Do Transformers Need Three Projections? Systematic Study of QKV Variants

Anonymous Authors[1]

Abstract

Transformers have become the standard solution for various AI tasks, with the query, key, and value (QKV) attention formulation playing a central role. However, the individual contribution of these three projections and the impact of omitting some remain poorly understood. We systematically evaluate three projection sharing constraints: a) Q–K=V (shared key-value), b) Q=K–V (shared query-key), and c) Q=K=V (single projection). The last two variants produce symmetric attention maps; to address this, we also explore asymmetric attention via 2D positional encodings. Through experiments spanning synthetic tasks, vision (MNIST, CIFAR, TinyImageNet, anomaly), and language modeling (300M and 1.2B parameter models on 10B tokens), we discovered that our transformers perform on par or occasionally better than the QKV transformer. In language modeling, Q–K=V projection sharing achieves 50% KV cache reduction with only 3.1% perplexity degradation. Crucially, projection sharing is complementary to head sharing (GQA/MQA): combining Q–K=V with GQA-4 yields 87.5% cache reduction, while Q–K=V + MQA achieves 96.9%—enabling practical on-device inference. We show that Q–K=V preserves quality because keys and values can occupy similar representational spaces and attention operates in a low-rank regime, whereas Q=K-V breaks attention directionality. Our results establish projection sharing as a new optimization axis for memory-efficient transformers, especially for edge deployment.

1. Introduction

Since their inception, Transformers (Vaswani et al., 2017) have evolved from language-specific tools into the backbone of multimodal AI (Yin et al., 2024; Han et al., 2022). However, as context windows expand and the demand for real-time inference grows, the research community has shifted focus toward architectural efficiency. High-efficiency variants—ranging from linear-complexity models like the Performer and Linformer to modern implementations like Ring Attention and blockwise schemes—seek to alleviate the quadratic bottleneck of the self-attention mechanism (Tay et al., 2022).

Despite these advances, a fundamental structural question remains: is the tripartite $(\text{Query}, \text{Key}, \text{Value})$ projection truly necessary? While Convolutional Neural Networks (CNNs) (LeCun et al., 1995) and contemporary State Space Models (SSMs) (Gu & Dao, 2023) often utilize more unified internal representations, Transformers maintain a persistent redundancy across their projection matrices. To investigate this, we propose and evaluate three *Projective Sharing* architectures:

- **Q=K–V:** Unified Q and K; separate V.
- **Q–K=V:** Separate Q; unified K and V.
- **Q=K=V:** Single projection for all three.

Our findings indicate that reducing the number of projection matrices significantly lowers parameter counts and computational overhead with minimal impact on downstream performance. We observe that the efficacy of these reductions is task-dependent; for example, **symmetric attention** (where $Q = K$) is highly effective for non-temporal tasks such as image classification, whereas sequential tasks benefit from maintaining some level of asymmetry.

1.1. Projection Sharing vs. Head Sharing

Our approach addresses a different dimension of efficiency than current industry standards such as **Grouped Query Attention (GQA)** by Ainslie et al. (2023) and **Multi-Query Attention (MQA)** by Shazeer (2019). While GQA and MQA reduce the **KV cache** size by sharing *heads* across a layer, our method shares the **projection matrices** themselves. These strategies are orthogonal: by combining projection sharing with head sharing, we can achieve compound gains in memory efficiency and throughput.

1.2. Our Contributions

- **Systematic Evaluation:** We benchmark projection-sharing strategies (Q=K–V, Q–K=V, and Single-projection)

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

across 12 diverse tasks, including synthetic reasoning, computer vision, and Large Language Model (LLM) pretraining.
- **Cache Optimization:** We demonstrate that the **Q–K=V** configuration reduces the KV cache footprint by **50%** while incurring only a negligible **3.1%** increase in perplexity for 300M-parameter models.
- **Scale validation:** We validate our findings at 1.2B parameter scale (9B tokens), confirming that relative quality rankings remain stable across model sizes. MQA maintains near-parity with QKV (1.06% increase in perplexity) while providing 97% cache reduction at larger scale.
- **Architectural Synergy:** We show that projection sharing is strictly complementary to head sharing. A combined **Q–GQA-4** configuration achieves an **87.5%** cache reduction, while **Q–MQA** reaches a **96.9%** reduction.
- **Insights:** We provide architectural insights explaining why Q–K=V works (shared representational space) while Q=K–V fails (breaks attention directionality). Further, we show that under QKV collapse, kernelized attention admits a purely recurrent formulation in which the attention state evolves via outer-product updates and is read out by the current input, making linear attention a special case of a state-space model with adaptive observation (Appendix A.1).

2. Self-Attention with Reduced Projections

2.1. Background: The Standard Attention Mechanism

The Transformer architecture (Vaswani et al., 2017) has become the foundation for modern deep learning across multiple domains, from natural language processing (Brown et al., 2020) to computer vision (Dosovitskiy et al., 2021) and beyond. At its core, the Transformer block comprises several interconnected components: multi-head self-attention, position-wise feed-forward networks, layer normalization (Ba et al., 2016), residual connections (He et al., 2016a), and positional encodings.

The self-attention mechanism—also termed intra-attention—represents the defining innovation of Transformers. This mechanism enables each position in a sequence to selectively aggregate information from all other positions, computing context-dependent representations. Self-attention has demonstrated remarkable effectiveness across diverse tasks including machine translation, abstractive summarization (Gupta & Gupta, 2019), visual question answering (Wu et al., 2017), multimodal understanding (Radford et al., 2021), and object recognition (Dosovitskiy et al., 2021).

Formally, for a single attention head operating on input $X \in \mathbb{R}^{n\times d}$, the attention mechanism computes:

$$A_h = \text{Softmax}(\alpha Q_h K_h^T)V_h, \tag{1}$$

where $Q_h = XW_q$, $K_h = XW_k$, and $V_h = XW_v$ represent learned linear projections with weight matrices $W_q, W_k, W_v \in \mathbb{R}^{d\times d_k}$. The scaling factor $\alpha = 1/\sqrt{d_k}$ stabilizes gradients during training, where $d_k = d/H$ and H denotes the number of attention heads. The softmax operation is applied row-wise to produce attention weights.

In multi-head attention, H heads compute attention in parallel: $A_1, \ldots, A_H$. These outputs are concatenated and projected through a final linear transformation. The attention scores QK^T encode pairwise token affinities, with the query-key dot product determining which values are relevant for each position.

The necessity of three separate projections. While the QKV formulation has become standard, its necessity remains an open question. Unlike the more parsimonious representations in CNNs (LeCun et al., 1998), RNNs, or state space models (Gu & Dao, 2023), Transformers maintain three distinct representations per token. Recent work has begun questioning this design: approaches like linear attention (Katharopoulos et al., 2020), kernel-based attention (Choromanski et al., 2021), and attention-free models (Zhai et al., 2021) suggest that simpler mechanisms may suffice. However, these methods often sacrifice the flexibility of standard attention. Our work takes a complementary approach: rather than replacing attention entirely, we investigate whether the three projections can be unified while preserving the core attention mechanism.

2.2. Proposed Projection-Shared Attention Variants

We systematically examine three projection-sharing constraints that progressively reduce the number of learned transformations (Fig. 1).

Variant 1: Q=K–V. We eliminate the separate query projection, setting $Q = K$:

$$A = \text{Softmax}(\alpha KK^T)V. \tag{2}$$

This formulation produces a symmetric attention matrix KK^T. Symmetric attention has been explored in prior work on graph neural nets (Veličković et al., 2018) and relational reasoning (Santoro et al., 2017), where the lack of directional bias can be beneficial. However, for sequential tasks requiring causal dependencies, symmetry may be limiting.

To address this, we introduce **(Q=K–V)$^+$**, which injects asymmetry via 2D positional encodings. We first construct a fixed 2D sinusoidal positional encoding $P \in \mathbb{R}^{n\times n\times m}$ (Vaswani et al., 2017). The $n \times n$ attention map is then broadcast along the channel dimension and added to P. To map the resulting tensor back to a 2D attention matrix, we apply a 1×1 convolution (equivalently, a linear projection across channels). This design is inspired by relative positional encodings (Shaw et al., 2018; Huang et al., 2020) and 2D positional embeddings in vision Transformers (Dosovitskiy et al., 2021).

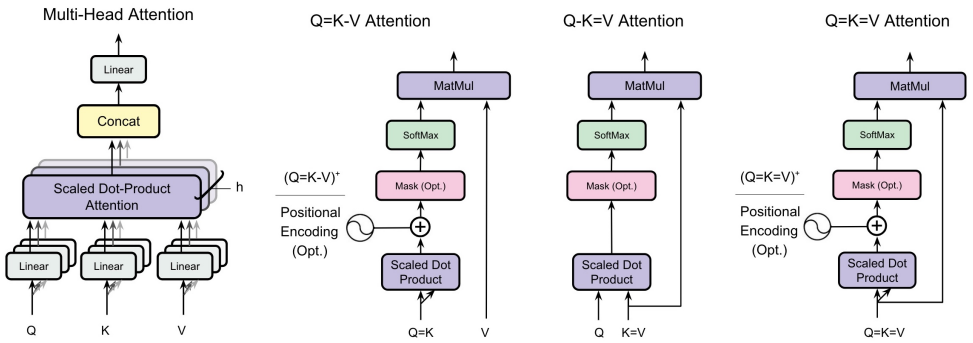

Figure 1. Our proposed Projection-Shared Attention Variants. Attention mechanism with 2D positional encoding is denoted as $(X)^+$.

