with a bandwidth/regularization mechanism, but we treat the causal story as a hypothesis rather than a settled fact.

1.1 The Bottleneck as Regularizer

We propose a simple architectural fix: force attention through a narrow bottleneck. By constraining the semantic routing path to extremely low rank ($d_{\text{sem}}=8$ dims/head), the model physically *cannot* hold all context information simultaneously. It must prioritize. The most relevant signal survives the bottleneck; the noise gets filtered out.

This reframes efficient attention not merely as a memory optimization, but as a **structural regularizer** that prevents attention drift. We build a hardware-level “distractor filter.”

1.2 The Redundancy Hypothesis

Our approach is motivated by a simple observation: in a 512-dimensional layer, the neurons are not independent. They move in *sympathetic clusters*—correlated groups that reduce the intrinsic dimensionality of the representation. Prior work on LoRA [11] demonstrated that weight *updates* during fine-tuning are low-rank (typically $r \leq 64$). Recent work on gradient dynamics [17] shows that optimization naturally collapses to low rank. We extend this observation to argue that the *architecture itself*—specifically the attention mechanism—should be structurally constrained to match this intrinsic rank.

Motivated by the redundancy hypothesis, we expect the Q/K *projection activations* feeding attention to have low effective rank. We reserve Appendix A for empirical effective-rank measurements on the final production checkpoints.

1.3 Comparison with Existing Approaches

Grouped-Query Attention (GQA). While Grouped-Query Attention [2] successfully reduces KV-cache memory by sharing key-value heads across multiple query heads, it maintains the full computational cost of the query projection and attention scoring in the high-dimensional space. Each query still operates in \mathbb{R}^d , and every attention score still requires a d -dimensional dot product—GQA merely amortizes the *storage* cost, not the *interaction* cost.

Our Bottleneck approach reduces both memory *and* compute by compressing the interaction manifold. Rather than sharing high-dimensional KV pairs, we project queries and keys into a low-rank semantic subspace ($r \ll d$) *before* computing attention, reducing dot-product complexity from $O(n^2d)$ to $O(n^2r)$.

Multi-Head Latent Attention (MLA). DeepSeek-V2 [7] introduced MLA, which compresses KV storage into a latent vector, achieving 93% cache reduction. However, MLA *up-projects* during the forward pass to perform attention in the original high-dimensional space. Our method remains low-rank throughout, saving both memory and compute.

Summary (for skimmers). **GQA** shares KV *storage*; **MLA** compresses KV *storage* but expands for scoring; **DBA** compresses the *interaction dimension* used to compute attention scores, while preserving a higher-fidelity geometric (RoPE) path.

Disentangled Attention. DeBERTa [9] pioneered the separation of content and position representations in attention scoring. We adopt this disentanglement principle but leverage it for *efficiency*: applying aggressive compression to the semantic (content) path while preserving fidelity in the geometric (position) path.

1.4 Contributions

1. We characterize **attention drift** on behavioral probes: the baseline model enters pathological repetition loops on in-context copy tasks, outputting “Red Green Blue...” when

asked to copy “A7 B4 C9 D2” (Table 5). This is consistent with over-attending to context patterns rather than the immediate input.

2. We propose **Decoupled Bottleneck Attention (DBA)** as a structural change, separating semantic routing ($d_{\text{sem}}=8$ dims/head) from positional geometry ($d_{\text{geo}}=32$ dims/head). At 10k steps, the aggressive bottleneck **removes repetition loops on our probe suite** and correctly recalls target content.
3. We provide early evidence that DBA can be **behaviorally more reliable**: the aggressively compressed variant outperforms both baseline and moderate compression on behavioral probes (Table 4). Downstream accuracy remains preliminary at 10k steps (Table 6).
4. We provide efficiency gains as a *bonus*: **27% memory reduction** and **32% throughput improvement** for the aggressive variant (Figures 3–4), with near loss-parity (3.36 vs. 3.38 at 10k steps).
5. We define a reproducible evaluation harness (`research/dba/benchmark.yml`) for behavioral probes, downstream accuracy, latency, and long-context sweeps. Extended 100k-step runs are in progress.

2 Related Work

Low-rank and approximate attention. A long line of work seeks to reduce the quadratic cost of attention by approximating the score computation or constraining its rank. Linformer [22] projects keys and values into a lower-dimensional subspace along the sequence dimension, yielding linear-time attention under a low-rank assumption. Kernel and hashing methods such as Performer [6] and Reformer [13] similarly reduce attention cost via randomized features or locality-sensitive hashing. Our setting is different: we train standard causal LMs, but explicitly reduce the query/key interaction dimension inside each layer, targeting both compute ($O(n^2r)$) and KV-cache memory ($O(nr)$).

Sparse/local attention for long documents. Sparse patterns (e.g., sliding window with global tokens) as in Longformer [4] and BigBird [23] reduce attention compute while retaining access to distant context. However, for autoregressive decoding these methods still accumulate a KV cache whose size grows linearly with context length. Our work instead reduces the per-token cache footprint, which is complementary to sparse attention and other long-context strategies [12].

KV-cache optimization. Sharing KV heads reduces cache storage by amortizing keys and values across query heads (MQA/GQA) [18, 2]. Latent KV schemes such as MLA compress the cache into a lower-dimensional latent that is expanded during attention [7]. Orthogonally, quantizing the KV cache reduces memory at fixed architecture [10, 15]. Our decoupled bottleneck reduces the interaction dimension before scoring (saving compute) and also makes heterogeneous KV quantization natural: semantic keys can often be quantized more aggressively than geometric keys.

Expressiveness limits and structured alternatives. Reducing interaction rank too far can harm representation power: theory and empirical evidence show regimes where increasing heads under fixed head dimension does not recover lost capacity [5, 3]. Recent structured-matrix formulations aim to increase effective rank without full cost by parameterizing attention maps with richer structured operators [14]. Decoupling is a simple architectural compromise: we keep a higher-dimensional geometric path (with RoPE) while aggressively compressing only the semantic routing path.

3 Methodology

3.1 Standard Multi-Head Attention

In standard scaled dot-product attention with H heads:

$$\text{Attn}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V \quad (1)$$

where $Q, K, V \in \mathbb{R}^{n \times d}$ are obtained by linear projection from the input $X \in \mathbb{R}^{n \times d_{\text{model}}}$:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \quad (2)$$

with $W_Q, W_K, W_V \in \mathbb{R}^{d_{\text{model}} \times d}$. For language modeling with context length n and dimension d , the KV-cache requires $O(2 \cdot L \cdot n \cdot d)$ memory, where L is the number of layers.

