Do Value Vectors in Deep Layers Need Context from the Residual Stream?

Anonymous ACL submission

Abstract

The success of the transformer architecture as the backbone of modern LLMs is in large part due to its use of attention layers. An attention layer follows the standard neural network paradigm: it takes the residual stream as input and thereby produces context-dependent query, key, and value vectors. However, we find that model performance meaningfully improves when deeper layers learn a context-free value vector that preserves the original token information, without drawing on any context information from the residual stream. When the model has access to this context-free value vector, adding back the context-dependent component of the value vector provides no gain. Such context-free value vectors can be stored as sparse model parameters, thereby eliminating the need for recomputation or caching. Through systematic ablations on the key design choices for such context-free value vectors, we propose **Bank of Values** (BoV), a new way of computing value vectors in attention by learning a lookup table of token-specific value vectors for each of the last third of layers. BoV consistently improves upon the standard attention mechanism in both validation loss and downstream benchmark performance across two model sizes, and surpasses competing attention architectures while using a fraction of the compute and memory.

1 Introduction

Modern LLMs invariably adopt the transformer architecture, whose basic formulation necessarily includes the attention layer (Vaswani et al., 2017). Attention computes dot products between query and key vectors to obtain attention scores, then writes a weighted sum of value vectors back to the residual stream according to these scores. Despite its sophisticated design, the transformer retains an old paradigm first proposed in ResNet (He et al., 2016): the input to each layer of a deep network is the aggregate output of all previous layers, which we refer to as the residual stream. This design proves effective for computing queries and keys, as it allows deeper layers to accumulate context information propagated from other tokens through attention (Ghandeharioun et al., 2024). However, whether context information is necessary for value vectors is left largely unexamined, even though value vectors do not themselves participate in cross-token computation. As a result, the residual stream remains the default input for computing value vectors, diluting the original context-free token information available to them.

Recent work has begun to explore whether attention benefits from increasing the share of original token information relative to the existing context information in the value vector. Value Residual Learning (Zhou et al., 2024) linearly interpolates the value vectors in deeper layers with the value vector from the first attention layer. Since the value vector in the first layer is a linear transformation of the token embedding, it deposits original token information into the value vectors in later layers. Similarly, MoVE (Li, 2026) and nanochat (Karpathy, 2025) modulate value vectors in later layers with a token-specific value vector learned for that layer. All of these methods claim improvements over the standard attention mechanism, implying that a larger share of original token information in the value vector benefits deeper layers. At the same time, studies have explored using only a context-free value vector for attention, directly reusing the first layer's value vector (SVFormer, Zhou et al., 2024), or components of it (SkipV1Former, Wu et al., 2025), in all subsequent layers. These methods report degraded performance relative to the standard attention baseline. Together, the results suggest that context information from the residual stream is essential to value vectors in deeper layers, and that adding a context-free component that preserves the original

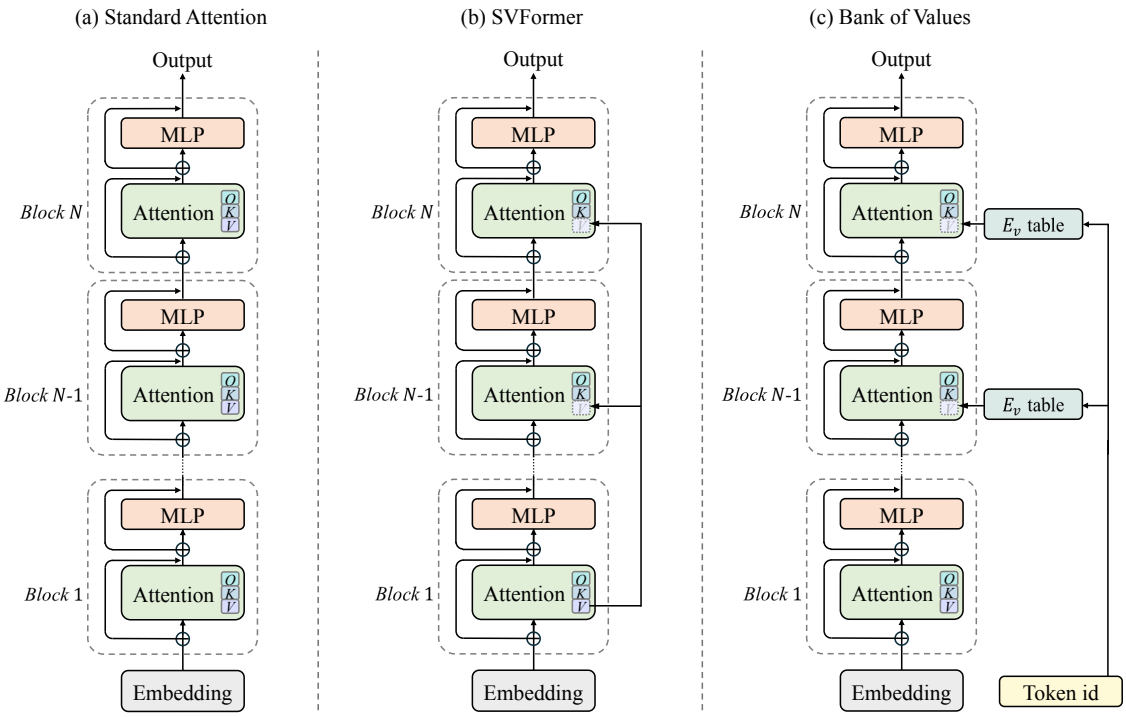

Figure 1: Overview of Bank of Values. (a) Standard Attention: a context-dependent value vector is computed directly from the residual stream. (b) SVFormer: a context-free value vector is copied from the first layer's value vector into all subsequent layers. (c) Bank of Values: in each of the last third of layers, a context-free value vector is looked up by token id from that layer's value vector table $\mathbf{E}_v$ and scaled by a learnable coefficient.

token information is beneficial but not sufficient.