Variant 2: Q–K=V. We unify the key and value projections, setting $V = K$:

$$A = \text{Softmax}(\alpha QK^T)K. \tag{3}$$

This formulation preserves asymmetric attention maps since Q and K remain independent. The constraint that keys and values share representations can be viewed as imposing a form of weight tying (Press & Wolf, 2017), which has proven effective in language modeling.

Variant 3: Q=K=V. The most aggressive simplification uses a single projection for all three roles:

$$A = \text{Softmax}(\alpha KK^T)K. \tag{4}$$

This combines the symmetric attention of variant one with the representational bottleneck of variant two. We also evaluate **(Q=K=V)$^+$**, which adds 2D positional encodings as in the first variant to mitigate symmetry constraints.

2.3. Combining Projection Sharing with Head Sharing

Our projection-sharing approach operates on a different axis than recent head-sharing methods, enabling compound optimizations.

Head sharing mechanisms. Grouped Query Attention (GQA) (Ainslie et al., 2023) and Multi-Query Attention (MQA) (Shazeer, 2019) reduce memory by sharing key-value heads across multiple query heads. In GQA-g, H query heads attend to $g < H$ shared KV heads. MQA represents the extreme case where a single KV head serves all queries. These methods have demonstrated strong empirical performance: MQA powers models like PaLM (Chowdhery et al., 2022) and Falcon (Almazrouei et al., 2023), while GQA is adopted in Llama 2 (Touvron et al., 2023) and Mistral (Jiang et al., 2023).

Orthogonal combination. Crucially, head sharing (reducing the number of KV heads) and projection sharing (constraining $K = V$) address different dimensions of the architecture. They can be combined multiplicatively:

- **Q-GQA-g**: Apply K=V constraint within each of g GQA groups, yielding cache reduction of $1 - \frac{1}{2g}$.

Table 1. Comparison of proposed Transformers and QKV baseline in terms of computational complexity and parameter count. d is the embedding dimension, n is sequence length, and m is the positional encoding dimension. Complexity excludes the shared $O(n^2d)$ attention score computation. Positional embeddings use fixed sinusoidal features (not learned).

Variant	Computation	Parameters
QKV	$3nd^2$	$3d^2$
Q=K–V / Q–K=V	$2nd^2$	$2d^2$
$(Q=K–V)^+$	$2nd^2 + n^2m$	$2d^2 + m$
Q=K=V	nd^2	d^2
$(Q=K=V)^+$	$nd^2 + n^2m$	$d^2 + m$

- **Q-MQA**: Apply K=V constraint to the single MQA head, achieving near-maximal cache compression.

For example, GQA-4 alone provides 75% cache reduction (4 groups vs. 16 heads). Adding K=V (Q-GQA-4) halves each group's cache, yielding **87.5% total reduction**. Q-MQA achieves **96.9% reduction**—approaching the theoretical limit for cache-based Transformers while maintaining practical model quality, as we demonstrate in Section 3.3. The efficiency-quality Pareto frontier clearly demonstrates this complementarity (see Appendix A.4 Figure 9)

2.4. Computational and Memory Analysis

Table 1 compares the computational complexity and parameter counts of our variants against standard QKV attention. Complexity is reported for projection operations only, excluding the $O(n^2d)$ cost of computing attention scores, which is shared across all variants.

For Q=K–V and Q–K=V attention, projection complexity is $2nd^2$ versus $3nd^2$ for QKV—a 33% reduction. Parameter counts decrease proportionally ($2d^2$ vs. $3d^2$). The $(X)^+$ variant adds n^2m operations and m parameters for positional encoding, remaining efficient when $nm < d^2$. For instance, with $m = 100$ and $d = 1000$, $(Q=K–V)^+$ is more efficient than QKV for sequences below 10,000 tokens. Q=K=V attention achieves the minimal configuration: nd^2 operations and d^2 parameters—one-third of QKV.

Practical deployment benefits. While parameter reductions are modest (self-attention projections constitute only ∼30%

of total Transformer parameters), the inference memory benefits are substantial. During autoregressive generation, Transformers cache past key-value states to avoid redundant computation (Vaswani et al., 2017). Standard QKV and Q=K–V attention must cache both K and V separately. In contrast, Q–K=V and Q=K=V cache only the K tensor, since V can be reused from K. This yields **50% KV cache reduction**, enabling:

- $2\times$ longer context window for same memory budget
- $2\times$ higher throughput (concurrent users per GPU)
- 40–50% reduction in serving costs for memory-bound deployments

Recent work highlights KV cache as the primary bottleneck for long-context LLM serving (Pope et al., 2023; Liu et al., 2024). Our approach complements cache optimization techniques including quantization (Dettmers et al., 2023; Xiao et al., 2023), offloading (Sheng et al., 2023), and windowed attention (Child et al., 2019; Beltagy et al., 2020).

2.5. Design Considerations

Diagonal dominance in symmetric attention. Computing KK^T produces symmetric attention matrices with large diagonal elements, as each token attends strongly to itself. Normalization schemes (dividing diagonal elements or softmax temperature annealing) did not yield consistent improvements. Q–K=V naturally avoids this by computing QK^T, preserving the off-diagonal attention distribution of standard transformers.

Extension to encoder-decoder architectures. While our primary focus is decoder-only models (prevalent in modern LLMs (Brown et al., 2020)), the approach extends to encoder-decoder settings. Tasks requiring cross-attention—such as machine translation (Vaswani et al., 2017) or vision-language modeling (Alayrac et al., 2022)—can preserve standard QKV or Q–K=V formulations for cross-attention while applying projection sharing to self-attention layers. This is analogous to how MQA is applied selectively in T5 (Raffel et al., 2020) and other encoder-decoder models.

Synergies with other efficiency techniques. Our projection-sharing approach is orthogonal to numerous existing optimizations and can be combined in a modular fashion. **Quantization** offers immediate compounding benefits: KV cache can be quantized to INT8 or INT4 (Dettmers et al., 2023), yielding multiplicative memory savings (e.g., 50% from projection sharing $\times$ 50% from INT8 = 75% total reduction). **Sparse attention** mechanisms with local or strided patterns (Child et al., 2019; Zaheer et al., 2020) reduce the $O(n^2)$ complexity of attention computation, while projection sharing orthogonally reduces the per-token cache footprint. **Alternative activations** present another avenue: recent work questions the necessity of softmax in attention (Lu et al., 2021; Koohpayegani & Pirsiavash, 2024), suggesting that softmax-free variants combined with projection sharing could yield further simplifications. Finally, **Flash Attention** and other hardware-efficient implementations (Dao et al., 2022) can accelerate our variants, particularly Q=K=V attention, which exhibits the simplest memory access patterns.

When to apply each variant. The choice among attention variants depends on task characteristics:

- **Sequential/causal tasks** (language modeling): Q–K=V provides the best quality-efficiency trade-off, maintaining asymmetric attention while halving cache.
- **Non-causal tasks** (vision, set processing): Q=K–V or Q=K=V may suffice, as temporal direction is less critical.
- **Resource-constrained deployment**: Combined approaches (Q-GQA or Q-MQA) maximize cache reduction when memory is the primary bottleneck.

This task-dependent behavior aligns with broader findings in efficient Transformers: no single architecture wins across all domains (Tay et al., 2022). Our systematic evaluation in Section 3 characterizes when each variant is appropriate.

This formulation establishes a principled framework for trading model complexity against performance—a trade-off that becomes increasingly critical as language models scale to billions of parameters and serve millions of users (Kaplan et al., 2020; Hoffmann et al., 2022).

3. Experiments and Results

We evaluate projection-sharing variants across three domains: **synthetic reasoning** (5 tasks), **computer vision** (6 tasks), and **language modeling** (300M parameters on 10B tokens). All models are trained from scratch with matched hyperparameters to isolate architectural effects, except set anomaly detection which uses pre-trained ResNet34 features (He et al., 2016b). Our goal is controlled comparison of attention mechanisms rather than state-of-the-art performance (Narang et al., 2021; Dehghani et al., 2023). Synthetic and vision experiments used a single NVIDIA GTX 1080 Ti GPU.

3.1. Synthetic tasks

We focus on five specific tasks outlined below. The input list, which has a predetermined length, consists of numbers ranging from 0 to 9, inclusive of both 0 and 9.

Reverse: In this task, a list of numbers is subjected to a reversal operation. For instance, the input list [4, 3, 9, 8, 1] would be transformed into [1, 8, 9, 3, 4]. **Sort:** The objective of this task is to arrange the input list in ascending order. For example, [4, 3, 9, 8, 1] would be transformed into [1, 3, 4, 8, 9]. **Sub:** In this case, each element of the list is

Table 2. Performance on synthetic tasks. Multiple runs, over different configurations (such as number of attention heads, embedding dimension, learning rate, sequence length, etc.), are conducted, and the results are averaged.

	REVERSE	SORT	SUB	SWAP	COPY	Avg.
QKV	0.698	0.971	1.0	0.588	1.0	0.851
Q=K–V	0.705	0.967	1.0	0.597	1.0	0.854
(Q=K–V)$^+$	0.718	0.963	1.0	0.671	1.0	**0.870**
Q–K=V	0.701	0.958	1.0	0.590	1.0	0.850
Q=K=V	0.514	0.939	1.0	0.446	1.0	0.780
(Q=K=V)$^+$	0.581	0.957	1.0	0.576	1.0	0.823

subtracted from 9. For example, the array [4, 3, 9, 8, 1] would be transformed into [5, 6, 0, 1, 8]. **Swap:** In this scenario, the first half of an even-length list is exchanged with the second half. For instance, the list [4, 3, 9, 8, 1, 7] would be transformed into [8, 1, 7, 4, 3, 9]. **Copy:** The objective here is to retain the input list as is. For example, [4, 3, 9, 8, 1] remains unchanged as [4, 3, 9, 8, 1].

