KVPATH: CONNECTING KEY-VALUE REPRESENTATIONS ACROSS TRANSFORMER LAYERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers typically cache a separate key-value (KV) representation for each token at every layer. We hypothesize that a token's KV representations across neighboring layers contain shared information, while layer-specific variation can be captured by low-dimensional vectors. Motivated by this, we propose KVPath, a new KV representation paradigm that connects KV representations across network depth. KVPath treats the sequence of KV representations for a token within a group of neighboring layers as a path. Each path is encoded by a shared base representation and compact delta vectors, which together determine the KV representation at each layer. Caching the base once together with the delta vectors preserves distinct layer-specific KV representations while avoiding the storage of a separate full KV representation at every layer. Experiments show that grouped-query attention (GQA) and multi-head latent attention (MLA) augmented with KVPath outperform standard GQA and MLA respectively under matched KV-cache and model-parameter budgets.

1 INTRODUCTION

Large language models (LLMs) (Vaswani et al., 2017; Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2023) rely on the key-value (KV) (Pope et al., 2023) cache to avoid recomputing the key-value representations of previous tokens during autoregressive generation. For each token, a standard Transformer independently produces and stores key and value representations at every layer. Consequently, KV-cache memory grows linearly with both sequence length and network depth, making it a major bottleneck for long-context inference and high-throughput serving (Kwon et al., 2023; Zheng et al., 2024). While substantial effort has been devoted to reducing KV-cache cost through quantization (Liu et al., 2024b), token eviction (Zhang et al., 2023), and low-dimensional representations (Chang et al., 2025), most approaches treat the KV representations produced at different Transformer layers independently.

Recent work has begun to exploit cross-layer redundancies in KV representations. Architectures like CLA (Brandon et al., 2024) and YOCO (Sun et al., 2024) adopt hard cross-layer KV sharing; however, exact sharing leaves no flexibility for layer-specific variation, restricting the model's overall representation capacity. On the other hand, post-training compression methods like MiniCache (Liu et al., 2024a) and xKV (Chang et al., 2026) rely on heuristics or singular value decomposition (SVD), which lack end-to-end learnability. This suggests a promising middle ground: learnably sharing information across layers while preserving layer-specific variations.

The way a Transformer updates its hidden states offers a useful perspective on how to realize such a design. Rather than constructing each layer's hidden representation from scratch, the model updates the preceding state through residual connections. For a token within a group of m neighboring layers, this evolution can be written as

$$h^{(0)} \xrightarrow{\Delta h^{(1)}} h^{(1)} \xrightarrow{\Delta h^{(2)}} \cdots \xrightarrow{\Delta h^{(m-1)}} h^{(m-1)}, \tag{1}$$

where each transition follows $h^{(l)} = h^{(l-1)} + \Delta h^{(l)}$. Since each layer projects its evolving hidden state into keys and values, a natural question arises: can we similarly form a progressive path for KV representations across depth? We hypothesize that the sequence of KV representations across neighboring layers can be viewed as a continuous path, where the layer-specific variations along this path can be captured by low-dimensional delta vectors relative to a shared base.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Building on this hypothesis, we introduce **KVPath**, a new KV representation paradigm that models a token's KV representations across neighboring layers as a structured trajectory. Specifically, KVPath represents key-value states across depth through a progressive sequence:

$$KV^{(0)} \xrightarrow{\Delta KV^{(1)}} KV^{(1)} \xrightarrow{\Delta KV^{(2)}} \cdots \xrightarrow{\Delta KV^{(m-1)}} KV^{(m-1)}, \tag{2}$$

where $\Delta KV^{(l)}$ denotes the layer-to-layer transition from layer $l-1$ to layer l. We propose two distinct instantiations of this path representation: a concatenation-based variant where each layer appends a compact delta vector to a shared base representation, and a residual-based variant where successive states are progressively updated by projecting low-dimensional delta vectors back to the full state dimension. Both variants store the shared base only once alongside the compact deltas, preserving distinct layer-specific KV representations without caching full vectors at every layer. Furthermore, KVPath is end-to-end learnable during pre-training, enabling the model to adaptively partition shared knowledge from layer-specific variations.

Our experiments show that, under matched parameter counts and KV-cache budgets, GQA and MLA augmented with KVPath outperform their vanilla counterparts. KVPath instantiated with GQA also outperforms CLA under the same constraints. At matched parameter counts, KVPath reduces KV-cache storage by 37.5% for both architectures while retaining comparable performance; for MLA, average perplexity even decreases by 1.89%. Taken together, these results suggest that compact KV representations learned across layers can make more effective use of cache capacity, improving model quality at fixed budgets and reducing storage requirements at comparable quality. We will release our code and model checkpoints.