We find that all the aforementioned studies share the same oversight: when targeting value vectors in deeper layers, the effect of computing a single context-free value vector is never studied in isolation. In the rare cases where it is, the context-free value vector is nevertheless applied to all layers, which fails to account for different layers' varying dependence on context. Systematically ablating how value vectors are computed along four axes (Section 3), we find that almost all of the reported gains in previous work come from the context-free component that represents the original token, not from the context-dependent component. Removing the standard computation path of value vectors from the residual stream has little to no impact on model performance.

The insight that deeper layers largely benefit from a context-free value vector has an important consequence: since these value vectors do not require context information, they need not be computed from a particular input sequence, nor be cached for reuse in subsequent decode steps. Instead, context-free value vectors can persist as FLOP-free, sparse model parameters that are looked up in $O(1)$ time during a forward pass. We therefore propose **Bank of Values** (BoV), which replaces the standard value vector computation from the residual stream with a static lookup in a dedicated value vector table in each of the last third of layers. As a result, deeper layers no longer store a value matrix among their parameters, nor do they cache anything other than previous token indices for fast retrieval. Although value vector lookup tables have themselves been explored in the works above, they are uniformly learned in addition to the standard context-dependent value vector, incurring substantial memory pressure and FLOP overhead for minimal gains (Section 5.2).

Empirically, we train a transformer model with BoV at two model sizes, 135M and 780M, for 1.50×10^{18} and 3.91×10^{19} FLOPs, respectively. In a FLOP-controlled setting, BoV attains a lower validation loss on a held-out split of roughly 41.9M tokens from the ClimbMix dataset (Diao et al., 2025) and a higher average score across the 21 benchmarks of the DCLM CORE suite (Li et al., 2024) than an otherwise identical model with the standard attention mechanism. Moreover, compared with existing methods that similarly fuse original token information into value vectors, BoV achieves better performance with

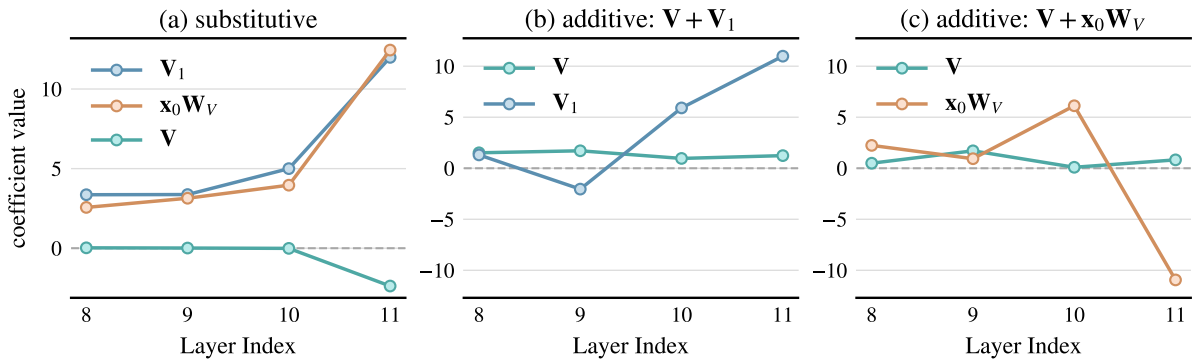

Figure 2: Value of the unbounded, learnable coefficient for each component of the value vector at each target layer of the 12-layer model, for the substitutive (a) and additive (b, c) variants.

less memory and fewer FLOPs per token.

2 Preliminaries

We establish notation for a single forward pass of a transformer model, omitting the batch dimension for clarity. A model embeds a length-T token sequence into $\mathbf{e} \in \mathbb{R}^{T\times d}$, where d is the hidden dimension. Its normalized form $\mathbf{x}_0 = \mathrm{RMSNorm}(\mathbf{e})$, which we call the initial token embedding, is the input to the first transformer block and carries no context information on its own.

Transformer blocks are joined by residual connections (He et al., 2016). The input $\mathbf{x}_i$ to the i-th block's attention layer, which we call the aggregate residual stream, is the RMSNorm of the sum of the initial token embedding $\mathbf{x}_0$ and the outputs of all preceding layers. Unlike the context-free $\mathbf{x}_0$, the residual stream $\mathbf{x}_i$ accumulates information from other tokens through the attention layers below it, and is therefore context-dependent.

The attention layer projects query, key, and value vectors from $\mathbf{x}_i$,

$$\mathbf{Q} = \mathbf{x}_i\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{x}_i\mathbf{W}_K, \quad \mathbf{V} = \mathbf{x}_i\mathbf{W}_V, \tag{1}$$

where $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times d}$.

Each of $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ is split into H heads of dimension $d_h = d/H$, and each head h computes scaled dot-product attention,

$$\mathbf{O}^{(h)} = \mathrm{softmax}\left(\frac{\mathbf{Q}^{(h)}\mathbf{K}^{(h)\top}}{\sqrt{d_h}}\right)\mathbf{V}^{(h)}. \tag{2}$$

The per-head outputs are concatenated and projected by $\mathbf{W}_O \in \mathbb{R}^{d\times d}$ into the attention output $\mathbf{a}_i = \mathrm{concat}_{h=1}^{H}(\mathbf{O}^{(h)})\,\mathbf{W}_O$, which is added back to the residual stream, updating each token's representation with information from other tokens.