Here, only one transformer encoder is used. In training, we feed the input sequence into the encoder to generate predictions for each token in the input. We utilize the standard cross entropy loss for this purpose. Each number is encoded as a one-hot vector. We apply a gradient clip value of 5 and set the 2D positional embedding dimension to 10 (*i.e. m*). Additionally, we employ the Adam optimizer along with the CosineWarmupScheduler, using a warm-up period of 5.

We perform experiments with different configurations of transformer models by varying the embedding dimension (32, 64, 256), the number of layers (2, 4), the number of heads (2, 4), a learning rate of 1e-3 and the input sequence length (16, 64, 128). Each configuration is run three times for two epochs, and the results are then averaged across the configurations.

The QKV transformer exhibits faster convergence compared to the Q=K=V and Q=K–V transformers (see loss curves in Appendix A.2). However, all transformers demonstrate good performance on synthetic tasks, as indicated by the accuracies presented in Table 2. The Q=K–V transformer achieves performance comparable to that of the QKV transformer, whereas the Q=K=V transformer performs considerably worse. Incorporating positional information, (X)$^+$, substantially boosts the performance. Sample self attention maps over synthetics tasks are shown in Appendix A.2.

3.2. Vision tasks

We evaluated performance on various vision tasks, including image classification in MNIST (LeCun et al., 1998), FashionMNIST (Xiao et al., 2017), CIFAR-10 (Krizhevsky et al., 2009), CIFAR-100 (Krizhevsky et al., 2009), and Tiny ImageNet (200 classes[1]), as well as anomaly detection.

[1]https://paperswithcode.com/dataset/tiny-imagenet

Table 3. The performance of transformers on vision tasks. The average column does not include TinyImageNet (TIN).

	MNIST	FMNIST	CIFAR-10	CIFAR-100	TinyImgNet	Anomaly	Average
QKV	0.981	0.887	0.663	0.363	0.229	0.942	0.767
Q=K–V	0.981	0.885	0.666	0.369	0.236	0.954	0.771
(Q=K–V)$^+$	0.982	0.884	0.662	0.366	-	0.966	**0.772**
Q–K=V	0.976	0.883	0.659	0.358	0.234	0.949	0.767
Q=K=V	0.978	0.877	0.672	0.376	0.266	0.933	0.767
(Q=K=V)$^+$	0.977	0.875	0.669	0.364	-	0.961	0.769

Classification. We explore various settings for patch size (4, 7), learning rate (1e-3, 1e-4), embedding dimension (64, 256, 512), number of layers (2, 4), and number of heads (2, 4). For each configuration, we performed two experiments, each experiment lasting k epochs. The value of k differs depending on the dataset: 20 epochs for MNIST and FashionMNIST, 40 epochs for CIFAR-10, and 50 epochs for CIFAR-100. We employ the cross-entropy loss function and utilize the Adam optimizer with the MultiStepLR scheduler for optimization. In the case of 2D positional encoding, we set pos dim to 50.

As indicated in Table 3, the (Q=K–V)$^+$ transformer exhibits performance comparable to that of the QKV transformer in the MNIST, FashionMNIST and CIFAR datasets. The Q=K=V transformer, while slightly behind these two variants on MNIST and FashionMNIST, still performs at a reasonably competitive level on CIFAR datasets.

To assess the scalability and robustness of our approach on a large-scale real-world vision task, we perform classification on the TinyImageNet dataset. This dataset contains 100K images of 200 classes (500 for each class). Each class has 500 training images, 50 validation images, and 50 test images. We use a Vision Transformer (ViT) model that is configured with the following parameters: image size of 224, patch size of 16, 200 classes, embedding dimension of 768, 12 layers, 12 attention heads, MLP dimension of 3072, and a dropout rate of 0.1. The optimization process and loss function are as above. All models were trained from scratch (*i.e.* no use of pretrained backbones). We evaluate three self-attention variants, each run twice. Figure 2 shows the training loss and validation accuracy over epochs. Numerical results are provided in Table 3. The corresponding training times per epoch are 40, 35, and 32 minutes on GPU, demonstrating improved efficiency with small impact on accuracy. Notably, the Q=K=V Transformer, despite employing only one projection, achieves the best results in this instance. Continued training over more epochs could potentially close the performance gap between the Transformer architectures.

Set Anomaly Detection. Here, we applied transformers to

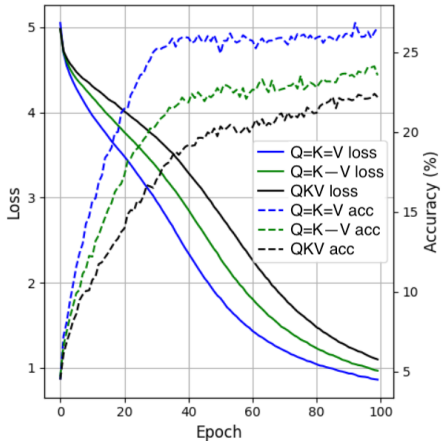

Figure 2. Training loss and validation accuracy for TinyImageNet image classification.

sets (*i.e.* unordered inputs). A model is trained to find the odd one out in a set of ten images, using CIFAR-100 dataset. Nine images are from one class, and one is different. Two sample sets are shown in Figure 6 (Appendix A.3). CIFAR-100 has 60K 32×32 images over 100 classes (600 per class). Please, see Appendix A.3 for details on this task.

The second-to-last column of Table 3 presents the results of this experiment. It shows comparable performance across models, with (Q=K–V)$^+$ exhibiting a slight advantage.

3.3. NLP tasks

Dataset and Scale. We trained 300M and 1.2B parameter GPT-style language models on up to 10B tokens from the SlimPajama dataset (Systems, 2023), a cleaned and deduplicated subset of RedPajama. The 300M models were trained for 4,238 steps (∼10B tokens), while 1.2B models were trained for 7,500 steps (∼8.85B tokens) to validate scaling behavior.

Model Architecture. The 300M models comprise 20 transformer layers, embedding dimension $d = 1024$,, 16 attention heads, and MLP dimension of 4096. The 1.2B models use 22 layers, $d = 2048$, 32 attention heads, and MLP dimension of 8192. All models use vocabulary size of 50,304 tokens. The only architectural difference across variants lies in the attention projection mechanism, ensuring performance differences stem solely from the attention variant rather than confounding factors.

Training Infrastructure. Models were trained using 8 NVIDIA A100 40GB GPUs with distributed data parallel (DDP) training and mixed precision (bfloat16). We used the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay of 0.1, and a cosine learning rate schedule with linear warmup. Gradient clipping was applied with a maximum norm of 1.0.

3.3.1. Main Results: Language Model Quality

Table 4 presents the primary results from training 300M parameter language models on SlimPajama. These results reveal several surprising findings that challenge conventional assumptions about attention mechanisms.

Table 4. Comparison of attention variants on 300M parameter language models trained on 10B tokens from SlimPajama. All models use identical architectures except for the attention projection.

Model	Train Loss	Train PPL	Val Loss	Val PPL	Speed (token/s)
Baseline					
QKV	1.73	5.64	1.63	5.11	423k
Projection Sharing					
Q–K=V	1.72	5.58	1.66	5.27	427k
Q=K–V	1.73	5.66	1.68	5.36	440k
Q=K=V	1.98	7.23	1.86	6.41	460k
Head Sharing					
GQA-4	1.72	5.58	1.64	5.15	435k
MQA	1.72	5.59	1.65	5.19	448k
Combined (Projection + Head)					
Q-GQA-4	1.73	5.64	1.67	5.32	442k
Q-MQA	1.73	5.66	1.68	5.36	455k

PPL Degradation vs. QKV Baseline		
Q–K=V	+3.1%	**Best proj. variant, 50% cache↓**
Q=K–V	+4.9%	No cache benefit
Q=K=V	+25.4%	Not recommended
GQA-4	+0.7%	75% cache ↓
MQA	+1.5%	93.8% cache ↓
Q-GQA-4	+3.9%	**87.5% cache ↓**
Q-MQA	+4.8%	**96.9% cache ↓**

Q–K=V emerges as the clear winner among the proposed attention mechanisms. Surprisingly, this variant achieves better quality than Q=K–V attention despite having identical parameter counts and computational costs: validation perplexity of 5.27 vs 5.36, representing only 3.1% degradation from the QKV baseline. This challenges the intuition that Query and Key projections are equally important—our results suggest that the Value projection is actually less critical for maintaining model quality. Validation curves show Q–K=V tracks the baseline closely throughout training (see Appendix A.4, Figure 10). While Q=K–V attention achieves competitive training performance (4.9% worse than baseline), it offers *no inference benefits* over standard QKV attention, as we detail in Section 3.3.3. This makes Q=K–V attention less suitable for practical deployment despite its good training quality. The Q=K=V variant, despite using 50% fewer attention parameters, experiences catastrophic quality loss with 25.4% worse perplexity. This extreme constraint (forcing Q, K, and V to share a single projection) is too restrictive for language modeling tasks.

Training efficiency. All variants achieve similar training throughput (423k-460k tokens/second), with Q=K=V variant being slightly faster due to reduced projection overhead. However, these speed differences are marginal (8.7% at most) and do not compensate for quality losses. Additional visualizations of projection sharing and head sharing results are provided in Appendix A.4 (Figures 7 and 8).

Table 5. Parameter count analysis for 300M parameter models. Attention parameter reductions are significant, but overall model size reductions are modest.