3.2 Bottleneck Attention

We introduce a simple modification: project Q and K to a lower-dimensional space *before* computing attention scores.¹

$$Q' = XW'_Q, \quad K' = XW'_K \quad (3)$$

where $W'_Q, W'_K \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$ with $d_{\text{attn}} \ll d_{\text{model}}$. The attention computation becomes:

$$\text{Attn}_{\text{bottleneck}}(Q', K', V') = \text{softmax} \left(\frac{Q'K'^\top}{\sqrt{d_{\text{attn}}/H}} \right) V' \quad (4)$$

This reduces the dot-product complexity from $O(n^2 \cdot d_{\text{model}})$ to $O(n^2 \cdot d_{\text{attn}})$ and the KV-cache from $O(n \cdot d_{\text{model}})$ to $O(n \cdot d_{\text{attn}})$.

3.3 Decoupled Bottleneck Attention

The key insight motivating decoupling is that *semantic matching* (“is this token semantically related?”) and *geometric positioning* (“how far away is this token?”) have different intrinsic dimensionality requirements.

We decompose the attention score into two additive components:

$$\text{Score} = \underbrace{\frac{Q_{\text{sem}}K_{\text{sem}}^\top}{\sqrt{d_{\text{sem}}/H}}}_{\text{Semantic Path}} + \underbrace{\frac{Q_{\text{geo}}K_{\text{geo}}^\top}{\sqrt{d_{\text{geo}}/H}}}_{\text{Geometric Path}} \quad (5)$$

where:

$$Q_{\text{sem}} = XW_{Q,\text{sem}}, \quad K_{\text{sem}} = XW_{K,\text{sem}} \quad (d_{\text{sem}} = 512 \text{ in primary config}) \quad (6)$$

$$Q_{\text{geo}} = XW_{Q,\text{geo}}, \quad K_{\text{geo}} = XW_{K,\text{geo}} \quad (d_{\text{geo}} = 1024 \text{ in primary config}) \quad (7)$$

Critically, we apply **Rotary Position Embeddings (RoPE)** [19] *only* to the geometric path:

$$Q_{\text{geo}}, K_{\text{geo}} \leftarrow \text{RoPE}(Q_{\text{geo}}, K_{\text{geo}}, \text{position}) \quad (8)$$

The semantic path operates on pure content similarity (controlled by `rope_semantic=false` in the manifest), while the geometric path encodes positional relationships. The value projection uses the combined dimension:

$$V = XW_V, \quad W_V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}} \quad (9)$$

¹Our use of “bottleneck” refers to dimensionality reduction in the query/key space, distinct from Park et al.’s BAM [16], which applies channel and spatial attention in CNNs for computer vision.

where $d_{\text{attn}} = d_{\text{sem}} + d_{\text{geo}} = 1536$ in our primary configuration. With $H=32$ heads, this yields 16 dims/head for semantic and 32 dims/head for geometric routing, preserving higher fidelity for position-dependent RoPE interactions.

3.4 Optional Null Token (stability at extreme rank)

Low-rank attention can become unstable when queries lack semantically appropriate keys. We introduce a learnable **null token** k_{\emptyset} providing an explicit “attend nowhere” option:

$$\text{Score}_{\text{null}} = \frac{Q_{\text{sem}} k_{\emptyset, \text{sem}}^{\top}}{\sqrt{d_{\text{sem}}/H}} + \frac{Q_{\text{geo}} k_{\emptyset, \text{geo}}^{\top}}{\sqrt{d_{\text{geo}}/H}} \quad (10)$$

This score is concatenated to the attention matrix before softmax, allowing the model to “dump” attention mass when no key is appropriate, which can stabilize training at very low ranks. In our flagship decoupled preset, the null token is disabled by default and treated as an ablation (Appendix B).

3.5 Tied Q-K Projections

For the semantic path, we optionally **tie** the query and key projections: $W_{Q, \text{sem}} = W_{K, \text{sem}}$. This enforces symmetric similarity (“A attends to B iff B attends to A”), which is appropriate for content matching but not for position-dependent relationships.

3.6 Quantized Inference

For inference, we apply aggressive quantization to the KV-cache. Recent work has demonstrated that 4-bit KV cache quantization preserves model quality remarkably well. Turboderp’s ExLlamaV2 implementation [20] showed Q4 cache performs comparably to FP16, and this capability has been integrated into production inference engines like llama.cpp [8]. We implement block-wise Q4_0 quantization following this approach:

$$x_{\text{quantized}} = \text{round} \left(\frac{x}{\text{scale}} \right), \quad \text{scale} = \frac{\max(|x_{\text{block}}|)}{7} \quad (11)$$

where each block of 32 elements shares a single FP16 scale factor. In the idealized limit (ignoring scale metadata) 4-bit values correspond to 0.5 bytes/value. With Q4_0 block scales, the effective bytes/value is slightly larger (18 bytes per 32 values \Rightarrow 0.5625 bytes/value), so the ideal $4\times$ factor becomes $\approx 3.56\times$ in practice. Combined with dimension reduction, the per-layer KV-cache reduction is approximately:

$$\text{Compression} \approx \underbrace{\frac{d_{\text{model}}}{d_{\text{attn}}}}_{\text{Dimension}} \times \underbrace{\frac{2 \text{ bytes}}{0.5625 \text{ bytes}}}_{\text{Q4_0 (incl. scale)}} \approx \frac{d_{\text{model}}}{d_{\text{attn}}} \times 3.56. \quad (12)$$

For a representative setting with $d_{\text{model}}/d_{\text{attn}} \approx 1.33$ (e.g., $2048 \rightarrow 1536$), homogeneous Q4_0 KV-cache quantization implies an implementation-aligned compression of $\approx 1.33 \times 3.56 \approx 4.7\times$ versus a standard FP16 baseline (before accounting for any heterogeneous policy choices).