2 METHOD

We introduce KVPath, a KV representation paradigm that models key and value states across network depth as a structured trajectory. Our central hypothesis is that neighboring KV representations share a foundational base, while layer-to-layer transitions along depth can be captured by compact, low-dimensional delta vectors.

2.1 PATH REPRESENTATION

We partition the model's L layers into groups of m consecutive layers, assuming m divides L. For each token and KV head, the states within a group form a *path*, indexed by $\ell = 0, \ldots, m-1$. Let $k^{(\ell)}, v^{(\ell)} \in \mathbb{R}^d$ denote the key and value states before positional encoding. KVPath constructs successive states using compact, layer-specific delta components $k_\delta^{(\ell)}, v_\delta^{(\ell)} \in \mathbb{R}^{d_\delta}$, where $d_\delta < d$:

$$\begin{aligned} k^{(\ell)} &= \mathcal{F}_K^{(\ell)}\big(k^{(\ell-1)}, k_\delta^{(\ell)}\big), \\ v^{(\ell)} &= \mathcal{F}_V^{(\ell)}\big(v^{(\ell-1)}, v_\delta^{(\ell)}\big), \qquad 1 \le \ell < m. \end{aligned} \tag{3}$$

The transition $\mathcal{F}^{(\ell)}$ takes different forms in the two constructions described below.

2.2 PATH CONSTRUCTION

We realize the compact cross-layer changes in two ways, illustrated in Figure 1. Both begin with hidden states generated by the first layer, which we call the *alpha component*, and use a low-dimensional *delta component* to describe variation along the path. In KVPath-Concat, the alpha component occupies shared coordinates of every KV state, while the delta component supplies the remaining, layer-dependent coordinates. In KVPath-Residual, the alpha component is the full initial KV state, while each subsequent layer's delta component encodes an update that is expanded and added to the preceding state.

KVPath-Concat: variation in fixed coordinates. As shown in Figure 1(a), each KV state consists of a shared alpha component (k_α, v_α) with d_α dimensions per key or value head and a layer-dependent delta component $(k_\delta^{(\ell)}, v_\delta^{(\ell)})$ with d_δ dimensions. The first layer generates the alpha component through a path-specific projection W_α. Each layer, including layer 0, generates its delta

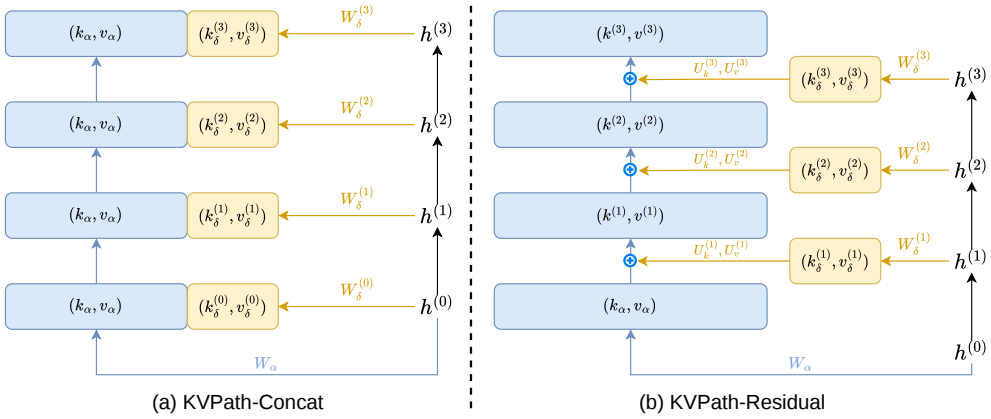

Figure 1: Two KVPath constructions within a four-layer path. (a) KVPath-Concat: each layer concatenates an identical alpha component with its own delta component. Cross-layer differences are confined to the coordinates of the delta component. (b) KVPath-Residual: the alpha component initializes the full K/V state. Subsequent layers project their inputs into low-dimensional K/V updates, expand them separately, and add them to the preceding state.

component through its own projection $W_\delta^{(\ell)}$:

$$\begin{aligned}(k_\alpha, v_\alpha) &= W_\alpha h^{(0)}, \\ (k_\delta^{(\ell)}, v_\delta^{(\ell)}) &= W_\delta^{(\ell)} h^{(\ell)}, \qquad 0 \le \ell < m.\end{aligned} \tag{4}$$

Every layer reuses the alpha component without modification and concatenates it with its own delta component to form the full state:

$$\begin{aligned}k^{(\ell)} &= [k_\alpha \,\|\, k_\delta^{(\ell)}], \\ v^{(\ell)} &= [v_\alpha \,\|\, v_\delta^{(\ell)}], \qquad 0 \le \ell < m.\end{aligned} \tag{5}$$

where $\|$ denotes concatenation and $d = d_\alpha + d_\delta$. We cache the alpha component once per path and the delta component for each of the m layers within the path. Counting both K and V, the resulting storage is $2d_\alpha + 2md_\delta$.