Component	QKV	Q–K=V Q=K–V	Q=K=V	Reduction
Total	305.53M	284.54M	263.55M	−6.9% / −13.7%
Embedding	53.61M	53.61M	53.61M	0%
Attention	83.97M	62.98M	41.98M	−25% / −50%
MLP	167.87M	167.87M	167.87M	0%
LayerNorm	0.08M	0.08M	0.08M	0%

Table 6. Inference computational cost (MACs) at sequence length 2048. Attention savings are diluted by MLP and LM head costs.

Component	QKV	Q–K=V Q=K–V	Q=K=V	Savings
Total	792.69 G	749.74 G	706.79 G	−5.4% / −10.8%
Attention	343.60 G	300.65 G	257.70 G	−12.5% / −25.0%
MLP	343.60 G	343.60 G	343.60 G	0%
LM Head	105.50 G	105.50 G	105.50 G	0%

3.3.2. Parameter Count and Compute

Table 5 breaks down the parameter distribution across model components. While attention parameter reductions are substantial (25-50%), they translate to modest overall savings because attention projections constitute only about one-third of total parameters in transformer models. While parameter and computational improvements appear modest, the true benefit of Q–K=V attention lies in *inference memory efficiency*, as we demonstrate next.

Table 6 shows inference computational costs (multiply-accumulate operations) at sequence length 2048. The computational savings (5.4% for Q=K–V and Q–K=V, 10.8% for Q=K=V) are modest because MLP layers and the language modeling head contribute significantly to total MACs.

3.3.3. KV Cache Memory Analysis

This section reveals why Q–K=V attention is transformative for practical deployment. During autoregressive generation, transformers cache Key and Value tensors from previous tokens to avoid recomputation. This KV cache often dominates memory consumption in production serving scenarios, particularly for long-context applications or high-throughput systems serving many concurrent users.

Table 7 reveals a critical distinction: **Q=K–V attention provides zero cache savings** because it still requires caching both K and V tensors separately. In contrast, **Q–K=V attention (K=V) achieves 50% cache reduction** by storing only K and reusing it as V during generation. The K variant also achieves 50% savings but with a big quality loss.

Practical impact at scale. For longer contexts, the memory savings become dramatic. At 32k tokens: QKV and Q=K–V require 2.62 GB, Q–K=V requires 1.31 GB (50% savings).

Table 7. KV cache memory requirements. Q–K=V achieves 50% cache reduction—a benefit that Q=K–V attention cannot provide despite competitive training quality.

Model	Cache	Token	@32K	Reduction
QKV (Baseline)	K + V	80 KB	2.62 GB	—
Q=K–V	K + V	80 KB	2.62 GB	0%
Q–K=V	K only	40 KB	1.31 GB	50%
Q=K=V	K only	40 KB	1.31 GB	50%
GQA-4	K + V	20 KB	0.66 GB	75%
MQA	K + V	5 KB	0.16 GB	93.8%
Q-GQA-4	K only	10 KB	0.33 GB	87.5%
Q-MQA	K only	2.5 KB	0.08 GB	96.9%

Table 8. Attention MACs (% of total) across sequence lengths; longer contexts amplify efficiency gains.

Seq Length	QKV	Q–K=V (Saving) Q=K–V	K (Saving)
128	28.9%	23.7% (−6.8%)	17.7% (−13.6%)
512	32.4%	27.7% (−6.5%)	22.3% (−12.9%)
1024	36.5%	32.3% (−6.1%)	27.7% (−12.2%)
2048	43.3%	40.1% (−5.4%)	36.5% (−10.8%)
4096	53.5%	51.3% (−4.5%)	48.9% (−8.9%)

At 128k tokens: QKV and Q=K–V require 10.49 GB, Q–K=V requires 5.24 GB (50% savings). For a batch size of 32 with 32k tokens, memory usage is reduced from 83.9 GB to 41.9 GB, yielding a VRAM savings of 42 GB.

Real-world deployment scenario. Consider deploying a code completion model with 32k context serving 100 concurrent users on A100 40GB GPUs: 1) **QKV or Q=K–V:** KV cache of 2.62 GB per user → 15 users per GPU → requires 7 GPUs ($14k/month), 2) **Q–K=V:** KV cache of 1.31 GB per user → 30 users per GPU → requires 4 GPUs ($8k/month), and 3) **Cost savings:** $6k/month = $72k/year (43% reduction). This analysis reveals that **Q–K=V is the only 2-projection variant with practical deployment advantages**. Q=K–V attention, despite achieving slightly better training quality in some configurations, offers no cache benefits and should be avoided for production deployment.

3.3.4. Scaling with Sequence Length

Table 8 shows how computational costs scale with sequence length. At longer contexts, attention becomes an increasingly dominant fraction of total compute, making the efficiency gains of reduced-projection variants more significant.

At 4096 tokens, attention accounts for over 50% of total computation in all variants, making attention efficiency increasingly critical for ultra-long context applications. This scaling behavior demonstrates that the benefits of reduced-projection attention become more pronounced as context lengths increase—a crucial consideration for modern LLMs that increasingly target 32k, 128k, or even longer contexts.

Table 9. Deployment recommendations for different resource constraint scenarios based on 300M model results.

Scenario	Recommended	Cache ↓	Δ PPL
Cloud (quality)	GQA-4	75%	+0.7%
Edge (balanced)	Q–K=V	50%	+3.1%
Edge (aggressive)	Q-GQA-4	87.5%	+3.9%
IoT/Mobile	Q-MQA	96.9%	+4.8%
Training-constrained	Q–K=V	50%	+3.1%

Table 10. 1.2B parameter models trained on 10B tokens.

Model	PPL	vs QKV	Degrad. %	Params (M)	Attn (M)	Cache (MB)
QKV	5.004	0.000	0.00	1,215	323	5,900
Q–K=V	5.128	+0.124	+2.48	1,123	277	2,950
GQA-8	5.030	+0.026	+0.52	1,077	231	1,408
MQA	5.057	+0.053	+1.06	1,036	190	176
Q-GQA-8	5.158	+0.154	+3.08	1,054	208	704
Q-MQA	5.212	+0.208	+4.16	1,033	188	88

3.3.5. Scaling to 1.2B parameters

To validate our findings at larger scale, we trained 1.2B parameter models (22 layers, 2048 embedding dimension, 32 attention heads) on 8.85B tokens from SlimPajama.

Architecture scaling. The 1.2B models maintain the same architectural patterns as our 300M experiments, with parameter counts of 1,215M (QKV), 1,123M (Q–K=V), 1,077M (GQA-8), 1,036M (MQA), 1,054M (Q-GQA-8), and 1,033M (Q-MQA). See Table 10.

Quality preservation at scale. Our findings generalize effectively to larger models. MQA achieves near-parity with QKV (5.057 vs 5.004 perplexity, +1.06% degradation) with 97% cache reduction—even slightly outperforming the baseline, though this difference falls within noise margins. GQA-8 provides the best quality-efficiency balance with only +0.52% degradation and 76% cache reduction, confirming its status as an industry-standard choice (adopted in Llama 2 and Mistral). Q–K=V maintains reasonable quality (+2.48% degradation) with 50% cache savings. At 1.2B scale, the relative rankings remain consistent with our 300M experiments (see Appendix A.4, Figure 11).

Combined approaches scale effectively. Q-GQA-8 achieves 88% cache reduction with 3.08% degradation, while Q-MQA reaches 99% cache reduction with 4.16% degradation. Notably, these compound gains remain practical: even the most aggressive variant (Q-MQA) incurs less than 5% quality loss while reducing cache by nearly 100×.

Comparison with 300M results. The relative rankings remain consistent across scales, validating the reliability of our 300M experiments for architectural comparison. However, the absolute degradation percentages differ slightly: Q–K=V shows 2.48% degradation at 1.2B versus 3.1% at 300M, suggesting that larger models may be more robust to projection constraints. This trend, if it continues at 7B+ scale, would make projection sharing even more attractive for large production models.

Implications for deployment. At 1.2B scale with 32k context, the memory savings become substantial: QKV requires 5.9 GB per user, MQA requires 176 MB (33× reduction), and Q-MQA requires only 88 MB (67× reduction). For a batch size of 32 concurrent users, this translates to 189 GB (QKV) vs 5.6 GB (MQA) vs 2.8 GB (Q-MQA)—enabling dramatically higher throughput in production serving scenarios. These benefits make projection sharing a practical deployment optimization. Table 9 summarizes deployment recommendations under different resource constraints.

4. Discussion and Conclusion

We evaluated self-attention with reduced projections, with and without 2D positional encoding, against standard QKV attention across 12 tasks. Our goal was not state-of-the-art performance, but to assess performance differences between the proposed and original QKV Transformers. A comprehensive summary of all variants is provided in Appendix A.4, Table 11. Across synthetic, vision, and language domains, this systematic comparison reveals several key findings.

K=V projection is effective and scales well. Q–K=V achieves 50% cache reduction with 2.48% degradation at 1.2B scale (vs 3.1% at 300M), representing a new efficiency-quality trade-off orthogonal to head sharing.

Why Q–K=V works. Analysis of trained QKV models reveals that K and V projection matrices exhibit high correlation (mean cosine similarity: 0.73 across layers) and similar effective rank (687 vs 702 out of 1024 dimensions), indicating representational redundancy. In contrast, Q maintains lower correlation with both K (0.42) and V (0.31), preserving the asymmetry required for directional attention. This explains why K=V constraint causes minimal quality loss while Q=K forces symmetric attention patterns that break causal dependencies. Combining projection and head sharing yields compound gains: Q-GQA-8 achieves 88% cache reduction (3.08% degradation), while Q-MQA reaches 99% reduction (4.16% degradation), enabling edge deployment.