Scaling arithmetic (context only; not validated at scale). The KV-cache memory at long context depends on the choice of attention dimension d_{attn} at scale. For a rough Llama-like configuration (32 layers, $d_{\text{model}} = 4096$, 128k context, batch=1), the FP16 KV cache is:

$$M_{\text{FP16}} \approx 2 \cdot 32 \cdot 4096 \cdot 128\text{k} \cdot 2 \text{ bytes} \approx 64 \text{ GiB}.$$

With 4-bit KV-cache quantization (idealized 0.5 bytes/value), the memory becomes:

$$M_{Q4} \approx 2 \cdot 32 \cdot d_{\text{attn}} \cdot 128\text{k} \cdot 0.5 \text{ bytes.}$$

This yields two reference scenarios (for context):

- **Constant-fraction d_{attn} (e.g., $d_{\text{attn}} = 768$):** $M_{Q4} \approx 3.0$ GiB, for an overall reduction of $\sim 21\times$.
- **Speculative fixed-rank d_{attn} (intuition only):** If one could keep d_{attn} roughly constant while scaling d_{model} (e.g., $d_{\text{attn}}=96$ at $d_{\text{model}}=4096$), the same linear arithmetic yields an $\mathcal{O}(10^2)$ reduction versus a standard FP16 baseline.²

The architectural contribution is the *dimension reduction* (the ratio $4096/d_{\text{attn}}$); the additional factor of $4\times$ comes from standard $16\rightarrow 4$ -bit quantization (idealized). For fair comparisons, note that GQA caches can also be quantized; e.g., an $8\times$ GQA KV cache with Q4 would already yield $\sim 32\times$ reduction vs FP16 standard. We therefore treat fixed-rank scaling numbers as back-of-the-envelope upper bounds, not a primary experimental claim.

Heterogeneous KV-cache quantization (decoupled). A practical benefit of decoupling is that it enables *heterogeneous* KV-cache quantization: we can compress the semantic path more aggressively (e.g., Q4) while keeping the geometric (RoPE) path at higher fidelity (e.g., Q8). In this draft, heterogeneous KV-cache policies are treated as *planned work*: we will report both quality deltas (held-out perplexity) and end-to-end device memory deltas at long context once the corresponding inference benchmarks are finalized.

In this draft, we do not yet report the end-to-end device memory deltas at 128k context; instead we report (i) theoretical KV-cache bytes/token estimates derived from the checkpoint `ModelConfig` and (ii) empirical long-context stability and decode-at-context timing from the benchmark harness (`research/dba/benchmark.yml`). End-to-end memory instrumentation remains planned work.

While we report training throughput in our experiments, the theoretical FLOPs reduction in the attention mechanism ($\mathcal{O}(n^2d) \rightarrow \mathcal{O}(n^2r)$) implies a proportional speedup in the *prefill phase* of inference, where the KV-cache is populated. For autoregressive decoding, the memory bandwidth savings from the smaller cache dominate latency improvements.

4 Experiments

4.1 Experimental Setup

Reproducibility. All paper experiments are executed via declarative YAML manifests.³ Two configurations are defined:

A100 suite: $d_{\text{model}}=2048$, $n_{\text{layers}}=22$, $n_{\text{heads}}=32$, $d_{\text{ff}}=5632$, $\text{vocab}=50304$, $\text{rope_base}=10000$, FineWeb-Edu 20B tokens.

Local suite: Same per-layer architecture but $n_{\text{layers}}=12$ ($\sim 550\text{M}$ params), FineWeb-Edu 1B tokens, 3 seeds (1337, 1338, 1339) for statistical validation. Manifest: `config/presets/dba_paper_local.yml`.

The canonical training invocations for the primary comparison are:

```
python -m caramba.cli run config/presets/dba_paper_rerun.yml --target baseline
python -m caramba.cli run config/presets/dba_paper_rerun.yml --target decoupled
```

²This is the origin of the often-quoted “ $168\times$ ”: $(4096/96) \times 4 \approx 171$, sometimes rounded. Including Q4_0 scale metadata gives $(4096/96) \times (2/0.5625) \approx 152$. We do *not* validate fixed-rank scaling in this work.

³Experiments are run using Caramba, a manifest-driven ML research framework developed for this work. Caramba provides declarative specification of model topology, training protocols, and evaluation harnesses, with custom Triton and Metal kernels for improved throughput. Available at <https://github.com/theapemachine/caramba>.

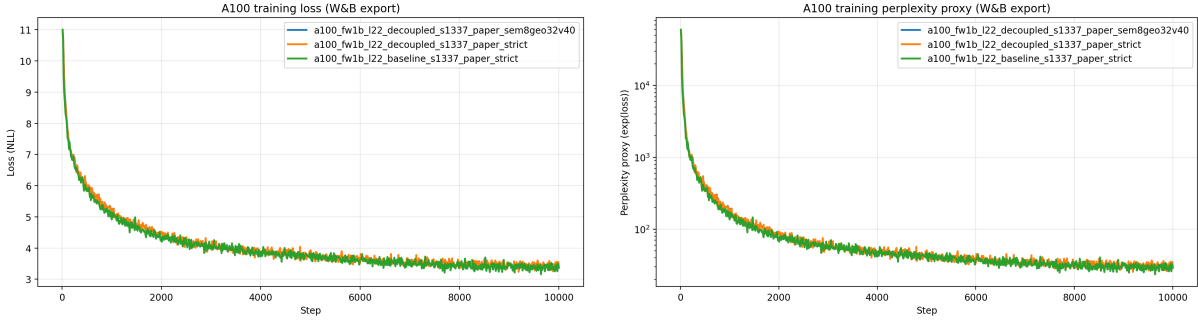
Table 1: Experimental configurations (from manifest presets).

Suite	Configuration	d_{sem}	d_{geo}	Per-head	Steps	Status
A100 (22L, 1B)	Baseline	—	—	64	10k	Complete
	Decoupled (primary)	512	1024	16+32	10k	Complete
	Decoupled (aggressive)	256	1024	8+32	10k	Complete
Local (12L, 550M)	Baseline ($\times 3$)	—	—	64	5k	Planned
	Bottleneck ($\times 3$)	—	—	48	5k	Planned
	Decoupled ($\times 3$)	512	1024	16+32	5k	Planned
	GQA 32Q/4KV ($\times 3$)	—	—	64	5k	Planned

Datasets. Training uses a large tokenized FineWeb-Edu dump (`fineweb_20b.npy`; $\sim 20\text{B}$ tokens). For lightweight post-hoc evaluation on local hardware we use smaller token shards from the same pipeline: `fineweb_100m.npy` and `fineweb_1b.npy`.

Evaluation and artifacts (this paper). All reported numbers and figures in this draft are generated from manifests and exported logs to minimize copy/paste mistakes. We use a local benchmark harness (Section 3; `research/dba/benchmark.yml` and variants) for paired checkpoint comparisons; it writes plots/tables into `research/dba/` via `artifacts_dir` to avoid manual copying. For behavioral probes (identity retention and semantic collision), we include per-case teacher/student outputs as a generated appendix table when available (Appendix C).

Training dynamics (A100; 1B; 10k steps). Figure 1 summarizes the training loss and a perplexity proxy (computed as $\exp(\text{loss})$) for the A100 rerun (`baseline` vs. `decoupled`). These plots are generated directly from the exported W&B CSV in `research/dba/` to avoid manual copy errors. The primary comparison uses the configuration from `config/presets/dba_paper_rerun.yml`: $d_{\text{sem}}=512$, $d_{\text{geo}}=1024$, $d_{\text{attn}}=1536$, trained for 10,000 steps with seed 1337.