KVPath-Residual: cumulative low-dimensional updates. As shown in Figure 1(b), the alpha component now denotes the entire initial state (k_α, v_α), with d dimensions per key or value head. Each subsequent layer produces a delta component $(k_\delta^{(\ell)}, v_\delta^{(\ell)})$, consisting of separate d_δ-dimensional K and V update coordinates. A path-specific projection W_α initializes the state, while layer-specific projections $W_\delta^{(\ell)}$ generate the compact updates:

$$\begin{aligned}(k^{(0)}, v^{(0)}) &= (k_\alpha, v_\alpha) = W_\alpha h^{(0)}, \\ (k_\delta^{(\ell)}, v_\delta^{(\ell)}) &= W_\delta^{(\ell)} h^{(\ell)}, \qquad 1 \le \ell < m.\end{aligned} \tag{6}$$

The learned matrices $U_K^{(\ell)}, U_V^{(\ell)} \in \mathbb{R}^{d \times d_\delta}$ expand these coordinates separately for K and V. Each layer adds the expanded updates to the preceding full state:

$$\begin{aligned}k^{(\ell)} &= k^{(\ell-1)} + U_K^{(\ell)} k_\delta^{(\ell)}, \\ v^{(\ell)} &= v^{(\ell-1)} + U_V^{(\ell)} v_\delta^{(\ell)}, \qquad 1 \le \ell < m.\end{aligned} \tag{7}$$

A compact representation stores the full alpha component once and the delta components for the $m-1$ subsequent layers. Therefore the resulting storage is $2d + 2(m-1)d_\delta$.

2.3 ATTENTION WITH KVPATH

For KVPath-Concat, the query is split into an alpha component $q_\alpha^{(\ell)}$ and a delta component $q_\delta^{(\ell)}$, with dimensions d_α and d_δ, respectively. RoPE is applied only to the delta components of queries and keys, while the alpha components remain unrotated:

$$\begin{aligned}(q_\alpha^{(\ell)}, q_\delta^{(\ell)}) &= W_Q^{(\ell)} h^{(\ell)},\\ q^{(\ell)} &= [q_\alpha^{(\ell)} \parallel \mathrm{RoPE}(q_\delta^{(\ell)})],\\ k^{(\ell)} &= [k_\alpha \parallel \mathrm{RoPE}(k_\delta^{(\ell)})],\\ v^{(\ell)} &= [v_\alpha \parallel v_\delta^{(\ell)}].\end{aligned} \tag{8}$$

For KVPath-Residual, the query is obtained as $W_Q^{(\ell)} h^{(\ell)}$, and the keys and values are reconstructed using Eq. equation 7. Full-width RoPE is then applied to the query and reconstructed key, while the value remains unchanged.

With queries, keys, and values constructed as above, both variants use standard causal scaled dot-product attention with scale $1/\sqrt{d}$. KVPath constructs paths independently for each KV head and preserves the mapping between query heads and KV heads, making it directly applicable to MHA (Vaswani et al., 2017), GQA (Ainslie et al., 2023), and MQA (Shazeer, 2019). It also extends to MLA (DeepSeek-AI, 2024) by constructing paths over compressed KV latents, as detailed in the following subsection for both Concat and Residual variant.

2.4 COMPATIBILITY WITH MLA

Multi-head latent attention (MLA) (DeepSeek-AI, 2024) caches a compressed KV latent shared across attention heads. Each head uses its own up-projections to map this latent into distinct content keys and values. Queries and keys each consist of two components: an unrotated content component and a positional component to which RoPE is applied. The positional key is shared across heads and cached separately from the KV latent. The content and positional components jointly determine a single attention distribution over the values. KVPath extends this design by sharing latent information across neighboring layers while retaining head-specific projections.

