Decoupled Bottleneck Attention:

Scaling Efficient Transformers via Low-Rank Semantic Routing

Daniel Owen van Dommelen
Independent Research
theapemachine@gmail.com

January 2026

Abstract

The Key-Value (KV) cache in Transformer language models scales linearly with sequence length and attention dimension, creating a practical memory bottleneck for long-context inference. We propose **Decoupled Bottleneck Attention (DBA)**, an architectural modification that separates *semantic routing* (which token attends to which) from *positional geometry* (how far away it is). DBA computes attention scores as the sum of a low-rank semantic path and a higher-fidelity geometric path (with RoPE), enabling aggressive cache compression where it matters most.

This paper is a *manifest-first* report: all training runs are executed via the Caramba manifest runner (`python -m caramba.cli run`) using the paper presets in `config/presets/`. We are rerunning the primary baseline vs. DBA comparison in Caramba (switching from the earlier production codebase) and will update the reported step counts and figures as the new runs complete. Training uses FineWeb-Edu tokens and a lightweight benchmark script (`eval_ckpt.py`) for held-out evaluation. We also demonstrate long-context inference *stability* up to **131,072 tokens** on consumer hardware, while noting that perplexity at extreme context lengths can degrade sharply due to RoPE extrapolation (training context is 2k and `rope_base=10000`).

Keywords: Transformer, attention mechanism, low-rank, KV-cache, memory efficiency, quantization, long context, rotary position embeddings

1 Introduction

Modern Transformer architectures [21] achieve remarkable performance across language modeling, translation, and reasoning tasks. However, their quadratic attention complexity and linear KV-cache growth present fundamental scalability challenges for long-context applications.

1.1 The Redundancy Hypothesis

We begin with 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.

1.2 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 $O(n^2 d)$ to $O(n^2 r)$.

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.

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.

1.3 Contributions

1. We propose DBA: decoupling semantic routing from positional geometry, and evaluate a 1B-scale configuration where the semantic path is substantially smaller than the geometric path.

2. We propose **Decoupled Bottleneck Attention (DBA)**, which implements this decoupling by separating semantic and geometric scoring paths with asymmetric dimensionality.

3. We introduce a **Null Token** mechanism that provides an explicit "attend nowhere" option, and treat it as an ablation (Appendix B).

4. We provide a production-first evaluation workflow (`eval_ckpt.py`) that reports held-out perplexity, cached decode latency microbenchmarks, and long-context sweep behavior from trained checkpoints. Detailed KV-cache memory measurements at 128k (end-to-end device deltas) are planned and kept as placeholders in this draft.

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 ($O(n^2 r)$) 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:

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

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

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V \tag{2}$$

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]

$$Q' = XW'_Q, \quad K' = XW'_K \tag{3}$$

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:

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

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 $O(n \cdot d_{\text{attn}})$.

[1] Our 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.

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:

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

where:

$$Q_{\text{sem}} = XW_{Q,\text{sem}}, \quad K_{\text{sem}} = XW_{K,\text{sem}} \qquad (\text{e.g., } d_{\text{sem}} = 512) \tag{6}$$

$$Q_{\text{geo}} = XW_{Q,\text{geo}}, \quad K_{\text{geo}} = XW_{K,\text{geo}} \qquad (\text{e.g., } d_{\text{geo}} = 1024) \tag{7}$$

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

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

The semantic path operates on pure content similarity, while the geometric path encodes positional relationships. The value projection uses the combined dimension:

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

where $d_{\text{attn}} = d_{\text{sem}} + d_{\text{geo}}$. In our production presets, $(d_{\text{sem}}, d_{\text{geo}})$ is chosen as a function of model size, with a higher-dimensional geometric path to preserve RoPE fidelity.

3.4 The Null Token Mechanism

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:

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

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:

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

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:

$$\text{Compression} \approx \underbrace{\frac{d_{\text{model}}}{d_{\text{attn}}}}_{\text{Dimension}} \times \underbrace{\frac{2 \text{ bytes}}{0.5625 \text{ bytes}}}_{\text{Q4\_0 (incl. scale)}} \approx \frac{d_{\text{model}}}{d_{\text{attn}}} \times 3.56. \tag{12}$$

For a representative setting with $d_{\text{model}}/d_{\text{attn}} \approx 1.33$ (e.g., $2048 \to 1536$), 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:

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

This yields two reference scenarios (for context):

- **Constant-fraction d_{attn} (e.g., $d_{\text{attn}} = 768$):** $M_{\text{Q4}} \approx 3.0$ 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_{attn}=96 at d_{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$\to$4-bit quantization (idealized). For fair comparisons, note that GQA caches can also be quantized; e.g., an 8$\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 `eval_ckpt.py`. End-to-end memory instrumentation remains planned work.

While we report training throughput in our experiments, the theoretical FLOPs reduction in the attention mechanism ($O(n^2 d) \to O(n^2 r)$) 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.