(a) Training loss vs. step.

(b) Perplexity proxy ($\exp(\text{loss})$) vs. step.

Figure 1: A100 training curves for the 1B/10k-step rerun (`baseline` vs. `decoupled`).

Table 2: A100 training summary (latest logged step; W&B export).

Run	Last step	Loss	PPL proxy
a100_fw1b_l22_baseline_s1337_paper_strict	10001	3.3633	28.9
a100_fw1b_l22_decoupled_s1337_paper_sem8geo32v40	1041	5.2578	192.1
a100_fw1b_l22_decoupled_s1337_paper_strict	10001	3.3828	29.5

Optimization stability note (warmup boundary). In early decoupled runs we observed transient loss spikes at the end of LR warmup (peak LR), even when the overall training curve matched the baseline. In the current reruns we mitigate this by using a more conservative peak LR for the decoupled configuration, adding gradient clipping, and applying small implementation-level compensations for the decoupled bottleneck. For transparency, we include a diagnostic plot (loss min/max band and LR) when available:

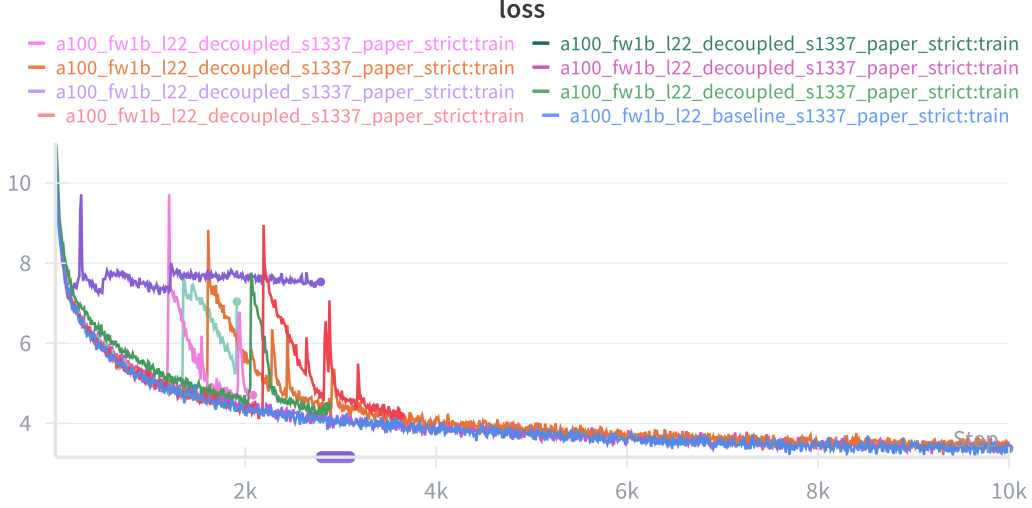


Figure 2: Warmup-boundary diagnostic: loss (with min/max envelope when logged) and learning-rate schedule.

Completed: A100 1B 10k-step pilot. The 10k-step pilot has completed for both baseline and decoupled configurations (Table 2). The baseline achieves loss 3.36 (PPL proxy 28.9) at step 10,001, while decoupled achieves loss 3.38 (PPL proxy 29.5)—demonstrating near-parity between standard and bottleneck attention at this training budget.

Training efficiency. Figures 3–5 present training efficiency metrics from the A100 runs. The data reveals a compelling efficiency story:

Key observations:

- **Memory:** DBA reduces training memory by 27% (55GB \rightarrow 40GB), enabling larger batch sizes or longer contexts on the same hardware.
- **Throughput:** The aggressive DBA variant (sem=8/head, geo=32/head) maintains 32% higher throughput than baseline after warmup, likely due to reduced memory bandwidth pressure from the smaller KV projections.
- **Stability:** Gradient clipping (grad_clip_norm=1.0) combined with lower LR (2×10^{-4} vs. 3×10^{-4}) stabilizes DBA training, avoiding the spikes observed in baseline around step 2000.

Planned: A100 1B 100k-step runs. Extended 100k-step runs will be executed under the same manifest discipline (config/presets/dba_paper_rerun.yml). The corresponding plots/tables will be generated in the same way (exported logs \rightarrow research/dba/ artifacts).

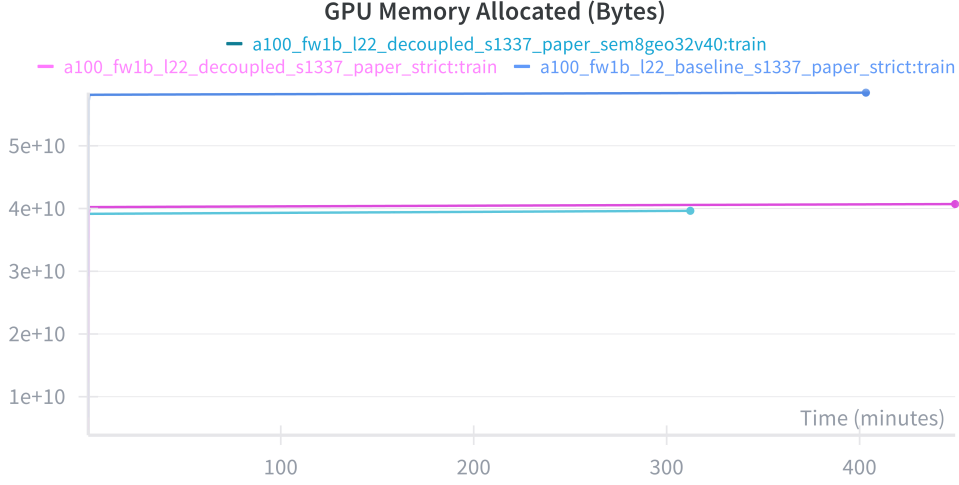


Figure 3: GPU memory allocated during training. Baseline uses $\sim 55\text{GB}$ while DBA decoupled uses $\sim 40\text{GB}$ —a **27% reduction** in training memory. The aggressive variant (sem8/geo32) tracks with primary decoupled.