In our MLA constructions, we use d_α for the KV latent dimension and d_h for the dimension of each reconstructed content key and value per head. Below, we suppress the head index: queries and up-projections are head-specific, while cached latents and delta components are shared across heads.

KVPath-MLA-Concat. The first layer of each path generates a shared alpha latent $c_\alpha \in \mathbb{R}^{d_\alpha}$. Each layer reconstructs its own alpha keys and values from this latent and generates delta components from its hidden state:

$$\begin{aligned}c_\alpha &= W_\alpha h^{(0)},\\ (k_\alpha^{(\ell)}, v_\alpha^{(\ell)}) &= W_U^{(\ell)}\, \mathrm{RMSNorm}^{(\ell)}(c_\alpha),\\ (k_\delta^{(\ell)}, v_\delta^{(\ell)}) &= W_\delta^{(\ell)} h^{(\ell)}.\end{aligned} \tag{9}$$

Here W_α projects the first layer's input into the shared latent. Each layer's $W_U^{(\ell)}$ maps this latent to head-specific alpha keys and values $k_\alpha^{(\ell)}, v_\alpha^{(\ell)} \in \mathbb{R}^{d_h}$, while $W_\delta^{(\ell)}$ produces $k_\delta^{(\ell)}, v_\delta^{(\ell)} \in \mathbb{R}^{d_\delta}$ shared across heads. Thus, sharing the alpha latent still allows different layers and heads to reconstruct distinct alpha components.

Each query head produces matching components $q_\alpha^{(\ell)} \in \mathbb{R}^{d_h}$ and $q_\delta^{(\ell)} \in \mathbb{R}^{d_\delta}$. Queries, keys, and values are concatenated as in Eq. equation 8, using the reconstructed $k_\alpha^{(\ell)}$ and $v_\alpha^{(\ell)}$. RoPE is applied only to the query and key delta components, and attention uses standard causal scaled dot-product attention with $d = d_h + d_\delta$. The compact cache stores a single alpha latent shared across the path, along with the key and value delta components for each of the m layers. The resulting cache size per token per path is $d_\alpha + 2md_\delta$.

KVPath-MLA-Residual. KVPath-Residual applies cumulative low-dimensional updates to the shared KV latent space while retaining MLA's decoupled positional key. For a path of m layers, the latents and keys are constructed as:

$$\begin{aligned} c^{(0)} &= W_\alpha h^{(0)}, \\ \delta^{(\ell)} &= W_\delta^{(\ell)} h^{(\ell)}, \qquad 1 \le \ell < m, \\ c^{(\ell)} &= c^{(\ell-1)} + U_c^{(\ell)} \delta^{(\ell)}, \qquad 1 \le \ell < m, \\ (k_\alpha^{(\ell)}, v_\alpha^{(\ell)}) &= W_U^{(\ell)} \,\mathrm{RMSNorm}(c^{(\ell)}), \qquad 0 \le \ell < m, \\ k_R^{(\ell)} &= W_R^{(\ell)} h^{(\ell)}, \qquad 0 \le \ell < m. \end{aligned} \tag{10}$$

The first layer initializes the d_α-dimensional latent $c^{(0)}$. Each subsequent layer generates a compact d_δ-dimensional update $\delta^{(\ell)}$, expanded by $U_c^{(\ell)} \in \mathbb{R}^{d_\alpha \times d_\delta}$ and added to the preceding latent. At every layer, head-specific projections $W_U^{(\ell)}$ reconstruct $k_\alpha^{(\ell)}$ and $v_\alpha^{(\ell)}$ (d_h dims) from $c^{(\ell)}$, while $W_R^{(\ell)}$ generates a shared positional key $k_R^{(\ell)} \in \mathbb{R}^{d_R}$.

For attention computation at layer ℓ, the hidden state $h^{(\ell)}$ is projected into content query $q_\alpha^{(\ell)} \in \mathbb{R}^{d_h}$ and positional query $q_R^{(\ell)} \in \mathbb{R}^{d_R}$. RoPE is applied exclusively to positional components, forming $q^{(\ell)} = [q_\alpha^{(\ell)} \,\|\, \mathrm{RoPE}(q_R^{(\ell)})]$ and $k^{(\ell)} = [k_\alpha^{(\ell)} \,\|\, \mathrm{RoPE}(k_R^{(\ell)})]$. Standard causal scaled dot-product attention with scaling factor $1/\sqrt{d_h + d_R}$ is then executed over key $k^{(\ell)}$ and value $v_\alpha^{(\ell)}$. Across each path, storing the initial latent $c^{(0)}$ (d_α dims), $m-1$ updates $\delta^{(\ell)}$ (d_δ dims each), and m positional keys $k_R^{(\ell)}$ (d_R dims each) requires a cache footprint of $d_\alpha + (m-1)d_\delta + md_R$ values per token.