	Output	Target	Coeff.	Val. loss
1	$\mathbf{V}$	—	—	0.854
2	$\mathbf{V}$	Last 4	γ_v	0.858
3	$\mathbf{V}_1$	All 12	1	0.872
4	$\mathbf{V}_1$	Every 2	γ_v	0.851
5	$\mathbf{V}_1$	Last 4	1	0.849
6	$\mathbf{V}_1$	Last 4	γ_v	0.847
7	$\mathbf{x}_0\mathbf{W}_V$	Every 2	γ_v	0.849
8	$\mathbf{x}_0\mathbf{W}_V$	Last 4	1	0.847
9	$\mathbf{x}_0\mathbf{W}_V$	Last 4	γ_v	**0.845**
10	$\mathbf{V}+\mathbf{V}_1$	Last 4	0.5	0.849
11	$\mathbf{V}+\mathbf{V}_1$	Last 4	γ_v	0.848
12	$\mathbf{V}+\mathbf{x}_0\mathbf{W}_V$	Last 4	0.5	0.848
13	$\mathbf{V}+\mathbf{x}_0\mathbf{W}_V$	Last 4	γ_v	**0.845**

Table 1: Validation loss (BPB) on the 12-layer, 135M model across variants. γ_v indicates an unbounded, learnable coefficient.

3 Motivation

In this work, we comprehensively study to what extent value vectors in attention layers need context information from the residual stream, and to what extent they instead benefit from the original token information. We provide the model with two context-free sources of original token information, which we collectively denote $\tilde{\mathbf{V}}$: a value vector $\tilde{\mathbf{V}} = \mathbf{x}_0\mathbf{W}_V$ computed from the initial token embedding $\mathbf{x}_0$ with the target layer's value matrix $\mathbf{W}_V$, or the value vector $\tilde{\mathbf{V}} = \mathbf{V}_1$ from the first attention layer. In each variant, $\tilde{\mathbf{V}}$ either replaces or is linearly interpolated with the standard value vector $\mathbf{V} = \mathbf{x}_i\mathbf{W}_V$ computed from the residual stream.

Our exploration spans four axes. (1) **Additive or substitutive.** We study whether the model benefits more from using $\tilde{\mathbf{V}}$ directly, which substitutes $\mathbf{V}$ entirely and supplies only context-free, original

token information, or from a weighted combination of $\tilde{\mathbf{V}}$ and $\mathbf{V}$, which mixes context-free and context-dependent information. Unless otherwise noted, all variants modify only the last third of attention layers. (2) **Learnable or fixed coefficient.** We study whether each component of the value vector is best scaled by an unbounded, learnable coefficient or by a fixed one. In the substitutive variants, the fixed coefficient for $\tilde{\mathbf{V}}$ is 1; in the additive variants, the fixed coefficient for both $\tilde{\mathbf{V}}$ and $\mathbf{V}$ is 0.5. The magnitudes of the learned coefficients reveal how strongly the model prefers $\tilde{\mathbf{V}}$ over $\mathbf{V}$. (3) **Source of original token information.** We study using $\mathbf{x}_0\mathbf{W}_V$, which is unique to each target layer, or $\mathbf{V}_1$, which is shared across all target layers, as $\tilde{\mathbf{V}}$. (4) **Target layers.** We study the effect of targeting the last third of layers, every other layer, or every layer.

We conduct all experiments on a 12-layer, 135M transformer model in the style of GPT-2 (Radford et al., 2019), using a simplified implementation of nanochat (Karpathy, 2025). We train every variant under a compute-controlled budget of 1.50×10^{18} FLOPs, which yields 1.97B tokens for the standard attention baseline.[1] All runs use a fixed global batch size of 524,288 tokens and a sequence length of 2048. We use the Muon optimizer for all matrix parameters with a learning rate of 0.02, and the AdamW optimizer for the remaining parameters, following nanochat's standard practices. A complete training configuration is given in Appendix B.

From Table 1 and Figure 2, we make the following observations about the relative contributions of $\mathbf{V}$ and $\tilde{\mathbf{V}}$ to the value vector.

(i) **Deep layers prefer a larger share of $\tilde{\mathbf{V}}$ over $\mathbf{V}$.** Figure 2(b, c) shows that, in the two additive variants, the learned coefficients for $\tilde{\mathbf{V}}$ are much larger in absolute value than those for $\mathbf{V}$. Several layers drive the coefficient of $\mathbf{V}$ toward 0, giving it minimal influence, while the final layer pushes the magnitude of $\tilde{\mathbf{V}}$'s coefficient past 10. When the value vector is computed from a single source, Figure 2(a) shows that the coefficients for $\tilde{\mathbf{V}}$ stay consistently above 2 and rise past 10 in the final layer, while the coefficient for $\mathbf{V}$ is driven toward 0 or negative.

(ii) **Only deep layers, not shallow ones, benefit from substituting $\mathbf{V}$ with $\tilde{\mathbf{V}}$.** With a fixed coefficient of 1, substituting $\mathbf{V}$ with $\mathbf{V}_1$ in every layer (row 3) leads to a higher loss than the standard attention baseline (row 1), whereas restricting the substitution to the last third of layers achieves significantly lower loss (row 5). With a learnable coefficient, substituting $\mathbf{V}$ with $\mathbf{V}_1$ (row 4) or $\mathbf{x}_0\mathbf{W}_V$ (row 7) in every other layer still lags behind substituting in the last third of layers (rows 6 and 9). Substituting $\mathbf{V}$ with $\tilde{\mathbf{V}}$ therefore provides little benefit in the shallow layers but a meaningful improvement in the deep ones.

(iii) **Once deep layers have access to $\tilde{\mathbf{V}}$, adding $\mathbf{V}$ yields little improvement.** With learnable coefficients, computing the value vector directly as $\mathbf{V}_1$ (row 6) achieves a slightly lower loss than $\mathbf{V}+\mathbf{V}_1$ (row 11), while computing it directly as $\mathbf{x}_0\mathbf{W}_V$ (row 9) matches the loss of $\mathbf{V}+\mathbf{x}_0\mathbf{W}_V$ (row 13). This corroborates observation (i): the contribution of $\mathbf{V}$ is minimized at deep layers in the additive variants.