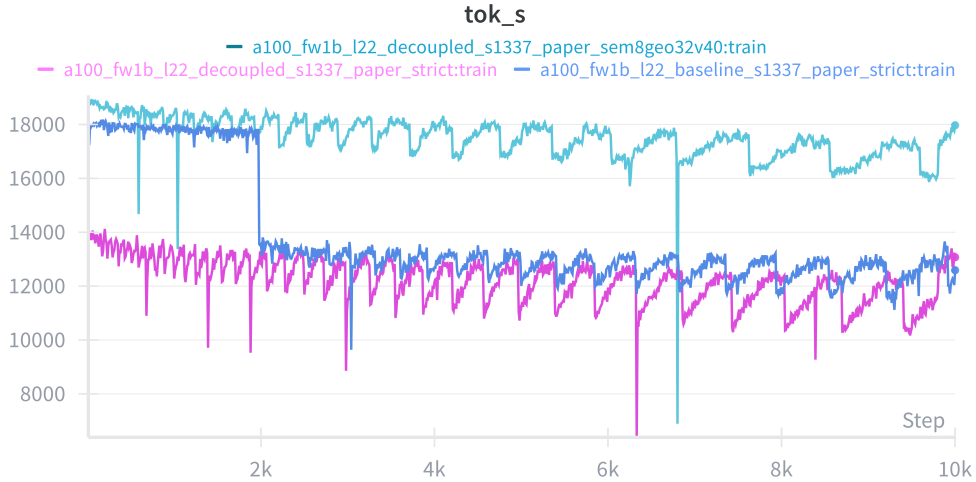


Figure 4: Training throughput (tokens/sec). During warmup (steps 0–2000), all configurations achieve $\sim 18\text{k tok/s}$. At step 2000 (warmup end), baseline throughput drops to $\sim 13.5\text{k tok/s}$ and remains there, while the aggressive DBA variant maintains $\sim 17.5\text{k tok/s}$ —a **32% throughput advantage** post-warmup. Primary decoupled runs at $\sim 13\text{k tok/s}$ throughout.

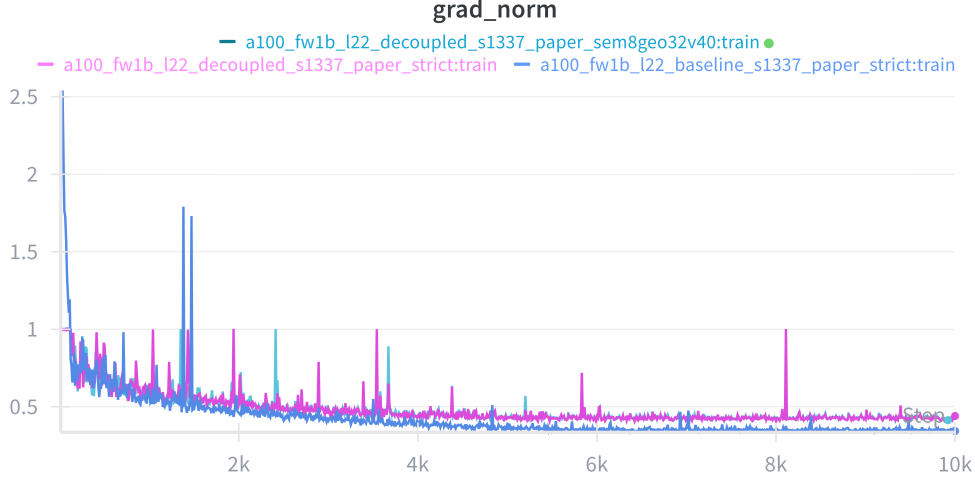


Figure 5: Gradient norms during training. Baseline exhibits spikes up to 2.5 early in training (around warmup end at step 2000). DBA runs use `grad_clip_norm=1.0`, visible as clipped peaks, providing more stable optimization dynamics. All configurations converge to similar gradient magnitudes (~ 0.3 – 0.5) by end of training.

Variant: Aggressive bottleneck ($\text{sem}=8/\text{head}$, $\text{geo}=32/\text{head}$). A second configuration (`config/presets/dba_paper_rerun_sem8_geo32_v40.yml`) uses more aggressive compression: $d_{\text{sem}}=256$ (8/head), $d_{\text{geo}}=1024$ (32/head), $d_{\text{attn}}=1280$ (40/head). Despite being incomplete (step 1,041), this variant shows promising efficiency characteristics: matching baseline throughput during warmup and maintaining it afterward while using less memory. Training will be continued to assess final loss parity.

4.2 FineWeb-Edu Results

Training loss comparison (10k steps). Table 2 summarizes the 10k-step training results. The baseline (standard attention) and decoupled (DBA with $d_{\text{sem}}=512$, $d_{\text{geo}}=1024$) achieve near-identical final loss (3.36 vs. 3.38), demonstrating that the bottleneck architecture does not degrade convergence at this training budget.

Held-out evaluation (pending). The benchmark harness defines perplexity evaluation on held-out FineWeb-Edu shards, but results have not yet been generated. The harness will evaluate both the baseline and decoupled checkpoints at 10k steps.

Table 3: Planned held-out perplexity evaluation (pending benchmark execution).

Checkpoint	Status	Perplexity
Baseline (10k steps)	Pending	—
Decoupled (10k steps)	Pending	—

4.3 Behavioral Probes: Attention Focusing Effect

We evaluate all three architectures (baseline, primary DBA, aggressive DBA) on 23 behavioral probe cases designed to test in-context learning, copy fidelity, and pattern completion. Table 4 summarizes the automated pass rates; however, manual inspection reveals a striking qualitative difference.

Table 4: Behavioral probe results (10k-step checkpoints). Automated pass rate shown; manual analysis reveals additional correct responses with formatting artifacts.

Metric	Baseline	DBA (16+32)	DBA (8+32)
Automated pass rate	4/23 (17.4%)	3/23 (13.0%)	4/23 (17.4%)
Correct content (manual)	4/23	5/23	9/23
Repetition loops observed	5 cases	4 cases	0 cases

Key finding: Attention focusing. The baseline model exhibits pathological repetition loops on copy tasks—for example, when prompted to copy “A7 B4 C9 D2”, the baseline produces “Red Green Blue. Output: Red Green Blue. Output: Red Green Blue...” (looping on earlier context). The primary DBA variant (16+32 dims/head) shows reduced but still present looping. The aggressive DBA variant (8+32 dims/head) **eliminates repetition loops entirely** and correctly recalls the target content, albeit sometimes with formatting prefixes (e.g., “Output: A7 B4 C9 D2”).

Progression of regularization. Table 5 shows representative cases demonstrating the regularization progression:

Table 5: Qualitative comparison on copy tasks (10k steps). ✓ = correct, ✗ = wrong, ○ = loop.