3 Experiments

3.1 Implementation Details

We train all the models on FineWeb-Edu (Penedo et al., 2024) using the GPT-2 BPE tokenizer (Radford et al., 2019). Training proceeds in three stages: we first train from scratch at a context length of 2K on 10 billion tokens, then extend the context to 8K and subsequently to 32K, using 1 billion additional tokens at each stage. Each extension is initialized from the final checkpoint of the preceding stage. We use Rotary Position Embeddings (RoPE) (Su et al., 2024) with the RoPE base fixed at 10^6 throughout training. We use AdamW (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, weight decay 0.1, and gradient clipping at 1.0. The global batch sizes are 256, 64, and 16 sequences for the 2K, 8K, and 32K stages, respectively, maintaining 524,288 input tokens per optimizer update. The peak learning rate is 10^{-3} for the initial 2K stage and 10^{-4} for the 8K and 32K extensions. We use linear learning-rate warmup followed by cosine decay (Loshchilov & Hutter, 2017) at each stage, with 100 warmup steps.

3.2 Main Results

We compare GQA (Ainslie et al., 2023), CLA (Brandon et al., 2024), KVPath-Concat, and KVPath-Residual under matched parameter and KV-cache budgets. All four models use 12 layers, a hidden size of 512, 8 query heads, and 4 KV heads. CLA uses a group size of four, and both KVPath variants use a path length of four, so each sharing group or path spans four consecutive layers. We adjust the SwiGLU intermediate dimensions to give all models exactly 98,710,016 parameters. Each model caches 6,144 elements per token across the full model, as detailed below.

GQA. Our GQA baseline uses a head dimension of 64 and independently generates and caches keys and values at every layer. With 12 layers and four KV heads, its cache contains 2 x 12 x 4 x 64 = 6,144 elements per token.

CLA. Our CLA baseline uses a head dimension of 256 and a group size of four. It shares the full K/V states within each group, while retaining independent query and output projections at each layer. With three groups and four KV heads, its cache contains 2 x 3 x 4 x 256 = 6,144 elements per token.

Table 1: Perplexity (↓) of the final 32K checkpoints evaluated at three context lengths.

Context	Model	C4	FineWeb-Edu	Wikipedia	PG-19	Avg.
2K	GQA	35.67	21.94	32.56	33.58	30.94
	CLA	36.22	22.41	33.13	35.11	31.72
	KVPath-Concat	**33.81**	**20.89**	**32.35**	**32.87**	**29.98**
	KVPath-Residual	34.21	21.05	32.61	33.71	30.40
8K	GQA	34.42	21.14	30.82	31.10	29.37
	CLA	34.91	21.56	31.20	32.67	30.08
	KVPath-Concat	**32.50**	**20.04**	**30.34**	**30.30**	**28.29**
	KVPath-Residual	32.93	20.24	30.74	31.11	28.76
32K	GQA	34.15	20.95	30.38	30.68	29.04
	CLA	34.57	21.33	30.67	32.22	29.70
	KVPath-Concat	**32.23**	**19.80**	**29.59**	**29.36**	**27.75**
	KVPath-Residual	32.65	20.02	30.19	30.41	28.32

KVPath-Concat. KVPath-Concat uses a path length of four. Each key and value head concatenates a shared 128-dimensional alpha component with a 32-dimensional delta component generated at each layer, giving a head dimension of 160. With three paths and four KV heads, its cache contains 2 x 3 x 4 x (128 + 4 x 32) = 6,144 elements per token.

KVPath-Residual. KVPath-Residual also uses a path length of four and a head dimension of 160. The first layer of each path initializes the full K/V states. Each subsequent layer generates separate 32-dimensional K and V updates, projects them to 160 dimensions, and adds them to the preceding states. With three paths and four KV heads, its cache contains 2 x 3 x 4 x (160 + 3 x 32) = 6,144 elements per token.

We evaluate on C4 (Raffel et al., 2020), FineWeb-Edu (Penedo et al., 2024), Wikipedia (Wikimedia Foundation, n.d.), and PG-19 (Rae et al., 2020). For each model, we evaluate the final 32K checkpoint at context lengths of 2K, 8K, and 32K. All models use identical evaluation tokens and sequence boundaries at each context length. We use approximately 10M tokens per benchmark and hold the evaluated token region fixed across context lengths, partitioning it into non-overlapping sequences. We report GPT-2 token-level perplexity, and compute the average as the unweighted arithmetic mean over the four benchmarks.