(iv) **Deep layers benefit from a layer-specific value vector rather than a shared one.** With both a fixed and a learnable coefficient, using a layer-specific $\mathbf{x}_0\mathbf{W}_V$ as $\tilde{\mathbf{V}}$ (rows 8 and 9) consistently outperforms using a shared $\mathbf{V}_1$ (rows 5 and 6). This is consistent with $\tilde{\mathbf{V}} = \mathbf{x}_0\mathbf{W}_V$ being strictly more expressive than $\tilde{\mathbf{V}} = \mathbf{V}_1$: although both are computed from the same input $\mathbf{x}_0$, multi-head attention lets each layer's value matrix $\mathbf{W}_V$ project $\mathbf{x}_0$ into a different low-rank subspace, which a single shared projection cannot reproduce.

Overall, when restricted to the last third of layers, computing the value vector directly from a layer-specific $\mathbf{x}_0\mathbf{W}_V$ provides the largest gain, and the role of $\mathbf{V}$ is marginal. This shows that value vectors in deep layers benefit primarily from context-free, original token information, an insight we build on in our architecture design in Section 4.

4 Bank of Values: Persisting Context-Free Value Vectors as Model Parameters

From Section 3, value vectors in deep attention layers are most effective when computed directly from $\mathbf{x}_0\mathbf{W}_V$. Importantly, since both $\mathbf{x}_0$ and $\mathbf{W}_V$ depend only on model parameters, the value vector that a token with id i produces,

$$\mathbf{v}_i = \mathrm{RMSNorm}(\mathbf{e}_i)\,\mathbf{W}_V, \tag{3}$$

is deterministic and independent of the preceding sequence, where $\mathbf{e}_i$ is the embedding of the token

[1] This budget is cost-effective: its validation loss, measured in bits per byte (BPB), is only 0.06 BPB higher than that of training on 40B tokens.

```python
def bov_value(idx: Tensor, VE: Tensor, gamma_v: Tensor) -> Tensor:
    # idx: [B, T] token ids        VE: [vocab, H*D] per-layer table
    return gamma_v * VE[idx]                        # [B, T, H, D]


def forward(self, x: Tensor, idx: Tensor, cos_sin: tuple[Tensor, Tensor],
            kv_cache: KVCache | None = None) -> Tensor:
    q = norm(apply_rope(self.c_q(x), cos_sin))
    k = norm(apply_rope(self.c_k(x), cos_sin))

    if kv_cache is None:
        # training / prefill: value comes from the table, not from x
        v = bov_value(idx, self.VE, self.gamma_v)
        return flash_attn(q, k, v, causal=True)

    # ---- inference: cache K, but store token ids in place of V ----
    p = kv_cache.pos
    kv_cache.k[:, p:p+T]   = k
    kv_cache.idx[:, p:p+T] = idx
    # keys: read the full history from the K cache
    k_hist = kv_cache.k[:, :p+T]
    # values: rebuild from stored ids via table lookup (no V cache)
    v_hist = bov_value(kv_cache.idx[:, :p+T], self.VE, self.gamma_v)
    return flash_attn(q, k_hist, v_hist, causal=True)
```

Figure 3: PyTorch-style pseudocode for Bank of Values. `bov_value()` performs an $O(Td)$ lookup of all past value vectors, scaled by a coefficient `gamma_v`. `forward()` stores past token indices in `kv_cache.idx` and uses them to copy value vectors into `v_hist` for dot-product attention.

with id i. We can therefore learn each $\mathbf{v}_i$ directly as a model parameter, stored at each target layer in a value table

$$\mathbf{E}_v \in \mathbb{R}^{|\mathcal{V}| \times d}, \qquad \mathbf{E}_v[i] = \mathbf{v}_i, \tag{4}$$

where $|\mathcal{V}|$ is the vocabulary size and d the hidden dimension. During a forward pass, the value vector of each token in the last third of layers is looked up directly from $\mathbf{E}_v$. Scaling each looked-up value by an unbounded, learnable per-layer coefficient $\gamma_v \in \mathbb{R}$, we obtain an attention variant that replaces the dense value matrix $\mathbf{W}_V$ with a sparse lookup table $\mathbf{E}_v$, computing the value vector at position p as

$$\mathbf{v}_p = \gamma_v \, \mathbf{E}_v[i_p], \tag{5}$$

where i_p is the id of the token at position p. We call this attention variant **Bank of Values** (BoV). The architecture of BoV is shown in Figure 1, alongside standard attention and SVFormer (Zhou et al., 2024).

Inference. In standard attention, the model caches the value vectors of all previous tokens during decode to avoid recomputing them, reducing the per-step cost of producing past values from $O(Td^2)$ to $O(d^2)$. In BoV, by contrast, the value vector of each token is looked up directly from $\mathbf{E}_v$, so no projection or recomputation is ever required, and the value vectors of all T preceding tokens are retrieved by a single `gather` in $O(Td)$ time. When a target layer computes attention, the model gathers the value vectors of all preceding tokens from $\mathbf{E}_v$ and discards them as soon as the layer finishes. Across the target layers, therefore, at most one layer materializes the full set of value vectors at any moment, in contrast to standard attention, which must keep a value cache for every layer at all times.

Memory overhead. Because BoV persists the value vectors of all vocabulary tokens as the parameter table $\mathbf{E}_v$, the fixed memory cost of $\mathbf{E}_v$ trades off predictably against the value cache of standard attention across context lengths. The net V-cache memory saved across all layers is

$$\Delta_{\mathrm{KV}} = \frac{L}{3} \, d \, (T - |\mathcal{V}|), \tag{6}$$

where L is the number of layers, T the context length, d the hidden dimension, and $|\mathcal{V}|$ the vocabulary size. As Eq. 6 shows, standard attention is more memory-efficient at shorter contexts, but beyond the breakeven length $T = |\mathcal{V}|$, BoV saves memory, with the saved fraction growing and eventually saturating. Figure 5(a) gives an overview of this tradeoff for the vocabulary sizes of two tokenizers (nanochat and Qwen3).