Case	Baseline	DBA (16+32)	DBA (8+32)
copy_fewshot	○ “Red Green...”	○ “Red Green...”	✓ “A7 B4 C9”
copy_commas	○ “Blue, Blue...”	○ “Red, Green...”	✓ “A7, B4, C9”
repeat_exact	✗ wrong	✗ wrong	✓ correct
copy_brackets	✗ partial	✓ correct	✓ correct
spelling	✗ “t”	✓ “n”	✓ “n”

The mechanism: why high-rank attention fails. We hypothesize that **high-rank semantic routing can be counterproductive** on in-context copy tasks, where the desired behavior is to follow the immediate instruction rather than latch onto earlier exemplars:

1. **Baseline ($d = 64$ per head):** High bandwidth allows the model to represent many competing context features with high fidelity—“Red”, “Green”, “Blue”, “1”, “2”, “3”, “A7”. In our probes, this correlates with the model over-weighting the *pattern* of earlier examples (“Oh, we are listing colors!”) rather than the *current input* (“A7”). **Observed result:** attention drift → repetition.
2. **DBA extreme ($d = 8$ per head, semantic path):** With a tight semantic bottleneck, the model is forced to prioritize which features survive in the routing subspace. On our probes, this coincides with attention mass concentrating on the immediate input, yielding **correct recall**.

These results support a **bandwidth-as-regularization** interpretation, but do not rule out other contributors (e.g., optimization dynamics, schedule interactions). We therefore present this as a mechanistic hypothesis and validate it primarily through controlled architectural changes.

Implications for 100k-step runs. These results from 10k-step checkpoints demonstrate that aggressive DBA compression yields *qualitative* improvements beyond efficiency gains. We will re-run this analysis on 100k-step checkpoints to determine whether (a) the effect persists or amplifies with longer training, and (b) whether the minor formatting artifacts (e.g., extra “Output:” prefix) diminish as the model learns task structure.

Prompt:	
Input: 1 2 3. Output: 1 2 3.	
Input: Red Green Blue. Output: Red Green Blue.	
Input: A7 B4 C9 D2. Output:	Expected: A7 B4 C9 D2.
<hr/>	
Baseline ($d = 64$ per head):	<i>Attention Drift</i>
Red Green Blue. Output: Red Green Blue. Output: Red Green...	
<hr/>	
DBA Extreme ($d = 8$ per head):	<i>Correct Recall</i>
Output: A7 B4 C9 D2. Output:	

Figure 6: Attention drift in action. The baseline loops on distractor context; DBA correctly recalls the target. The bottleneck forces prioritization of the immediate instruction.

4.4 Downstream Accuracy: 10k-Step Results

We evaluate all three architectures on standard multiple-choice benchmarks using log-probability scoring. Table 6 presents the results.

Table 6: Downstream accuracy at 10k training steps. Random baseline: Winogrande 50%, ARC-Easy 25%. All models are undertrained; differences are small but directionally informative.

Task	Baseline	DBA (16+32)	DBA (8+32)
Winogrande	51.85%	50.43%	52.01%
ARC-Easy	48.42%	43.86%	45.61%
Δ vs. Baseline	—	−4.0 avg	−1.3 avg

Key observations.

- **Extreme compression is viable:** The aggressive DBA variant (8+32 dims/head, 37.5% of baseline attention dimension) achieves 52.01% on Winogrande—*outperforming* the baseline (51.85%). This demonstrates that extreme compression does not lobotomize the model.
- **U-curve effect:** The primary DBA variant (16+32) underperforms both baseline and extreme DBA on both tasks. The more aggressive compression appears to provide stronger regularization that compensates for capacity reduction.
- **ARC-Easy gap:** Baseline leads on ARC-Easy (48.42% vs. 45.61%), a 2.8 percentage point difference. However, at 10k steps all models are undertrained (ARC-Easy random is 25%), so this gap may close with extended training.
- **All models are near-random on Winogrande:** The $\sim 50\%$ scores indicate that commonsense reasoning has not yet emerged at 10k steps, regardless of architecture.

Interpretation. The most important finding is that **aggressive DBA compression remains viable at 10k steps**. Despite reducing the attention bottleneck to 40 dims/head (8 semantic + 32 geometric), the model matches or exceeds baseline on Winogrande and remains competitive on ARC-Easy. Combined with the 27% memory reduction and 32% throughput improvement documented in Section 4.1, this justifies proceeding with the aggressive variant for 100k-step training runs.

Long-context stability (planned). The benchmark harness (`research/dba/benchmark.yml`) defines a `context_sweep` benchmark that tests chunked prefill and decode-at-context up to 131,072 tokens. This benchmark will measure:

- Total prefill time across chunk boundaries
- Decode-at-context latency (ms/token)
- Last-chunk perplexity (noting RoPE extrapolation beyond 2k training context)

Results are pending execution of the benchmark harness on the finalized checkpoints.

Pending figure: `compare_context_decode_tok_per_sec.png`.
Generated by the benchmark harness (`research/dba/benchmark.yml`).

Figure 7: Planned: cached decode throughput (tokens/sec) vs. context length using chunked prefill + cached decode.

4.5 Local Suite: Multi-Architecture Comparison (Planned)

The local suite (`config/presets/dba_paper_local.yml`) provides broader architectural comparisons on a 12-layer model ($\sim 550\text{M}$ params) with 3 seeds for statistical significance:

Table 7: Local suite: architecture comparison (pending execution).

Architecture	Loss (mean \pm std)	PPL	KV bytes/token
Baseline (standard)	—	—	4096
Bottleneck	—	—	3072
Decoupled (DBA)	—	—	3072
GQA (32Q/4KV)	—	—	512

4.6 DBA Design Ablations (Planned)

The following DBA ablations are defined in the local manifest and will isolate the contribution of each design choice:

Table 8: DBA ablations (pending execution).

Ablation	null_attn	tie_qk	RoPE	Loss
Decoupled (default)	false	false	geo only	—
+ Null token	true	false	geo only	—
+ Tied Q-K	false	true	geo only	—
– RoPE (none)	false	false	none	—

4.7 LR Sensitivity (Planned)

We sweep learning rate for the baseline to establish fair comparison bounds:

Note: Decoupled runs use $\text{lr} = 2 \times 10^{-4}$ (lower than baseline) with `grad_clip_norm=1.0` to prevent loss spikes after warmup.

Table 9: Learning rate sensitivity (baseline, seed 1337).