Table 1 shows that KVPath-Concat achieves the lowest perplexity on all four benchmarks at every context length. Relative to GQA, it reduces average perplexity by 3.09%, 3.65%, and 4.45% at 2K, 8K, and 32K, respectively. KVPath-Residual also improves average perplexity over GQA, with reductions of 1.75%, 2.08%, and 2.48%. The increasing average gains of both KVPath variants at longer contexts support the usefulness of compact layer-dependent KV representations under a fixed cache budget.

3.3 Results with MLA

We evaluate the two MLA constructions introduced in Section 2.4 under the same KV-cache budget as standard MLA (DeepSeek-AI, 2024). All three models use 12 layers, a hidden size of 512, and eight attention heads, following the training and evaluation protocols described above. Both KVPath variants use a path length of four, giving three paths across the model. Each model requires 1,920 cached elements per token, equivalent to 3,840 bytes in BF16. These counts describe the compact representations for incremental decoding and exclude metadata and temporary attention tensors. We adjust the SwiGLU intermediate dimensions to give all three models exactly 99,694,592 parameters.

MLA. The baseline caches a 128-dimensional KV latent and a 32-dimensional positional key at each layer. Head-specific up-projections reconstruct 64-dimensional content keys and values from the latent. Its total cache size is $12(128 + 32) = 1{,}920$ elements per token.

Table 2: Perplexity (↓) of MLA and its KVPath variants at three evaluation context lengths.

Context	Model	C4	FineWeb-Edu	Wikipedia	PG-19	Avg.
2K	MLA	36.27	22.22	32.22	34.88	31.40
	KVPath-MLA-Concat	**35.14**	**21.63**	**30.74**	**32.80**	**30.08**
	KVPath-MLA-Residual	35.45	21.83	32.17	34.00	30.86
8K	MLA	34.91	21.37	30.48	32.18	29.74
	KVPath-MLA-Concat	**33.82**	**20.78**	**28.86**	**30.05**	**28.38**
	KVPath-MLA-Residual	34.12	20.99	30.24	31.33	29.17
32K	MLA	34.63	21.15	30.08	31.28	29.28
	KVPath-MLA-Concat	**33.52**	**20.55**	**28.36**	**29.34**	**27.95**
	KVPath-MLA-Residual	33.84	20.77	29.77	30.58	28.74

KVPath-MLA-Concat. Each four-layer path shares a 384-dimensional alpha latent. Each layer reconstructs 64-dimensional alpha keys and values per head and independently generates 32-dimensional key and value delta components shared across heads. Concatenation gives 96-dimensional keys and values, with RoPE applied only to the query and key delta components. The cache stores one alpha latent per path and both delta components at every layer, totaling $3 \times 384 + 12(32 + 32) = 1{,}920$ elements per token.

KVPath-MLA-Residual. Each four-layer path begins with a 320-dimensional latent. The remaining three layers each generate a 64-dimensional update, which is projected to 320 dimensions and added to the preceding latent. Each layer reconstructs 64-dimensional content keys and values per head and generates a separate 32-dimensional positional key shared across heads. The cache stores the initial latent, the compact updates, and the positional keys, totaling $3 \times 320 + 9 \times 64 + 12 \times 32 = 1{,}920$ elements per token.

Table 2 shows that both KVPath variants achieve lower perplexity than MLA on all four benchmarks at every evaluation context length. KVPath-MLA-Concat performs best throughout, reducing average perplexity relative to MLA by 4.20%, 4.57%, and 4.57% at 2K, 8K, and 32K, respectively. KVPath-MLA-Residual also improves average perplexity, with reductions of 1.70%, 1.90%, and 1.86%. These results show that sharing information across layers can improve model quality under a fixed cache budget even when the cached representations are already compressed into MLA latents.

3.4 Scaling Analysis

To test whether the gains persist at a larger model scale, we evaluate KVPath-Concat, the best-performing construction in our preceding experiments, against the GQA baseline. Both models have exactly 606,405,632 parameters and use 32 layers, a hidden size of 1024, 16 query heads, and eight KV heads. GQA uses a head dimension of 64, while KVPath-Concat uses four-layer paths with a shared 128-dimensional alpha component and a 32-dimensional delta component per key or value head. The SwiGLU intermediate dimensions are 4096 and 3072, respectively, giving identical parameter counts. Both models require 32,768 cached elements per token. We train each model on approximately 56B tokens: 50B at a context length of 2K with a global batch size of 1024 sequences, followed by 5B at 8K with a global batch size of 256 sequences and 1B at 32K with a global batch size of 64 sequences. Each stage starts from the final checkpoint of the preceding stage, and all results below use the final 32K checkpoints.