For the workloads of modern LLMs, we argue that BoV is preferable for two reasons. First, long-

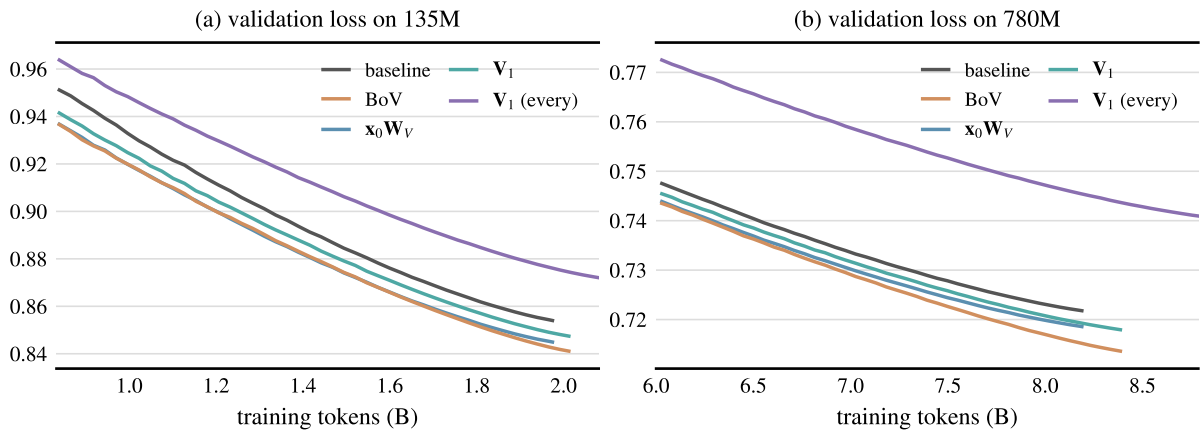

Figure 4: Validation loss over training at the (a) 135M and (b) 780M scales, for the standard attention baseline, BoV, and other attention variants that at least partially compute value vectors from a context-free component.

	Output	Layers	Coeff.	val bpb ↓	CORE ↑	ARC-E	HSwag	SQuAD	CoQA
1 (baseline)	$\mathbf{V}$	—	—	0.722	0.260	0.691	0.558	0.396	0.291
2 (**BoV**)	$\mathbf{E}_v[i]$	last 1/3	γ_v	$\mathbf{0.714}_{\uparrow .008}$	$\mathbf{0.272}_{\uparrow .012}$	$\mathbf{0.716}_{\uparrow .025}$	$\mathbf{0.574}_{\uparrow .016}$	$\mathbf{0.416}_{\uparrow .020}$	$\mathbf{0.321}_{\uparrow .030}$
Input vector variants									
3	$\mathbf{x}_0\mathbf{W}_V$	last 1/3	γ_v	$0.719_{\uparrow .003}$	$0.259_{\downarrow .001}$	$0.714_{\uparrow .023}$	$0.562_{\uparrow .004}$	0.396	$0.297_{\uparrow .006}$
4	$\mathbf{V}_1$	last 1/3	γ_v	$0.718_{\uparrow .004}$	$0.242_{\downarrow .018}$	$0.702_{\uparrow .011}$	$0.566_{\uparrow .008}$	$0.380_{\downarrow .016}$	$0.298_{\uparrow .007}$
Target layer variants									
5	$\mathbf{V}_1$	every	1	$0.741_{\downarrow .019}$	$0.210_{\downarrow .050}$	$0.677_{\downarrow .014}$	$0.516_{\downarrow .042}$	$0.230_{\downarrow .166}$	$0.229_{\downarrow .062}$

Table 2: Validation loss, aggregate CORE score, and representative benchmark scores for standard attention, BoV, and other variants, trained under an identical FLOP-controlled budget on the 780M model.

context inference is a common requirement across a wide range of LLM applications, precisely the regime in which persisting value vectors as parameters becomes advantageous. Second, the lookup into $\mathbf{E}_v$ requires only the token indices i_p, which are available well before the forward pass reaches the target layer; $\mathbf{E}_v$ can therefore be offloaded to host memory by default and prefetched on demand, improving memory utilization.

We provide PyTorch-style pseudocode for BoV in Figure 3.

5 Experiments

Model architecture and training setup. We train both BoV and the standard attention baseline at the 24-layer, 780M scale in the style of nanochat (Karpathy, 2025). We fix the compute budget for all experiments at 3.91×10^{19} FLOPs, which corresponds to 8.19B tokens for the baseline and 8.39B for BoV, owing to its lower FLOPs per token. As in Section 3, we use the Muon optimizer with learning rate 0.02 on all matrix parameters, and the AdamW optimizer with learning rate 0.15 on the value vector table $\mathbf{E}_v$ in BoV. The global batch size is fixed at 1,048,576 tokens with a sequence length of 2048. A complete training configuration can be found in Appendix B. We additionally train both architectures at the 12-layer, 135M scale, following the recipe in Section 3.

	HSwag	LAMBADA	CSQA	BoolQ
$\mathbf{E}_v[i]$	**0.574**	**0.451**	**0.324**	**0.635**
$\mathbf{V} + \mathbf{E}_v[i]$	**0.574**	0.433	0.259	0.594

Table 3: Comparison on four representative benchmarks between BoV (top), which retrieves value vectors as $\mathbf{E}_v[i]$, and an additive variant (bottom), which adds the retrieved values to the standard value vector as $\mathbf{V} + \mathbf{E}_v[i]$, on the 780M model. The better result in each column is in **bold**.