[2]This 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.

4 Experiments

4.1 Experimental Setup

Reproducibility discipline (manifest-only). All paper experiments are executed from a single manifest preset (`config/presets/dba\_paper\_rerun.yml`) using the Caramba CLI (`python -m caramba.cli run`). The two canonical training invocations for the main comparison are:

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

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

Datasets. Training uses a large tokenized FineWeb-Edu dump (`fineweb_20b.npy`; ∼20B 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 in this draft are produced by `eval_ckpt.py`, which runs (i) held-out perplexity, (ii) KV-cache bytes-per-token estimates derived from the checkpoint's `ModelConfig`, (iii) cached decode latency microbenchmarks, and (iv) a context-length sweep that tests long-context prefill and measures decode-at-context cost. The script also supports optional diagnostics (e.g., long-seq approximation drift for decoupled attention and correctness/parity checks) behind explicit flags. It writes `report.json` and CSVs into the run directory; plots are optional.

Training dynamics (A100; 1B; 100k steps). Figure 1 summarizes the training loss, perplexity proxy, throughput (tokens/sec), and wall-clock time per step logged during the A100 runs for both `decoupled` and `baseline`.

Missing figure: `A100-1b-100k-loss.png`.

(a) Training loss vs. step.

Missing figure: `A100-1b-100k-ppl.png`.

(b) Training perplexity (logged metric) vs. step.

Missing figure: `A100-1b-100k-tok_s.png`.

(c) Training throughput (tokens/sec) vs. step.

Missing figure: `A100-1b-100k-ms_step.png`.

(d) Step time (ms/step) vs. step.

Figure 1: A100 training curves for the 1B/100k-step runs (`baseline` vs. `decoupled`).

4.2 FineWeb-Edu Results (Current)

Decoupled checkpoint (A100 training; local evaluation). For the checkpoint `runs/a100_fw1b_l22_decoupled_s1337_paper_strict/ckpt_step100000.pt`, `eval_ckpt.py` reports the following held-out language modeling quality on FineWeb-Edu token shards:

Table 1: Held-out perplexity for the decoupled 1B checkpoint (100k steps).

Dataset shard	Avg. NLL	Perplexity
FineWeb-Edu 100M tokens (`fineweb_100m.npy`)	3.2811	26.604
FineWeb-Edu 1B tokens (`fineweb_1b.npy`)	3.2811	26.604

Baseline vs. decoupled (100k steps; apples-to-apples). We evaluate the paired checkpoints `ckpt_step100000.pt` for both runs (`decoupled` vs. `baseline`) on Apple MPS using identical, pre-generated test inputs for fairness. Table 2 reports held-out perplexity, and Figures 2 and 3 summarize cached decode throughput and robustness probes.

Table 2: Held-out perplexity at 100k steps (MPS) for decoupled vs. baseline. Lower is better.

Dataset shard	DBA PPL	Baseline PPL
FineWeb-Edu 100M tokens (`fineweb_100m.npy`)	26.604	23.835
FineWeb-Edu 1B tokens (`fineweb_1b.npy`)	26.604	23.835

Missing figure: `compare_perplexity.png`.

(a) Perplexity comparison (100k steps).

Missing figure: `compare_latency_tokens_per_sec.png`.

(b) Cached decode throughput vs. prompt length (100k steps).

Figure 2: Primary comparison at 100k steps (MPS). DBA maintains higher cached decode throughput at longer prompts while using a substantially smaller KV cache.

Missing figure: `compare_robustness_passkey_accuracy.png`.

(a) Passkey accuracy.

Missing figure: `compare_robustness_passkey_margin.png`.

(b) Passkey logprob margin.

Missing figure: `compare_robustness_kl.png`.

(c) Prompt KL stability.

Figure 3: Robustness probe comparison at 100k steps (MPS), using paired synthetic testcases.

Long-context stability up to 131,072 tokens (consumer hardware). Using the same decoupled checkpoint and `eval_ckpt.py`'s context sweep, we successfully completed chunked prefill and a decode-at-context step up to **131,072 tokens** on an Apple M4 Max with 128GB unified memory (`ok=true` at all tested lengths; Appendix C). At very long contexts, runtime becomes dominated by system memory pressure (swapping), but the model remains functionally stable. We emphasize that the extremely high perplexity at $\geq$8k context in this sweep reflects RoPE extrapolation outside the model's training context (`block_size=2048`) rather than an architectural memory failure.

Long-context decode throughput (planned figure). For long contexts beyond the training window, one-shot prefill benchmarks are invalid because the model hard-fails when `seq_len > block_size`. We therefore measure long-context decode throughput using chunked prefill with caches followed by a multi-token cached decode loop (`eval_ckpt.py -context-decode`), and will report the resulting comparison curve once generated:

4.3 Ablations and Additional Benchmarks (Planned)

This draft intentionally removes legacy result sections from earlier prototypes. The following experiments are planned and will be reported once the corresponding runs complete:

- **Baseline vs. decoupled (primary):** completed and reported in Section 3 (Tables 2, Figures 2 and 3).
- **Context-length breaking point up to 128k:** context sweep to identify stability limits and throughput curves.