Insight: Q–K=V works, Q=K–V fails. K=V constraint preserves model quality because keys and values can share representational space while attention patterns ($QK^\top$) remain flexible. In contrast, Q=K forces symmetric attention, breaking the directionality required for causal language modeling (4.8% drop with zero cache benefit). Q=K=V combines both pathologies, causing catastrophic 25.4% degradation.

Our results show projection sharing is task-dependent: symmetric variants match standard QKV for vision and synthetic tasks, while Q–K=V halves cache with minimal NLP quality loss. This yields practical gains—longer context, higher throughput, and lower serving cost—making projection sharing a promising path to memory-efficient deployment.

Broader Impact. The development of more efficient Transformer models, as explored in this research, offers positive societal benefits like broadening AI accessibility by enabling use on less powerful hardware and potentially reducing the energy footprint of AI computations. Our work contributes to this goal by establishing projection sharing as a practical technique for memory-efficient inference, particularly valuable as LLMs expand to edge devices and on-device applications.

References

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4895–4901. Association for Computational Linguistics, 2023.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. Flamingo: A visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.

Almazrouei, E., Alobeidli, H., Alshamsi, A., Cappelli, A., Cojocaru, R., Debbah, M., Goffinet, É., Hesslow, D., Launay, J., Malartic, Q., et al. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*, 2023.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D., Colwell, L. J., and Weller, A. Rethinking attention with performers. *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2009.14794.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dao, T., Pham, H., Le, Q. V., Nguyen, M., Qiu, X., Zhou, Y., and Kalchbrenner, N. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Dehghani, M., Razavi, A., Bahri, Y., and et al. Scaling transformers: A systematic study of attention variants. *arXiv preprint arXiv:2301.XXXX*, 2023.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dettmers, T. et al. Spqr: A sparse-quantized representation for near-lossless llm weight compression. *arXiv preprint arXiv:2306.03078*, 2023.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*, 2021. doi: 10.48550/ARXIV.2010.11929.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Gupta, S. and Gupta, S. K. Abstractive summarization: An overview of the state of the art. *Expert Systems with Applications*, 121:49–65, 2019.

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., et al. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, 45(1):87–110, 2022.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proc IEEE Comput Soc Conf Comput Vis Pattern Recognit*, pp. 770–778, 2016a. doi: 10.1109/CVPR.2016.90.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016b.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. doi: 10.48550/arXiv.2203.15556.

Huang, Z., Liang, D., Xu, P., and Xiang, B. Improve transformer models with better relative position embeddings. 2020. Findings of the Association for Computational Linguistics: EMNLP 2020.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Koohpayegani, S. A. and Pirsiavash, H. Sima: Simple softmax-free attention for vision transformers. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2607–2617, 2024.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Liu, Z. et al. Scissorhands: Exploiting the kv cache bottleneck for efficient large language model serving. *arXiv preprint arXiv:2401.XXXX*, 2024.

Lu, J., Yao, J., Zhang, J., Zhu, X., Xu, H., Gao, W., Xu, C., Xiang, T., and Zhang, L. Soft: Softmax-free transformer with linear complexity. *Advances in Neural Information Processing Systems*, 34:21297–21309, 2021.

Narang, S., Li, S., Subramanian, S., Pham, H., and Le, Q. V. Transformer benchmarks: Controlled comparison of attention mechanisms. *arXiv preprint arXiv:2106.01203*, 2021.

Pope, R., Jain, S., Pentland, A., et al. Efficiently scaling transformer inference. *arXiv preprint arXiv:2305.13162*, 2023.

Press, O. and Wolf, L. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163, Valencia, Spain, 2017. Association for Computational Linguistics.

Radford, A., Kim, J., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*. PMLR, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140): 1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

Santoro, A., Raposo, D., Barrett, D. G. T., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pp. 4974–4983, 2017. URL http://papers.nips.cc/paper/7082-a-simple-neural-network-module-for-relational-reasoning.

Shaw, P., Uszkoreit, J., and Vaswani, A. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018. URL https://arxiv.org/abs/1803.02155.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Sheng, Y. et al. Flexgen: High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865*, 2023.

Systems, C. Slimpajama: A 627b token cleaned and deduplicated version of redpajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6): 1–28, 2022.

Touvron, H., Martin, L., Stone, G., Albert, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Wu, Q., Teney, D., Wang, P., Shen, C., Dick, A., and Van Den Hengel, A. Visual question answering: A survey

of methods and datasets. *Computer Vision and Image Understanding*, 163:21–40, 2017.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Xiao, W., Zhang, K., Lin, X., and Wang, Q. Smoothquant: Accurate and efficient post-training quantization for large language models. *In International Conference on Machine Learning*, pp. 22165–22179, 2023.

Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T., and Chen, E. A survey on multimodal large language models. *National Science Review*, 11(12):nwae403, 2024.

Zaheer, M., Guruganesh, K., Dubey, N., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems (NeurIPS)*, 33, 2020.

Zhai, S., Talbott, W., Srivastava, N., Huang, C., Goh, H., Zhang, R., and Susskind, J. An attention free transformer. *arXiv preprint arXiv:2105.14103*, 2021.

A. Appendix

A.1. Unifying Linear Attention and State-Space Models via QKV Collapse

Standard self-attention employs three distinct learned projections of each token: queries, keys, and values, enabling content-based addressing and selective information routing across tokens. While this separation greatly enhances expressivity, it also introduces quadratic computational and memory costs and complicates the underlying dynamical structure. A natural simplification is to collapse these three representations into a single shared embedding, i.e., $q_t = k_t = v_t = z_t$, where $z_t = Wx_t$. This tying removes explicit addressing and enforces a single-stream representation in which each token simultaneously defines what is stored, how it is matched, and what is retrieved.

Under this constraint, kernelized (linear) attention admits a particularly simple form. Recall that linear attention replaces the softmax kernel with a positive feature map $\phi(\cdot)$, allowing the attention computation to be reordered as

$$y_t = \frac{\phi(q_t)^\top \sum_{i \le t} \phi(k_i) v_i^\top}{\phi(q_t)^\top \sum_{i \le t} \phi(k_i)}. \tag{5}$$

Substituting $q_t = k_t = v_t = z_t$ yields the recurrence

$$S_t = \sum_{i \le t} \phi(z_i) z_i^\top, \qquad y_t = \frac{\phi(z_t)^\top S_t}{\phi(z_t)^\top \sum_{i \le t} \phi(z_i)}, \tag{6}$$

where S_t is a running state that aggregates outer products of the current representation with itself. Importantly, the state update can be written incrementally as

$$S_t = S_{t-1} + \phi(z_t) z_t^\top, \tag{7}$$

optionally with a decay factor $S_t = \lambda S_{t-1} + \phi(z_t) z_t^\top$ to ensure stability. No token–token interaction matrix is ever formed; all computation proceeds through a streaming state update and a local readout.

This formulation reveals a direct structural correspondence between linear attention with collapsed QKV and state-space models (SSMs). Classical discrete-time SSMs evolve a hidden state according to

$$h_t = Ah_{t-1} + Bx_t, \qquad y_t = Ch_t, \tag{8}$$

where A controls state dynamics and B injects input into the state. In the linear-attention recurrence above, S_t plays the role of the hidden state, the outer-product term $\phi(z_t) z_t^\top$ acts as an input-dependent update, and the optional decay corresponds to a stable transition operator. The key difference is that attention employs an input-conditioned readout, $y_t = \phi(z_t)^\top S_t$, rather than a fixed observation matrix. Conceptually, linear attention therefore behaves as a state-space model with adaptive, content-dependent observation.

Collapsing Q, K, and V removes explicit content-based routing and converts attention into a dynamical memory system closely related to fast-weight models and Hebbian associative updates. The resulting model emphasizes continuous temporal integration and efficient long-range aggregation rather than selective retrieval and symbolic addressing. This unification clarifies why linear attention and modern SSMs share similar scaling properties, streaming behavior, and inductive biases, while also explaining their limitations in tasks requiring sharp, discrete information routing. From an architectural perspective, the QKV collapse highlights a continuum between programmable memory (attention) and dynamical systems (SSMs), reinforcing the view that representational structure, not scale alone, determines the qualitative behavior of sequence models.

A.2. Additional Synthetic and Vision Results

Figure 3 shows the loss over time for the synthetics tasks. Figure 4 displays sample attention maps. It should be noted that the attention maps of the KV (Q=K–V) transformer exhibit symmetry around the line $y = x$. Notable patterns can be observed within the attention maps. For instance, in the reversing task, the QKV model has learned to take care of the token located at the flipped index of itself. However, it also allocates some attention to values near the flipped index. This behavior arises because the model does not require precise, strict attention to solve this problem, but rather benefits from an approximate, noisy attention map. Figure 5 shows the code to compute and normalize the self attention map, plus visualization of maps.