To avoid introducing any inductive bias through BoV's initialization, we initialize each target layer's table by copying $\mathbf{x}_0\,\mathbf{W}_V$, evaluated at each token, into the corresponding row $\mathbf{E}_v[i]$, so that BoV's first forward pass is numerically identical to an attention layer that computes its value vectors from the initial token embedding alone, as

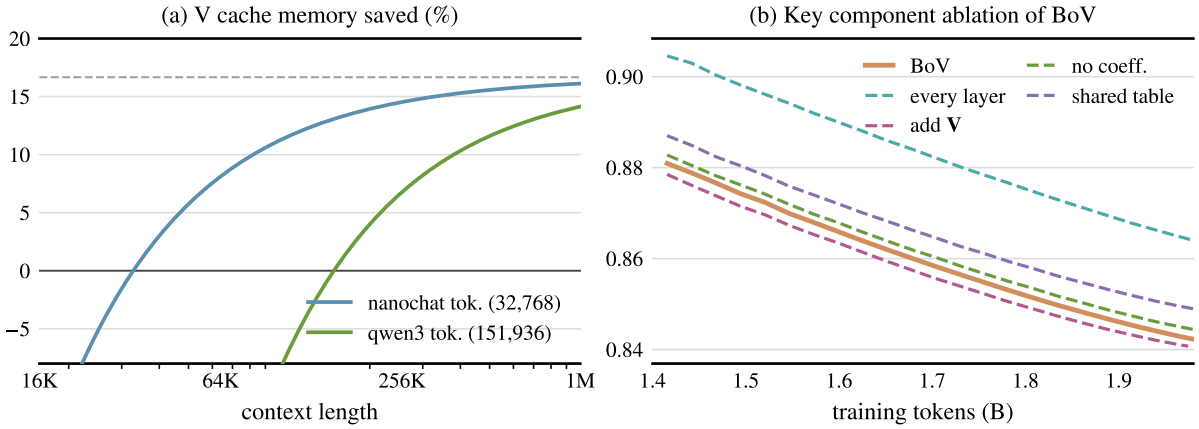

Figure 5: (a) Fraction of V cache memory saved by Bank of Values over standard attention as a function of context length, for the nanochat and Qwen3 tokenizers; and (b) ablation of BoV's design components on the 135M model.

$\mathbf{x}_0 \mathbf{W}_V$.

Evaluation setup. For the 24-layer variants, we evaluate on 21 benchmarks from the DCLM CORE suite (Li et al., 2024): Jeopardy (Li et al., 2024), ARC-Easy and ARC-Challenge (Clark et al., 2018), COPA (Roemmele et al., 2011), CommonsenseQA (Talmor et al., 2019), PIQA (Bisk et al., 2020), OpenBookQA (Mihaylov et al., 2018), LAMBADA (Paperno et al., 2016), HellaSwag (Zellers et al., 2019), Winograd (Levesque et al., 2012), Winogrande (Sakaguchi et al., 2019), AGIEval LSAT-AR (Zhong et al., 2023), SQuAD (Rajpurkar et al., 2016), CoQA (Reddy et al., 2019), BoolQ (Clark et al., 2019), and the BIG-bench QA WikiData, Language Identification, Dyck Languages, CS Algorithms, Operators, and Repeat-Copy-Logic tasks (Srivastava et al., 2022).

Following DCLM, we report a centered accuracy for each task, $(\mathrm{acc} - r)/(1 - r)$ where r is the task's random-guessing (or majority-class) baseline, and define the aggregate CORE score as the unweighted mean of the 21 centered accuracies. For both the 12-layer and 24-layer variants, we additionally report bits-per-byte validation loss on a held-out split of the ClimbMix pretraining corpus, computed over approximately 41.9M tokens. The main results, including scores on four representative benchmarks, are reported in Table 2, and the full benchmark results in Table 4.

5.1 Main Results

As shown in Figure 4(a, b), BoV outperforms the standard attention baseline on validation loss at both the 135M and 780M scales. It also performs favorably on most benchmarks (Tables 2 and 4), with the largest gains on the reading-comprehension (SQuAD, CoQA) and symbolic-problem-solving (BIG-bench) tasks, while matching the baseline on most commonsense-reasoning and language-understanding tasks. These results align with the intuition that computing attention with context-free value vectors particularly benefits tasks that rely on fewer contextual cues, while remaining competitive on the rest.

5.2 Comparison with Prior Methods

We compare BoV with the prior methods discussed in Section 3 that compute the value vector at least partially from a context-free component $\tilde{\mathbf{V}}$ (Table 2). At the 780M scale, none of these methods outperforms the standard attention baseline on the aggregate CORE score, indicating limited scalability.

In Table 3, we also compare BoV with a popular additive variant (Jordan and contributors, 2024; snimu, 2025) that adds a value-embedding lookup $\mathbf{E}_v[i]$ to the standard value vector computed from $\mathbf{x}_i$. We follow their original implementation, using input-dependent, bounded scaling coefficients and targeting every second layer. BoV performs favorably on most benchmarks while attaining a similar validation loss. The full comparison is reported in Table 5.

5.3 Component Design

Finally, we ablate each key component of BoV on the 12-layer model and report the impact on validation loss in Figure 5(b).

Sharing a single table across layers. Sharing a single $\mathbf{E}_v$ across all target layers increases validation loss, showing the importance of learning a layer-specific table of value vectors.

Fixing the coefficient at 1. Removing the learnable coefficient and fixing it at 1 also increases validation loss, indicating the benefit of scaling the attention output of the target layers.

Targeting every layer. Learning an $\mathbf{E}_v$ at every layer rather than only the last third increases validation loss significantly, corroborating the insight that only deep layers benefit substantially from context-free value vectors.

Retaining the standard value $\mathbf{V}$. Preserving the standard value vector $\mathbf{V}$ computed from the residual stream yields only a marginal improvement over BoV, showing that a context-invariant value vector is sufficient in deeper layers.

6 Related Work

Adding early-layer information to value vectors. Several methods inject early-layer information into the value vectors of deeper layers. Value Residual Learning adds or shares the first layer's values (Zhou et al., 2024), SkipV1Former reuses its value heads (Wu et al., 2025), and modded-nanogpt (Jordan and contributors, 2024) and MoVE (Li, 2026) add a per-layer token value-embedding table to the value vector; earlier sequence models similarly let layers attend to combinations of previous representations (Bapna et al., 2018). Preserving early-layer information is also motivated by over-smoothing in deep layers (Nguyen et al., 2023). BoV instead *replaces* the value vector in deep layers with a layer-specific lookup keyed by token identity.