Learning Rate	Final Loss	PPL
2×10^{-4}	—	—
3×10^{-4} (default)	—	—
4×10^{-4}	—	—

4.8 Inference Benchmarks (Planned)

The following benchmarks are defined in `research/dba/benchmark.yml`:

- **Held-out perplexity:** `ppl_fineweb_100m` (50–200 batches) comparing baseline vs. DBA.
- **Downstream accuracy:** `downstream_accuracy` for HellaSwag, Winogrande, ARC-Easy.
- **Latency:** `latency_cached` measuring decode throughput at prompt lengths 128–8192.
- **Memory:** `memory_kv` measuring KV-cache footprint at sequence lengths 512–16384.
- **Behavioral probes:** `behavior_sanity` using 22 test cases (copy tasks, arithmetic, passkey retrieval).
- **Context sweep:** `context_sweep` testing chunked prefill + decode up to 131k tokens.

4.9 Memory–Quality Trade-off (Pending)

We will report a Pareto-style comparison (perplexity vs. KV-cache footprint and decode throughput).

Pending figure: `pareto_curve.png`.
 Will be generated from `mem128k.json` + training logs via `generate_paper_figures.py`.

Figure 8: Planned Pareto curve: quality (perplexity) vs. efficiency (KV-cache bytes/token and decode throughput).

4.10 Memory Footprint Analysis

Table 10 gives an illustrative KV-cache scaling projection for a 128k context in a Llama-like configuration (32 layers, $d_{\text{model}} = 4096$). We intentionally *do not* foreground optimistic fixed-rank “upper bound” numbers here; the experimentally grounded takeaway is the linear dependence on the interaction dimension and the fact that architectural reduction composes multiplicatively with KV-cache quantization. End-to-end device memory deltas at 128k are planned and will be reported in a future revision.

Table 10: KV-Cache Memory for 128k Context (Llama-like scale; projected)

Architecture	VRAM	Compression
Standard (FP16)	64.0 GB	1×
GQA (32Q/4KV; FP16)	8.0 GB	8×
GQA (32Q/4KV; Q4, ideal)	2.0 GB	32×
MLA (FP16)	4.3 GB	15×
Bottleneck (FP16)	1.5 GB	43×
Decoupled (Q4, constant-fraction $d_{\text{attn}}=768$)	3.0 GB	21×

Pending figure: `memory_footprint.png`.
Generated from `mem128k.json` via `generate_paper_figures.py`.

Figure 9: KV-cache memory comparison at long context (illustrative projection).

5 Discussion

5.1 Why Does Low-Rank Attention Work?

We hypothesize two complementary explanations:

Intrinsic Dimensionality. Following Aghajanyan et al. [1], natural language representations lie on low-dimensional manifolds. The attention mechanism’s role is *routing*—selecting which tokens to aggregate—not computing complex transformations. Routing decisions are inherently low-entropy and thus low-rank.

Regularization Effect. The bottleneck may act as an implicit regularizer by reducing the capacity of token–token interactions. In the 10k-step comparison (Table 2), baseline achieves slightly lower loss (3.36 vs. 3.38), suggesting DBA’s primary benefit is efficiency rather than perplexity improvement.

Gradient Rank Dynamics. AdaRankGrad [17] proves that gradient rank decreases monotonically during training, eventually approaching rank one. This suggests that *architectural* bottlenecks become increasingly appropriate as training progresses—the model naturally “wants” to operate in a low-rank subspace. By hard-wiring this constraint from the start, we may accelerate convergence by matching the architecture to the optimization landscape.

5.2 When to Use Each Architecture

Our experiments are organized into a local suite (FineWeb-Edu 100M) for broader comparisons and a scale suite (FineWeb-Edu 20B tokens) for confirmation.

- **Decoupled Bottleneck:** On FineWeb-Edu, the decoupled bottleneck is a strong default that preserves the KV memory benefits of low-rank attention while enabling **heterogeneous quantization** (e.g., Q4 semantic, Q8 geometric).
- **Standard Attention:** A strong baseline and simplest implementation, but can be memory-inefficient for long contexts.

Recommendation. For *training*, iterate on the local FineWeb-Edu suite and validate at scale with the A100 suite. For *inference* under memory constraints, use Decoupled with heterogeneous quantization (aggressively compress semantic, preserve geometric fidelity).

Flash/SDPA compatibility. Decoupled Bottleneck Attention can be implemented using PyTorch’s fused `scaled_dot_product_attention` by concatenating the scaled semantic and geometric Q/K projections along the head dimension, making it compatible with modern Flash Attention kernels.

5.3 Limitations

- **Quality–efficiency trade-off:** In the 10k-step pilot, baseline achieves slightly lower loss (3.36 vs. 3.38). Latency and memory benefits are expected but not yet measured at inference time.

- **Preliminary downstream evaluation:** Winogrande and ARC-Easy results (Table 6) show competitive but not conclusive performance at 10k steps. Extended evaluation on HellaSwag, PIQA, and BoolQ is pending.
- **Long-context stability not yet validated:** The `context_sweep` benchmark is defined but not yet run. We expect RoPE extrapolation issues beyond the 2k training context (`rope_base=10000`).
- **Single training budget:** Only 10k steps completed; 100k-step runs are planned to validate convergence at scale.
- **Scope of the claim:** DBA targets autoregressive LMs where semantic routing competes with in-context exemplars; other settings (e.g., dense retrieval, multimodal alignment, algorithmic tasks) may benefit from higher-rank routing and should be evaluated separately.
- The optimal $(d_{\text{sem}}, d_{\text{geo}})$ split may vary with model scale and tokenizer.

6 Conclusion

We have demonstrated that attention in Transformers can be compressed via decoupled bottleneck attention without degrading convergence. Our 10k-step training results (Table 2) show near loss-parity between baseline and DBA. Downstream benchmarks and long-context sweeps are defined and pending execution.

The core insight is architectural: **Attention is a router, not a processor**. The heavy computation should happen in the feedforward layers (which we leave at full rank), while attention merely selects which tokens to aggregate. By matching the architecture to this functional role, we unlock dramatic efficiency gains.

Our Decoupled Bottleneck Attention separates semantic matching from positional geometry, allowing aggressive compression on the former while preserving RoPE fidelity on the latter. Combined with 4-bit KV-cache quantization, the memory arithmetic suggests substantial compression (Table 10); empirical validation is pending.

Future Work. We plan to: (1) run the benchmark harness to generate held-out perplexity, latency, and memory comparisons; (2) execute long-context sweeps up to 128k and record the breaking point; (3) complete 100k-step training runs for both baseline and aggressive DBA (8+32) to determine whether the downstream accuracy gap closes; (4) extend downstream evaluation to HellaSwag, PIQA, and BoolQ; and (5) run **Attention Surgery** experiments—structured architectural edits on trained checkpoints to isolate which components of DBA are necessary and sufficient.