Table 3 reports perplexity on the four benchmarks used in our main experiments. KVPath-Concat achieves lower perplexity than GQA across all benchmarks and all three evaluation context lengths (2K, 8K, and 32K), achieving relative reductions of 3.73%, 4.22%, and 4.81%, respectively. We also evaluate both models on eight downstream tasks in a zero-shot setting with a 2K context limit, using identical examples and prompts. As shown in Table 4, KVPath-Concat achieves higher accuracy on seven of the eight tasks.

Table 3: Perplexity (↓) at the 0.6B scale after 56B training tokens.

Context	Model	C4	FineWeb-Edu	Wikipedia	PG-19	Avg.
2K	GQA	21.93	12.92	19.63	20.29	18.69
	KVPath-Concat	**21.06**	**12.32**	**18.63**	**19.96**	**17.99**
8K	GQA	21.15	12.46	18.59	19.11	17.83
	KVPath-Concat	**20.30**	**11.84**	**17.60**	**18.55**	**17.07**
32K	GQA	21.06	12.39	18.40	19.06	17.73
	KVPath-Concat	**20.36**	**11.76**	**17.33**	**18.05**	**16.88**

Table 4: Zero-shot evaluation at the 0.6B scale with a 2K context limit.

Task	Metric	GQA	KVPath-Concat
HellaSwag	Acc. norm. (↑)	45.78	**48.66**
PIQA	Acc. (↑)	69.42	**70.18**
ARC-Easy	Acc. norm. (↑)	57.83	**60.23**
WinoGrande	Acc. (↑)	**55.25**	54.62
SciQ	Acc. (↑)	83.40	**87.60**
LAMBADA	Acc. (↑)	36.81	**38.93**
	Target-word PPL (↓)	28.97	**22.88**
BoolQ	Acc. (↑)	56.67	**61.50**
OpenBookQA	Acc. norm. (↑)	34.00	**34.20**

3.5 Ablation Study

We study the delta dimension d_δ and path length m using the 12-layer KVPath-Concat configuration from Section 3.2. We fix $d_\alpha = 128$ and the SwiGLU intermediate dimension at 1536, varying one factor at a time from the default $(m, d_\delta) = (4, 32)$. The default setting reuses the main-results checkpoint. Cache and parameter counts vary with these settings. Table 5 summarizes average perplexity across the four benchmarks. The results show that larger delta dimensions and shorter paths improve average perplexity at all three context lengths, while increasing cache storage.

3.6 Cache Budgets at Comparable Performance

We also compare cache requirements at comparable performance, complementing the preceding evaluation of quality under fixed cache budgets. Within each comparison, parameter counts are matched exactly: 98,710,016 for GQA and its KVPath variant, and 99,694,592 for MLA and its KVPath variant.

As shown in Table 6, KVPath-Concat reduces the cache by 37.5% for both architectures. Across the four benchmarks and three context lengths, average perplexity increases by approximately 0.4% for GQA and decreases by 1.89% for MLA. Thus, KVPath saves approximately 37.5% of the KV-cache storage at comparable or better average perplexity without increasing model parameter counts.

4 Limitations

Our experiments are limited to models with up to approximately 0.6B parameters due to computational constraints. Whether the observed quality and KV-cache benefits persist at larger model scales remains to be evaluated.

We have not developed specialized attention kernels for KVPath with complicated optimizations available in systems such as FlashAttention (Dao et al., 2022) and FlashInfer (Ye et al., 2025). Efficiently accessing shared components and reconstructing layer-specific states from the compact cache requires nontrivial kernel design. We therefore report compact KV-cache requirements rather than throughput comparisons. Developing optimized kernels and evaluating end-to-end serving performance are left for future work.

Table 5: KVPath-Concat ablations with $d_\alpha = 128$. Cache is measured in elements per token across the full model. Perplexity (↓) is averaged over the four benchmarks.

m	d_δ	Cache	2K PPL	8K PPL	32K PPL
4	16	4,608	30.60	28.90	28.51
4	32	6,144	29.98	28.29	27.75
4	64	9,216	29.31	27.71	27.22
2	32	9,216	29.57	27.98	27.50
4	32	6,144	29.98	28.29	27.75
12	32	4,096	30.66	29.16	28.87

Table 6: Cache requirements at comparable performance and matched parameter counts. Cache is measured in elements per token across the full model. Perplexity (↓) is averaged over the four benchmarks using the final 32K checkpoints.