Pending figure: `compare_context_decode_tok_per_sec.png`.
Generated by `eval_ckpt.py` with `-context-decode` and copied into the paper directory.

Figure 4: Planned: cached decode throughput (tokens/sec) vs. context length using chunked prefill + cached decode.

- **Downstream tasks:** HellaSwag / ARC / etc. (pending integration into the evaluation harness).
- **DBA design ablations:** null token, tied Q–K, RoPE variants, and semantic/geometric dimension splits.

4.4 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 5: Planned Pareto curve: quality (perplexity) vs. efficiency (KV-cache bytes/token and decode throughput).

4.5 Memory Footprint Analysis

Table 3 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 3: 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_{attn}=768)	3.0 GB	21×

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

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

Figure 6: KV-cache memory comparison at long context (illustrative projection).

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 our current 1B/100k-step comparison on FineWeb-Edu shards, the standard-attention baseline attains lower perplexity (Table 2), so the primary observed benefit of DBA 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:** at 100k steps on FineWeb-Edu shards, the baseline achieves lower perplexity than DBA (Table 2), while DBA is faster at longer prompts and uses a substantially smaller KV cache (Figure 2).
- **Limited downstream evaluation:** this draft does not yet include standardized benchmarks (HellaSwag / ARC / etc.).
- **Long-context quality is not validated:** we demonstrate *stability* up to 131,072 tokens on consumer hardware, but perplexity degrades sharply beyond the 2k training context due to RoPE extrapolation (`rope_base=10000`) and lack of long-context training / scaling.
- 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 contains significant redundancy. In the current production-only draft, we report held-out perplexity for a decoupled 1B checkpoint trained for 100k steps (Table 1) and an apples-to-apples comparison against a standard-attention baseline trained under the same discipline (Table 2; Figures 2 and 3). We also demonstrate successful long-context *stability* up to 131,072 tokens on consumer hardware (Appendix C).

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 that 128k-context inference can become *feasible* on consumer hardware under fixed-rank scaling assumptions (Figure 6); however, this is a projection and we do not claim validated 128k *quality* in this work.

Future Work. We plan to: (1) push context-length evaluation to 128k and record the breaking point; (2) add standardized downstream tasks; and (3) run **Attention Surgery** experiments—structured architectural edits on trained checkpoints to isolate which components of DBA are necessary and sufficient.

Attention Surgery and the Caramba framework. Ongoing "Attention Surgery" experiments are implemented in the extracted `caramba` framework, now hosted at `https://github.com/TheApeMachine/caramba`. These experiments aim to (i) modify attention modules post-hoc, (ii) verify functional parity and quantify drift, and (iii) benchmark quality/memory/latency trade-offs under controlled 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/experiments`.

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 Pending: Decoupled Ablations

This appendix reserves space for DBA ablations (null token, tied Q–K, RoPE variants, semantic/geometric dimension splits), executed under the same manifest-driven discipline and reported via `eval_ckpt.py`.

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

We report the `eval_ckpt.py` context sweep for the decoupled checkpoint, using chunked prefill and then measuring a single decode step at the final context length. 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 reflects RoPE extrapolation beyond the 2k training context and should not be interpreted as long-context *capability* without dedicated long-context training.

Table 4: Context sweep results for `ckpt_step100000.pt` on Apple M4 Max (MPS).

Context	Prefill (s)	Decode 1 tok (ms)	PPL (last chunk)	OK
2,048	0.65	17.30	23.73	true
4,096	1.89	40.59	16.87	true
8,192	6.21	47.50	548.60	true
16,384	22.58	61.56	1058.30	true
32,768	105.21	165.49	2536.44	true
65,536	753.66	297.93	1960.31	true
98,304	1251.93	307.68	2301.31	true
131,072	2109.60	295.40	2588.94	true

Pending figure: `context_decode_one_ms.png`.
Generated by `eval_ckpt.py` and copied into the paper directory for arXiv.

Figure 7: Decode-at-context cost (ms/token) vs. context length from `eval_ckpt.py`. At long contexts, runtime is dominated by system memory pressure (swapping) rather than an architectural failure mode; the sweep remained stable through 131,072 tokens (`ok=true`).

Missing figure: `latency_tokens_per_sec.png`.

Figure 8: Cached decode throughput microbenchmark (batch=1) from `eval_ckpt.py`.

Missing figure: `perplexity.png`.

Figure 9: Held-out perplexity on FineWeb-Edu token shards from `eval_ckpt.py`.

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 Likhosherstov, 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=Roc5O1ECEt`.

[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.