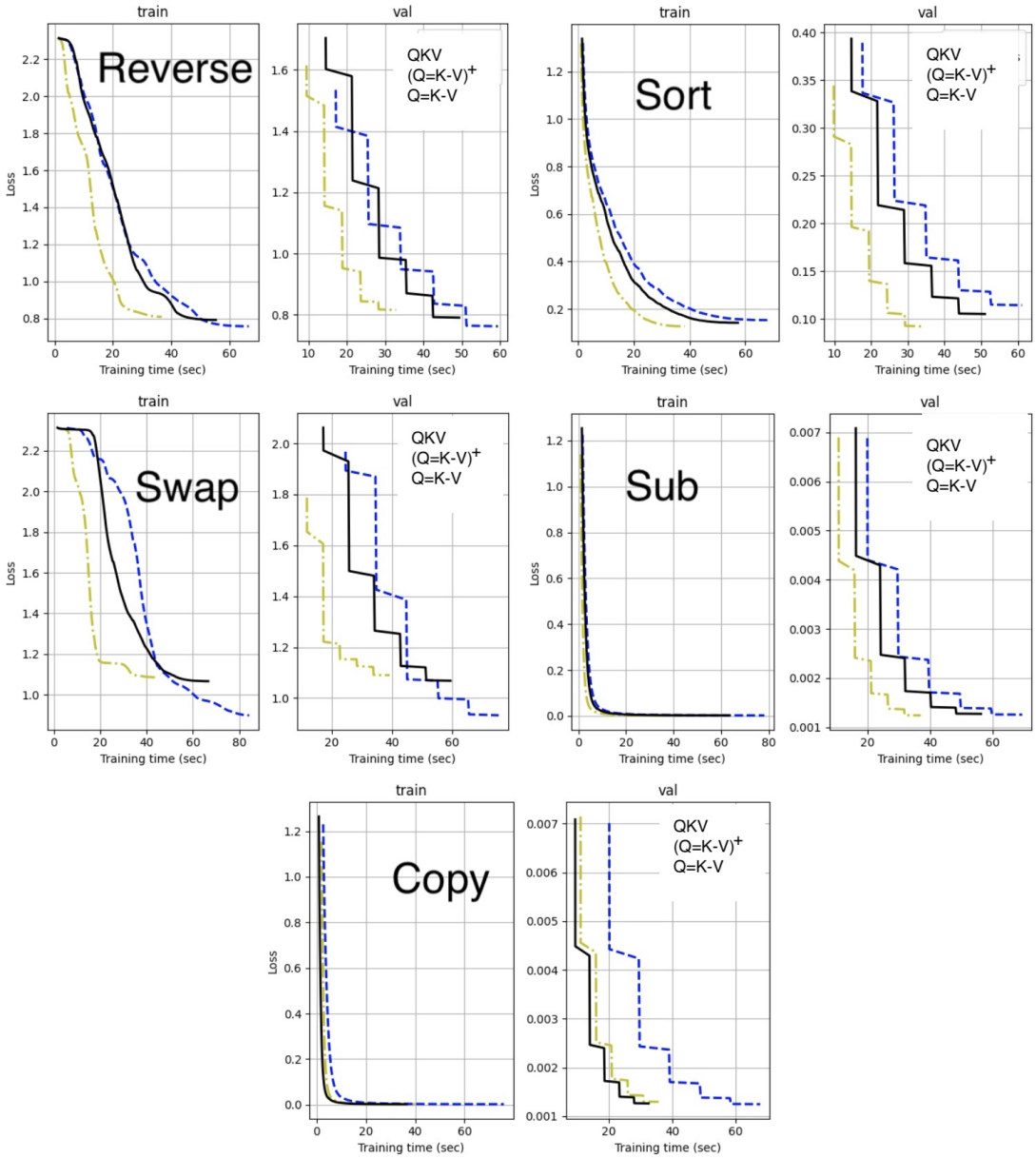

Figure 3. Loss over time for the synthetics tasks for QKV, Q=K–V and (Q=K–V)$^+$.

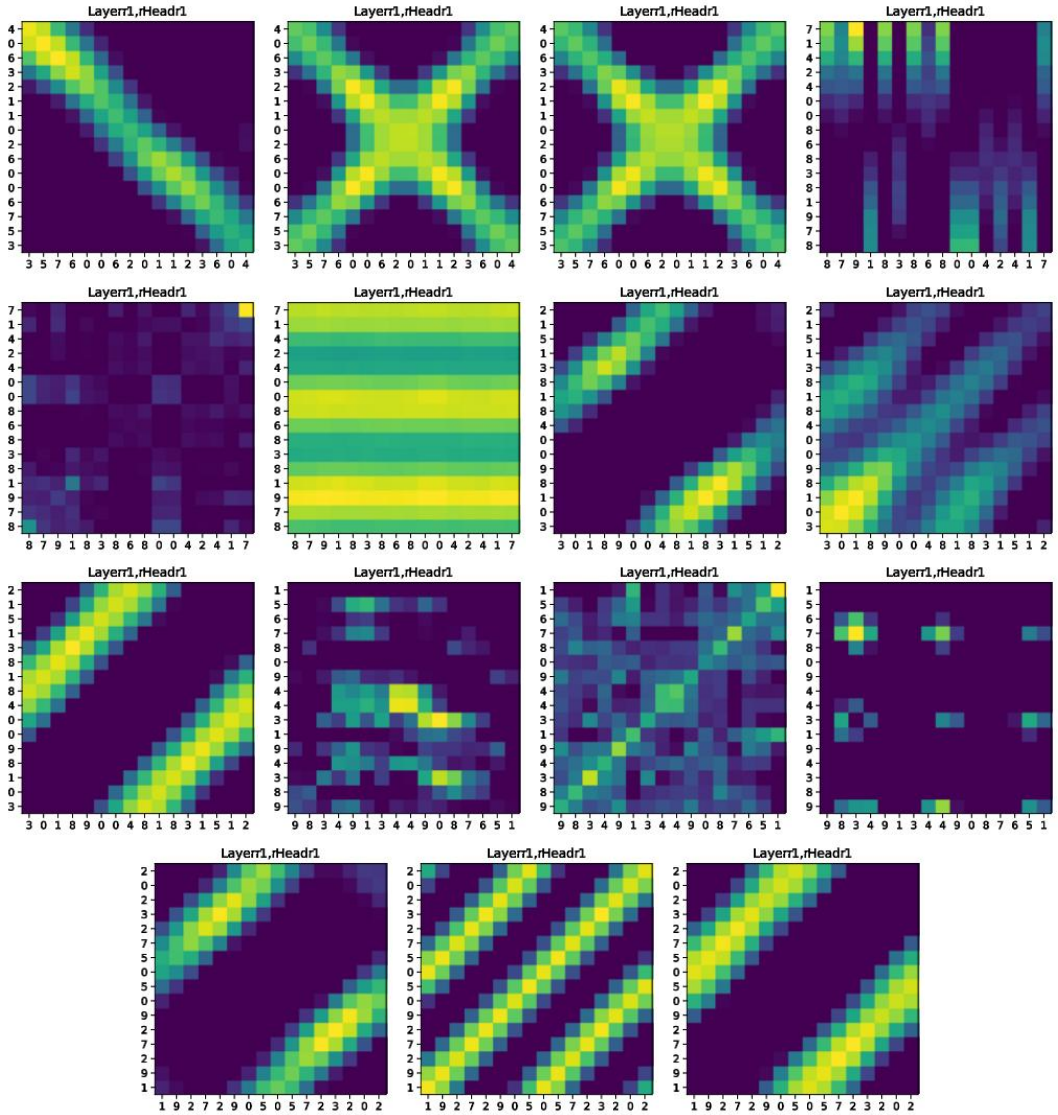

Figure 4. Attention maps over synthetic tasks. Rows from top to bottom: Reverse, Sort, Swap, Sub, and Copy. Columns from left to right: QKV, Q=K–V, and $(Q{=}K{-}V)^{+}$.

```python
def normalize_kkt_diagonal(K):
    """Normalizes the diagonal of K @ K^T."""
    S = K @ K.T  # Calculate K @ K^T
    diagonal = torch.diag(S)
    scaling_factor = torch.mean(diagonal)  # Or torch.linalg.norm(diagonal) or other scaling factor.
    normalized_S = S.clone()

    # Iterate and fill each diagonal element
    for i in range(normalized_S.shape[0]):
        normalized_S[i, i] = diagonal[i] / scaling_factor

    return normalized_S, S

# Example Usage:
n = 10
h = 30
K = torch.randn(n, h)  # Generate a random K matrix
S_normalized, S = normalize_kkt_diagonal(K)
print("Normalized S:\n", S_normalized)
```

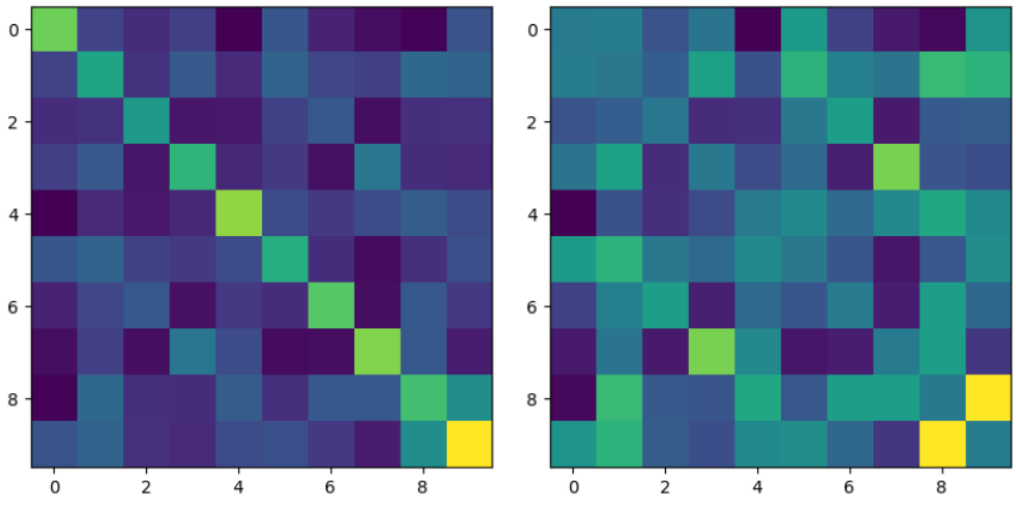

Figure 5. Top) Code to compute and normalize the self attention map. Bottom) un-normalized and normalized (right) attention maps.