Depth-aware transformer architectures. Another line changes how information flows across depth. DenseFormer feeds each block a learned weighted average of all previous block outputs (Pagliardini et al., 2024), Hyper-Connections replace the fixed residual connection with learnable connection weights (Zhu et al., 2025), and Attention Residuals replace the additive residual with a learned, input-dependent weighted sum over previous layer outputs (Kimi Team et al., 2026). Mixture-of-Depths Attention further lets each attention head attend to key/value pairs from preceding layers, not just the current one (Zhu et al., 2026). BoV instead modifies only the value vectors, leaving the rest of attention unchanged.

Parametric memory and lookup-based computation. A related line replaces learned matrix computations with table lookups. Persistent-memory attention augments each layer with static, learnable key/value vectors (Sukhbaatar et al., 2019); product-key memory adds a large key–value lookup layer retrieved via factored keys (Lample et al., 2019); MemoryFormer replaces a transformer's linear projection layers with hashed embedding lookups (Ding et al., 2024); and Engram offloads static information into a sparse, conditional $O(1)$ lookup (Cheng et al., 2026). In theory, one of the query, key, or value projections can even be dropped without loss of expressivity (Karbevski and Mijoski, 2025). BoV applies this idea to the value path, replacing the value projection in deep layers with a per-token learned table.

Efficient attention and KV-cache reduction. Many methods improve efficiency by optimizing attention kernels (Dao et al., 2022; Dao, 2023; Shah et al., 2024), cache layout and serving (Kwon et al., 2023), sparsifying attended tokens (Yuan et al., 2025; DeepSeek-AI, 2025), sharing or compressing KV heads (Shazeer, 2019; Ainslie et al., 2023; DeepSeek-AI, 2024; Brandon et al., 2024), quantizing the cache (Hooper et al., 2024; Liu et al., 2024), or pruning cached layers and states (Sun et al., 2024; Wu and Tu, 2024; Shen et al., 2025); others eliminate the V cache by recomputing values from the residual stream on demand (Qasim et al., 2026). These change how attention is computed or which states are kept; BoV instead removes the need to compute or store deep-layer values at all, and is composable with them.

7 Conclusion

In this work, we systematically study the value of adding context-free, original token information to value vectors. We find that deeper layers benefit largely from a context-free value vector that does not draw on the residual stream, which leads us to propose Bank of Values, an attention variant that persists value vectors directly as model parameters. Across downstream benchmarks and validation loss, Bank of Values outperforms the standard attention baseline and scales better than competing variants, while using less memory and fewer FLOPs.

Limitations

Our study has several limitations. First, owing to compute constraints, we validate Bank of Values only at the 135M and 780M scales under FLOP-controlled budgets; confirming that the gains persist for substantially larger models trained on more tokens is left to future work. Second, our findings are empirical: we observe that only deeper layers benefit from a context-free value vector, but we do not fully characterize the mechanism behind this depth dependence or the optimization objective it serves, which we believe is an important direction for understanding the role of value vectors in attention. Finally, the memory benefit of Bank of Values is context-length dependent: it stores a fixed value table of size $O(|\mathcal{V}|\, d)$ per target layer and only saves memory beyond a breakeven context length, so its advantage is realized primarily in long-context settings.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of EMNLP*.

Ankur Bapna, Mia Xu Chen, Orhan Firat, Yuan Cao, and Yonghui Wu. 2018. Training deeper neural machine translation models with transparent attention. In *Proceedings of EMNLP*.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

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

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, Han Zhang, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. 2026. Conditional Memory via Scalable Lookup: A New Axis of Sparsity for Large Language Models. *Preprint*, arXiv:2601.07372.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *Proceedings of NAACL-HLT*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *Preprint*, arXiv:1803.05457.

Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. *Preprint*, arXiv:2307.08691.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*.

DeepSeek-AI. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. *Preprint*, arXiv:2405.04434.

DeepSeek-AI. 2025. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. *Preprint*, arXiv:2512.02556.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. 2025. CLIMB: CLustering-based Iterative Data Mixture Bootstrapping for Language Model Pre-training. *Preprint*, arXiv:2504.13161.

Ning Ding, Yehui Tang, Haochen Qin, Zhenli Zhou, Chao Xu, Lin Li, Kai Han, Heng Liao, and Yunhe Wang. 2024. MemoryFormer: Minimize transformer computation by removing fully-connected layers. *Preprint*, arXiv:2411.12992.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. 2024. Patchscopes: A unifying framework for inspecting hidden representations of language models. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. In *Advances in Neural Information Processing Systems*.

Keller Jordan and contributors. 2024. modded-nanogpt. GitHub repository.

Marko Karbevski and Antonij Mijoski. 2025. Key and value weights are probably all you need: On the necessity of the query, key, value weight triplet in self-attention transformers. *Preprint*, arXiv:2510.23912.

Andrej Karpathy. 2025. nanochat: The best ChatGPT that $100 can buy. GitHub repository.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, and 1 others. 2026. Attention Residuals. *Preprint*, arXiv:2603.15031.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM Symposium on Operating Systems Principles*.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2019. Large memory layers with product keys. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The Winograd Schema Challenge. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, and 1 others. 2024. DataComp-LM: In Search of the Next Generation of Training Sets for Language Models. In *Advances in Neural Information Processing Systems*.

Yangyan Li. 2026. MoVE: Mixture of Value Embeddings – A New Axis for Scaling Parametric Memory in Autoregressive Models. *Preprint*, arXiv:2601.22887.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *International Conference on Machine Learning*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Tam Nguyen, Tan M. Nguyen, and Stanley J. Osher. 2023. Mitigating Over-smoothing in Transformers via Regularized Nonlocal Functionals. *Preprint*, arXiv:2312.00751.