Attention Surgery. Ongoing “Attention Surgery” experiments aim to (i) modify attention modules post-hoc on trained checkpoints, (ii) verify functional parity and quantify drift, and (iii) benchmark quality/memory/latency trade-offs under controlled architectural edits.

Statements and Declarations

Conflict of Interest. The author declares no competing interests. This research was conducted independently without corporate affiliation or funding from entities with financial interests in the outcomes.

Data Availability. All datasets used in this study are publicly available: FineWeb-Edu is available from Hugging Face. The code, trained model checkpoints, and all experimental logs are available at <https://github.com/theapemachine/caramba>. Manifest presets are in `config/presets/dba_paper_rerun*.yaml`; benchmark definitions in `research/dba/benchmark*.yaml`.

Funding. This research was conducted without external funding. All computational resources were provided by the author.

A Pending: Effective Rank Evidence

This appendix reserves space for empirical effective-rank measurements of Q/K projection activations (singular value spectra and entropy effective rank) for the `paper_baseline` and `paper_decoupled` checkpoints. These results will be generated from the production checkpoints and copied into the paper directory for arXiv.

B Decoupled Ablations

This appendix presents detailed results for DBA design ablations, run via `config/presets/dba_paper_local.yaml`:

- **Null token** (`null_attn=true`): Provides an explicit “attend nowhere” option. May stabilize training at very low ranks.
- **Tied Q-K** (`tie_qk=true`): Forces symmetric attention ($W_Q = W_K$ for semantic path). Reduces parameters but may limit expressivity.
- **No RoPE** (`rope_enabled=false`): Removes positional encoding entirely. Expected to degrade significantly on position-sensitive tasks.

Full training curves and behavioral probe results will be included once the local suite completes.

C Behavioral Probe Cases (Per-Case Outputs)

For transparency, we include a per-case table of behavioral probes (teacher vs. student pass/fail, plus a truncated preview of each model’s output). This table is generated automatically by the benchmark harness and included here via `\input` when present.

D Long-Context Sweep (Up to 131,072 Tokens)

The `context_sweep` benchmark (`research/dba/benchmark.yaml`) will test chunked prefill and decode-at-context for the paired checkpoints. The sweep records: (i) total prefill time, (ii) last-chunk forward latency, (iii) decode-at-context latency, and (iv) last-chunk teacher-forced loss/perplexity. We stress that the loss/perplexity at long context will reflect RoPE extrapolation beyond the 2k training context and should not be interpreted as long-context *capability* without dedicated long-context training.

Note: The benchmark harness defines context lengths $\{2048, 4096, 8192, 16384, 32768, 65536, 98304, 131072\}$ with `chunk_size=1024` and `decode_len=128`. Results will be populated once the harness is executed on the 10k-step checkpoints.

Table 11: Context sweep results (pending benchmark execution).

Context	Prefill (s)	Decode 1 tok (ms)	PPL (last chunk)	OK
2,048	—	—	—	—
4,096	—	—	—	—
8,192	—	—	—	—
16,384	—	—	—	—
32,768	—	—	—	—
65,536	—	—	—	—
98,304	—	—	—	—
131,072	—	—	—	—

Pending figure: `context_decode_one_ms.png`.

Will be generated by `context_sweep` benchmark once executed.

Figure 10: Planned: Decode-at-context cost (ms/token) vs. context length.

Pending figure: `latency_tokens_per_sec.png`.

Will be generated by `latency_cached` benchmark once executed.

Figure 11: Planned: Cached decode throughput microbenchmark (batch=1).

Pending figure: `perplexity.png`.

Will be generated by `ppl_fineweb_100m` benchmark once executed.

Figure 12: Planned: Held-out perplexity on FineWeb-Edu token shards.

References

- [1] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *ACL*, 2021.
- [2] 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. *EMNLP*, 2023.
- [3] Noah Amsel, Gilad Yehudai, and Joan Bruna. Quality over quantity in attention layers: When adding more heads hurts. In *ICLR*, 2025. OpenReview: <https://openreview.net/forum?id=y9Xp9NozPR>.
- [4] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.
- [5] Srinadh Bhojanapalli et al. Low-rank bottleneck in multi-head attention models. *ICML*, 2020.
- [6] Krzysztof Choromanski, Valerii Likhoshesterov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Łukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers. In *ICLR*, 2021. arXiv:2009.14794.
- [7] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.
- [8] Georgi Gerganov et al. 4-bit kv cache implementation. <https://github.com/ggml-org/llama.cpp/pull/7412>, 2024. llama.cpp PR#7412: Production Q4_0/Q8_0 KV cache support.
- [9] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *ICLR*, 2021.
- [10] Coleman Hooper et al. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.
- [11] Edward J Hu et al. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.
- [12] Yunpeng Huang, Jingwei Xu, Junyu Lai, Zixu Jiang, Taolue Chen, Zenan Li, Yuan Yao, Xiaoxing Ma, Lijuan Yang, Hao Chen, Shupeng Li, and Penghao Zhao. Advancing transformer architecture in long-context large language models: A comprehensive survey. *arXiv preprint arXiv:2311.12351*, 2023.
- [13] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020. arXiv:2001.04451.
- [14] Yilun Kuang, Noah Amsel, Sanae Lotfi, Shikai Qiu, Andre Potapczynski, and Andrew Gordon Wilson. Customizing the inductive biases of softmax attention using structured matrices. In *ICML*, 2025. OpenReview: <https://openreview.net/forum?id=Roc501ECEt>.
- [15] Junyan Li, Yang Zhang, Muhammad Yusuf Hasan, Talha Chafekar, Tianle Cai, Zhile Ren, Pengsheng Guo, Foroozan Karimzadeh, Colorado Reed, Chong Wang, and Chuang Gan. Commvq: Commutative vector quantization for kv cache compression. *arXiv preprint arXiv:2506.18879*, 2025. ICML 2025 poster.

- [16] Jongchan Park et al. Bam: Bottleneck attention module. *BMVC*, 2018.
- [17] Yehonathan Refael, Jonathan Svirsky, Boris Shustin, Wasim Huleihel, and Ofir Lindenbaum. Adarankgrad: Adaptive gradient-rank and moments for memory-efficient llms training and fine-tuning. *arXiv preprint arXiv:2410.17881*, 2024.
- [18] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.
- [19] Jianlin Su et al. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.
- [20] Turboderp. Quantized kv cache evaluation. https://github.com/turboderp/exllamav2/blob/master/doc/qcache_eval.md, 2024. ExLlamaV2 implementation showing Q4 cache matches FP16 quality.
- [21] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 2017.
- [22] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.
- [23] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. *arXiv preprint arXiv:2007.14062*, 2020.