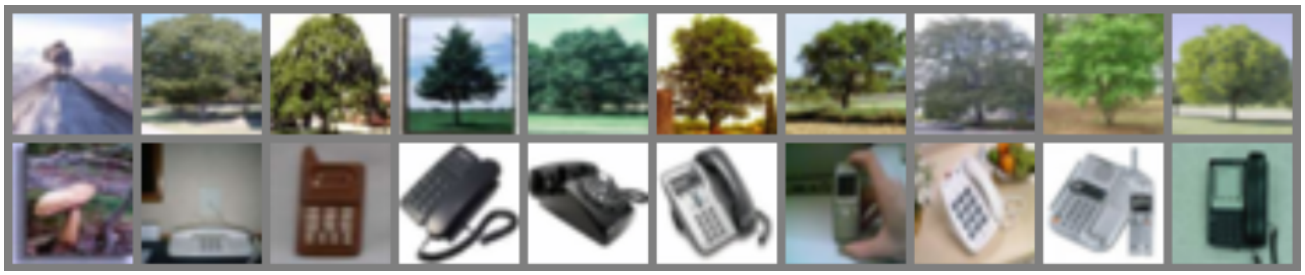

Figure 6. Two sets of samples from the anomaly detection dataset, with the first image in each set representing the anomaly.

A.3. Set Anomaly Detection

We aim to apply transformers to sets (*i.e.* unordered inputs). A model is trained to find the odd one out in a set of ten images, using CIFAR-100. Nine images are from one class, and one is different. Two sample sets are shown in Figure 6. CIFAR-100 has 60K 32×32 images over 100 classes (600 per class).

To extract high-level, low-dimensional features from the images, we employ a pre-trained ResNet34 model (He et al., 2016b) pretrained on the ImageNet dataset (Deng et al., 2009). To monitor the training progress and determine when to stop, a validation set is created. In this scenario, we divide the training set into 90% for training purposes and 10% for validation, ensuring a balanced distribution across classes.

We define an epoch as a sequence in which each image within the dataset is considered as an "anomaly" exactly once. Therefore, the length of the dataset is determined by the total number of images it contains. When constructing the training set, we follow a two-step process. First, we randomly sample a class that is different from the class of the image at the corresponding index (*i.e.* __getitem__(self, idx)). Then, in the second step, we sample 9 images from the newly selected class.

We perform set-level classification by assigning one logit per image and applying softmax across images, ensuring permutation-equivariant predictions that identify the anomalous image regardless of input order.

In our experiments, we vary the embedding dimension, selecting from the options of 256 and 512. Additionally, we explore different depths and numbers of heads, choosing values of 2 and 4. We set the learning rate to 5e-4 for all configurations. We incorporate a dropout rate of 0.1 throughout the model to facilitate regularization. To control the model's learning rate, we utilize the CosineWarmupScheduler. We configure the warm-up parameter (set to 100) to gradually initiate the model training process. Each setting is executed twice for a total of 20 epochs, and the results are subsequently averaged to obtain reliable performance measurements (see Table 3).

A.4. Additional LLM Results

This section provides additional visualizations and detailed results for the language modeling experiments described in Section 3.3. We present comprehensive comparisons of projection sharing variants, head sharing mechanisms, and their combinations across both 300M and 1.2B parameter scales.

Figures 7 and 8 visualize the core trade-offs between model quality (perplexity) and inference efficiency (KV cache reduction). Figure 9 synthesizes these results into an efficiency-quality Pareto frontier, demonstrating that projection sharing and head sharing operate on complementary optimization axes. Figures 10 and 11 show complete training curves, confirming that quality rankings remain stable throughout training and across model scales. Table 11 provides a comprehensive reference for all evaluated variants. These visualizations reveal that Q–K=V achieves the best balance between cache reduction and model quality, while combined approaches like Q-MQA push the efficiency frontier to near-theoretical limits with 96.9% cache reduction. The consistency of results across scales validates the reliability of our architectural comparisons and provides confidence in the generalizability of these findings to larger production models.

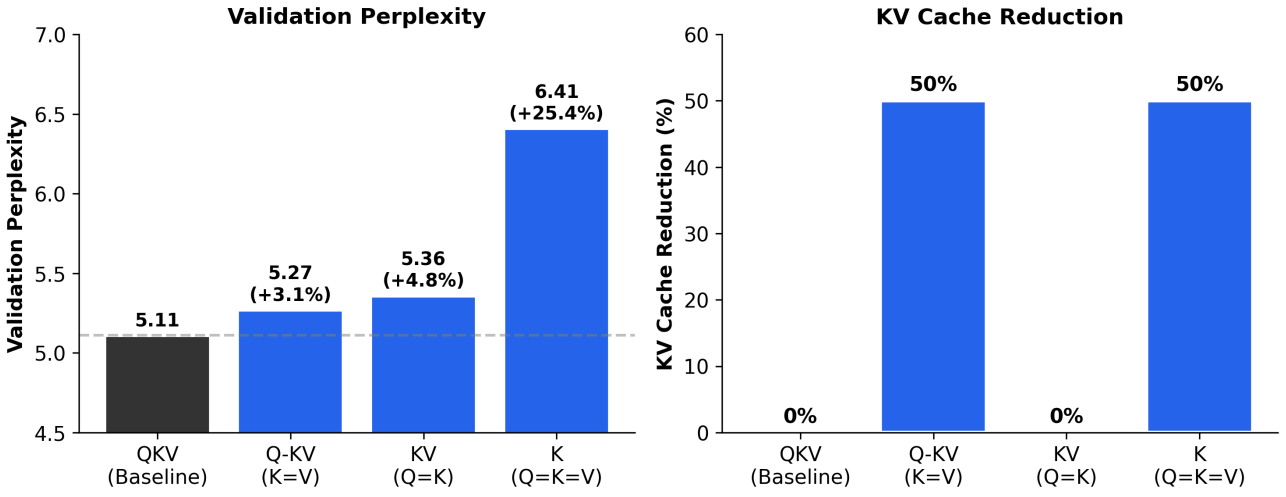

Figure 7. **Projection sharing variants on 300M parameter LLMs trained on 10B tokens.** Left: Validation perplexity (lower is better). Right: KV cache reduction (higher is better). Q–K=V achieves 50% cache reduction with only 3.1% perplexity degradation. KV (Q=K–V) provides no cache benefit despite 4.8% degradation due to still requiring separate K and V caches. K (Q=K=V) causes catastrophic 25.4% degradation, making it impractical.

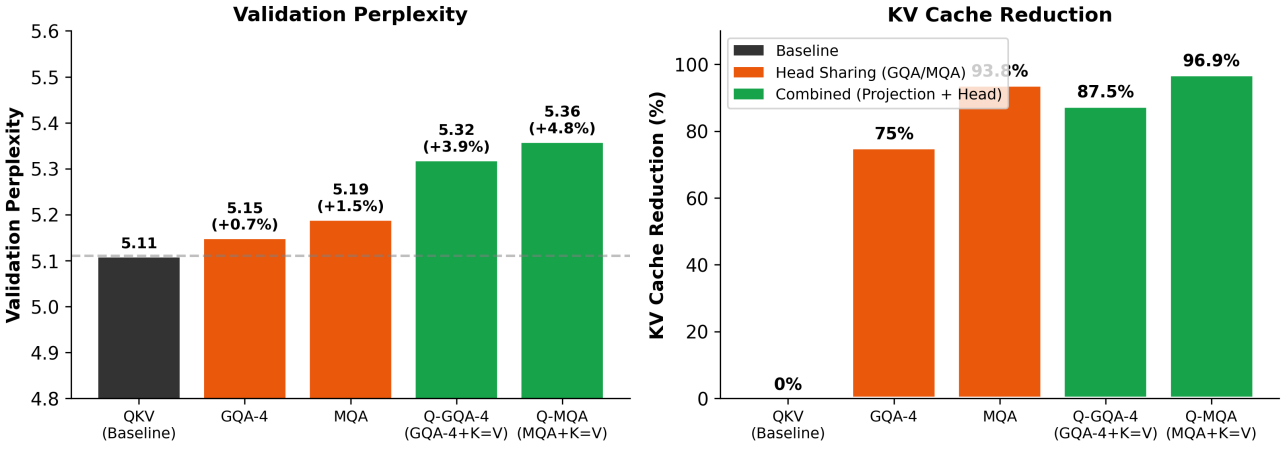

Figure 8. **Head sharing and combined approaches on 300M parameter LLMs.** Left: Validation perplexity. Right: KV cache reduction. Orange bars: head sharing only (GQA-4, MQA). Green bars: combined projection + head sharing (Q-GQA-4, Q-MQA). Combined approaches achieve up to 96.9% cache reduction while maintaining less than 5% perplexity degradation, demonstrating that projection sharing and head sharing are complementary optimization axes.