Model	Cache	2K PPL	8K PPL	32K PPL
GQA	6,144	30.94	29.37	29.04
+ KVPath-Concat	3,840	31.17	29.45	29.08
MLA	1,920	31.40	29.74	29.28
+ KVPath-Concat	1,200	30.82	29.17	28.72

5 Related Work

KV-cache reduction includes head sharing in MQA (Shazeer, 2019) and GQA (Ainslie et al., 2023), latent compression in MLA (DeepSeek-AI, 2024), low-rank projection in Palu (Chang et al., 2025), quantization (Liu et al., 2024b), and token eviction (Zhang et al., 2023). These approaches primarily target within-layer redundancy, precision, or token retention; KVPath instead structures representations across depth, supporting both KV states and compressed latents.

Cross-layer architectures differ in how they preserve layer-specific information. CLA (Brandon et al., 2024), YOCO (Sun et al., 2024), and MLKV (Zuhri et al., 2024) reuse identical KV states across sharing layers, precluding variation in the shared states. CLLA (Yang et al., 2024) permits distinct reconstructions through layer-specific projections of shared latents, but leaves the shared content latent unchanged across layers. KVPath explicitly supplements a shared base with compact, token-dependent deltas generated from layer-specific hidden states, allowing layer-specific information to enter the cached representation.

Post-training methods exploit redundancy in pretrained models: MiniCache (Liu et al., 2024a) merges neighboring KV states, xKV (Chang et al., 2026) applies cross-layer SVD, and CommonKV (Wang et al., 2025) combines joint weight decomposition with latent-cache merging. XQuant-CL (Tomar et al., 2025) quantizes inter-layer input differences for KV rematerialization. In contrast, KVPath learns its base-and-delta representation end-to-end during pre-training, directly shaping KV states or latents through concatenation or residual updates. Its distinction lies in jointly learning shared information and compact layer-specific variation, rather than imposing exact sharing or compressing existing representations.

6 Conclusion

We introduced KVPath, which treats a token's KV representations across neighboring layers as a path encoded by a shared base and compact delta vectors. Through Concat and Residual constructions, the model learns during pre-training what information to share across depth and what variation to preserve at each layer. Experiments with GQA and MLA demonstrate improved quality under matched parameter and KV-cache budgets. More broadly, the path perspective suggests that depth-wise specialization can be supported by shared information and low-dimensional variation, motivating attention architectures that learn KV representations jointly across layers.

AI USE STATEMENT

We used generative AI tools to assist with language editing and improve the clarity and readability of the manuscript. We also used OpenAI Codex to assist with the implementation of the model code used in this work. The authors take responsibility for the final manuscript, the implementation, and the reported results.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, et al. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems*, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, et al. Palu: KV-cache compression with low-rank projection. In *International Conference on Learning Representations*, 2025.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Hung-Yueh Chiang, Yash Akhauri, et al. xKV: Cross-layer KV-cache compression via aligned singular vector extraction. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, et al. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, et al. MiniCache: KV cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems*, 2024a.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, et al. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning*, 2024b.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, et al. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, 2024.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, et al. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems*, 2023.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, et al. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 2020.

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

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, et al. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, et al. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems*, 2024.

Aditya Tomar, Coleman Hooper, Minjae Lee, Haocheng Xi, Rishabh Tiwari, et al. XQuant: Breaking the memory wall for LLM inference with KV cache rematerialization. *arXiv preprint arXiv:2508.10395*, 2025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Yixuan Wang, Haoyu Qiao, Lujun Li, Qingfu Zhu, and Wanxiang Che. CommonKV: Compressing KV cache with cross-layer parameter sharing. *arXiv preprint arXiv:2508.16134*, 2025.

Wikimedia Foundation. Wikimedia downloads, n.d.

Zhen Yang, J. N. Han, Kan Wu, Ruobing Xie, An Wang, et al. Lossless KV cache compression to 2%. *arXiv preprint arXiv:2410.15252*, 2024.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, et al. FlashInfer: Efficient and customizable attention engine for LLM inference serving. In *Proceedings of Machine Learning and Systems*, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, et al. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, et al. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. MLKV: Multi-layer key-value heads for memory efficient transformer decoding. *arXiv preprint arXiv:2406.09297*, 2024.