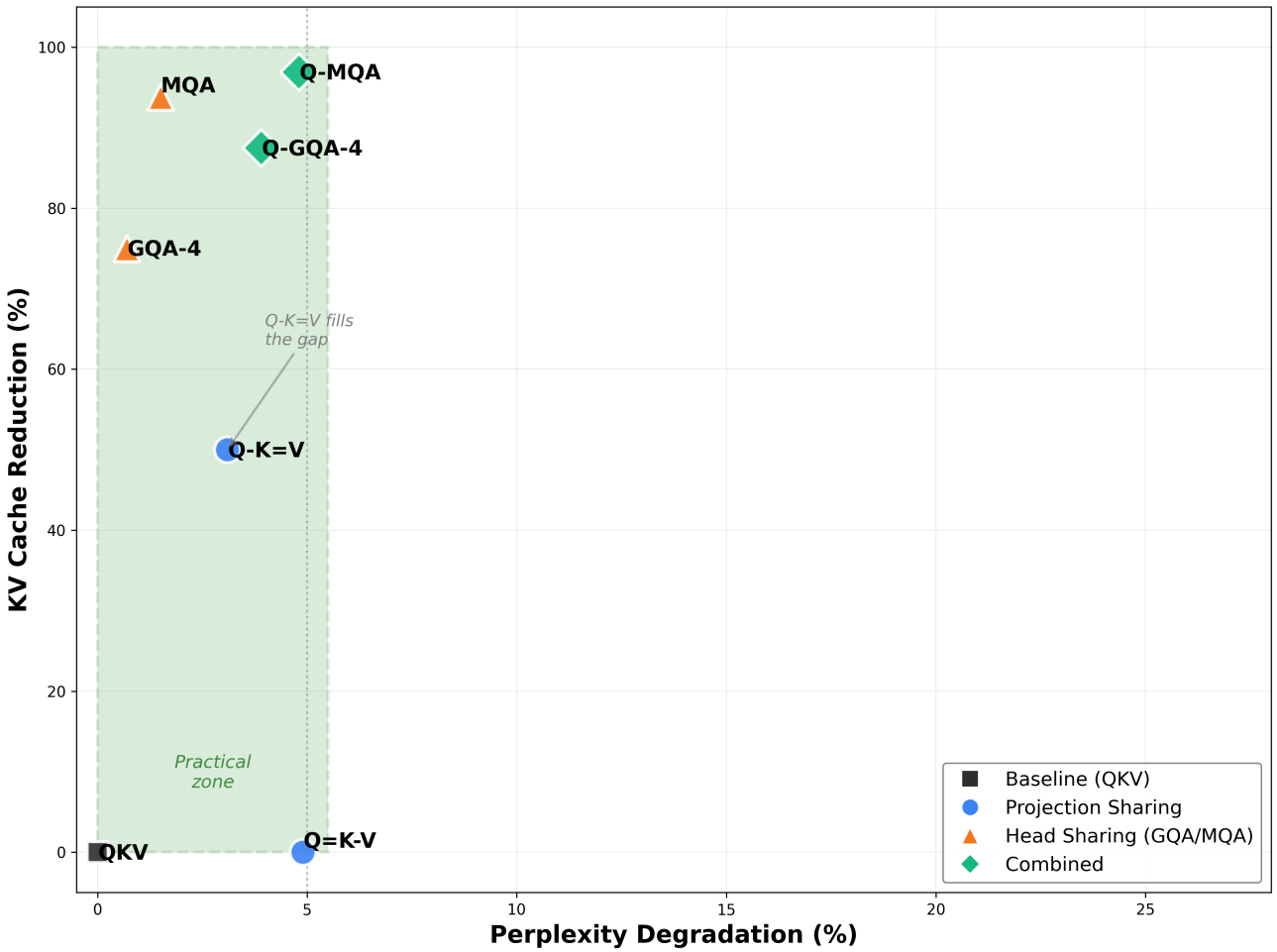

Figure 9. **Efficiency-quality Pareto frontier for attention variants.** Projection sharing (blue circles) and head sharing (orange triangles) occupy complementary regions. Combined approaches (green diamonds) achieve the highest cache reductions. The shaded region indicates practical deployment zone (<5% perplexity degradation). Q–K=V fills the gap between QKV baseline and head-sharing methods, providing 50% cache reduction with only 3.1% degradation.

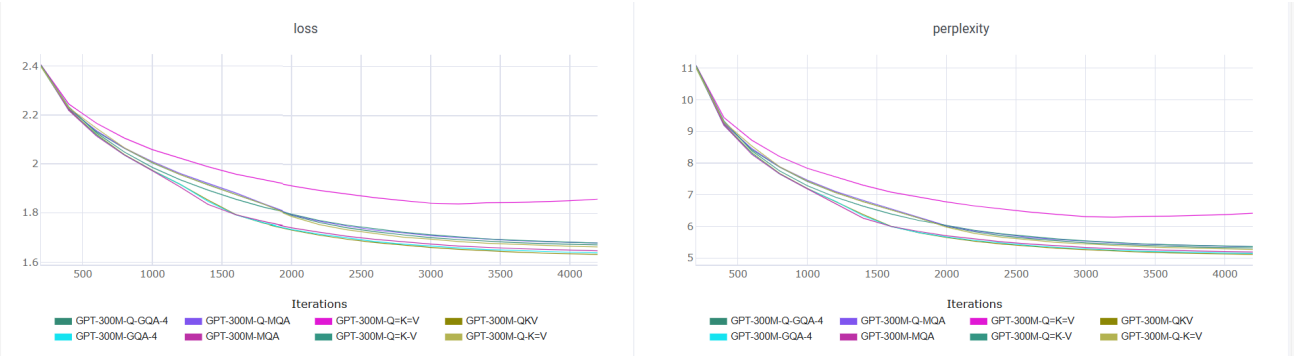

Figure 10. **Validation curves for 300M parameter models.** Left: Validation loss. Right: Validation perplexity over 10B training tokens. Q–K=V (dark teal) matches baseline QKV (olive) closely on held-out data, achieving 50% cache reduction with only 3.1% perplexity degradation. Q=K–V (light pink) shows higher validation loss, confirming suboptimal generalization. All head-sharing and combined variants converge to practical validation performance.

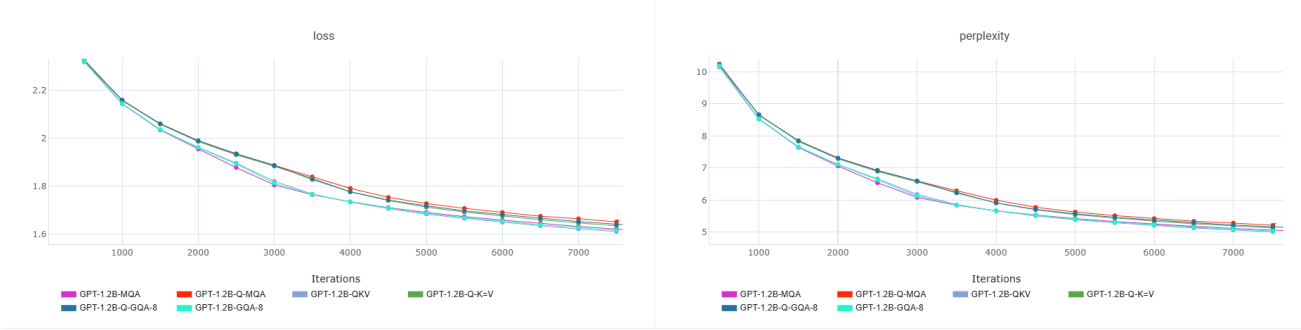

Figure 11. **Validation curves for 1.2B parameter models.** Left: Validation loss. Right: Validation perplexity over 8.85B training tokens. Rankings on held-out data remain consistent with 300M scale. Q–K=V (green) and head-sharing variants track baseline QKV (gray/brown) closely, while combined approaches (Q-GQA-8, Q-MQA) maintain $< 5\%$ degradation with 88-99% cache reduction, confirming scalability of our findings.

Table 11. Comprehensive summary of all attention mechanism variants evaluated. PE = Positional Encoding. Cache column shows what must be stored during autoregressive generation. Cache reduction and perplexity degradation were reported for 300M parameter models.

#	Notation	Projections	Cache	Cache↓	PPL Δ	Key Insight
Baseline						
1	QKV	Q, K, V	K+V	0%	0%	Standard attention
Projection Sharing						
2	Q=K-V	Q=K, V	K+V	0%	+4.9%	Symmetric, no cache benefit
3	(Q=K-V)$^+$	Q=K, V, +PE	K+V	0%	—	Adds 2D PE for asymmetry
4	Q–K=V	Q, K=V	K	50%	+3.1%	**50% Cache reduction (Optimal)**
5	Q=K=V	Q=K=V	K	50%	+25.4%	Too constrained
6	(Q=K=V)$^+$	Q=K=V, +PE	K	50%	—	PE doesn't recover quality
Head Sharing (Comparison Baselines)						
7	GQA-4	Q, K, V (4 groups)	K+V	75%	+0.7%	4 groups, 16 heads total
8	MQA	Q, K, V (1 head)	K+V	93.8%	+1.5%	Single KV head for all Q
Combined: Projection + Head Sharing						
9	Q-GQA-4	Q, K=V (4 groups)	K	87.5%	+3.9%	K=V within each group
10	Q-MQA	Q, K=V (1 head)	K	96.9%	+4.8%	K=V on single head

Key Takeaways from Additional Results

The visualizations and comprehensive comparisons in this appendix support several important conclusions:

1. **Q–K=V is the clear winner for projection sharing.** It achieves 50% cache reduction with only 3.1% perplexity degradation at 300M scale and 2.48% at 1.2B scale, representing a new point on the efficiency-quality Pareto frontier.
2. **Cache reduction, not parameter reduction, drives practical benefits.** While all projection sharing variants reduce parameters, only K=V constraints reduce inference memory. This explains why Q=K–V fails to provide deployment advantages despite competitive training quality.
3. **Projection and head sharing are strictly complementary.** Combined approaches achieve 87.5% (Q-GQA-4) to 96.9% (Q-MQA) cache reduction, enabling practical on-device inference for billion-parameter models.
4. **Quality rankings remain stable across scales.** The relative performance of all variants is consistent from 300M to 1.2B parameters, with larger models showing slightly better robustness to projection constraints.
5. **No training instabilities observed.** All variants converge smoothly without requiring specialized initialization, learning rate schedules, or architectural modifications beyond the attention mechanism itself.

These results establish projection sharing as a practical optimization for memory-efficient transformer deployment, particularly for applications requiring long contexts or high throughput in resource-constrained environments.