Matteo Pagliardini, Amirkeivan Mohtashami, François Fleuret, and Martin Jaggi. 2024. DenseFormer: Enhancing Information Flow in Transformers via Depth Weighted Averaging. *Preprint*, arXiv:2402.02622.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA Dataset: Word Prediction Requiring a Broad Discourse Context. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Kaleem Ullah Qasim, Jiashu Zhang, Muhammad Kafeel Shaheen, Razan Alharith, and Heying Zhang. 2026. The residual stream is all you need: On the redundancy of the KV cache in transformer inference. *Preprint*, arXiv:2603.19664.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI Technical Report*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A Conversational Question Answering Challenge. *Transactions of the Association for Computational Linguistics*.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. 2011. Choice of Plausible Alternatives: An Evaluation of Commonsense Causal Reasoning. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Communications of the ACM*.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-Precision. *Preprint*, arXiv:2407.08608.

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

Yiqun Shen, Song Yuan, Zhengze Zhang, Xiaoliang Wang, Daxin Jiang, and Cam-Tu Nguyen. 2025. LAVa: Layer-wise KV Cache Eviction with Dynamic Budget Allocation. *Preprint*, arXiv:2509.09754.

snimu. 2025. modded-nanogpt: Analyzing value-embedding-, UNet-, and x0-lambdas. Blog post.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, and 1 others. 2022. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *Preprint*, arXiv:2206.04615.

Sainbayar Sukhbaatar, Edouard Grave, Guillaume Lample, Hervé Jégou, and Armand Joulin. 2019. Augmenting self-attention with persistent memory. *Preprint*, arXiv:1907.01470.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2024. You only cache once: Decoder-decoder architectures for language models. *Preprint*, arXiv:2405.05254.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In *Proceedings of NAACL-HLT*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*.

Haoyi Wu and Kewei Tu. 2024. Layer-condensed KV cache for efficient inference of large language models. In *Proceedings of ACL*.

Zhoutong Wu, Yuan Zhang, Yiming Dong, Chenheng Zhang, Cong Fang, Kun Yuan, and Zhouchen Lin. 2025. Improving Model Representation and Reducing KV Cache via Skip Connections with First Value Heads. *Preprint*, arXiv:2510.16807.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention. *Preprint*, arXiv:2502.11089.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models. *Preprint*, arXiv:2304.06364.

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, and Zhenzhong Lan. 2024. Value Residual Learning For Alleviating Attention Concentration In Transformers. *Preprint*, arXiv:2410.17897.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. 2025. Hyper-Connections. In *International Conference on Learning Representations*.

Lianghui Zhu, Yuxin Fang, Bencheng Liao, Shijie Wang, Tianheng Cheng, Zilong Huang, Chen Chen, Lai Wei, Yutao Zeng, Ya Wang, Yi Lin, Yu Li, and Xinggang Wang. 2026. Mixture-of-Depths Attention. *Preprint*, arXiv:2603.15619.

	Baseline	BoV
val bpb ↓	0.722	**0.714**$_{\uparrow .008}$
CORE ↑	0.260	**0.272**$_{\uparrow .012}$
Reading comprehension		
SQuAD	0.40	**0.42**$_{\uparrow .02}$
CoQA	0.29	**0.32**$_{\uparrow .03}$
BoolQ	0.62	**0.64**$_{\uparrow .02}$
Symbolic problem solving		
BIG-bench Dyck	0.12	**0.15**$_{\uparrow .03}$
AGIEval LSAT-AR	0.27	**0.29**$_{\uparrow .02}$
BIG-bench CS-Alg.	0.41	**0.43**$_{\uparrow .02}$
BIG-bench Operators	0.18	**0.20**$_{\uparrow .02}$
BIG-bench Rep.-Copy	0.00	**0.06**$_{\uparrow .06}$
World knowledge		
Jeopardy	0.08	**0.12**$_{\uparrow .04}$
BIG-bench WikiData	**0.49**	0.47$_{\downarrow .02}$
ARC-Easy	0.69	**0.72**$_{\uparrow .03}$
ARC-Challenge	0.40	0.40
Commonsense reasoning		
COPA	**0.67**	0.66$_{\downarrow .01}$
CommonsenseQA	**0.33**	0.32$_{\downarrow .01}$
PIQA	0.74	0.74
OpenBookQA	0.38	0.38
Language understanding		
LAMBADA	0.43	**0.45**$_{\uparrow .02}$
HellaSwag	0.56	**0.57**$_{\uparrow .01}$
Winograd	**0.66**	0.65$_{\downarrow .01}$
Winogrande	0.56	0.56
BIG-bench LangID	0.26	0.26

Table 4: Full benchmark comparison of the **Baseline** and **BoV** on the 780M model across all 21 DCLM CORE tasks, grouped into the five DCLM categories, plus validation BPB and the aggregate CORE score.

A Additional Results

Tables 4 and 5 report the full benchmark results behind Tables 2 and 3.

B Additional Experimental Details

Table 6 reports the full training configuration for both model scales.

	$\mathbf{V} + \mathbf{E}_v[i]$	$\mathbf{E}_v[i]$
val bpb ↓	**0.712**	$0.714_{\downarrow .002}$
CORE ↑	0.268	$\mathbf{0.272}_{\uparrow .004}$
Reading comprehension		
SQuAD	**0.44**	$0.42_{\downarrow .02}$
CoQA	0.31	$\mathbf{0.32}_{\uparrow .01}$
BoolQ	0.59	$\mathbf{0.64}_{\uparrow .05}$
Symbolic problem solving		
BIG-bench Dyck	**0.17**	$0.15_{\downarrow .02}$
AGIEval LSAT-AR	0.27	$\mathbf{0.29}_{\uparrow .02}$
BIG-bench CS-Alg.	0.43	0.43
BIG-bench Operators	0.17	$\mathbf{0.20}_{\uparrow .03}$
BIG-bench Rep.-Copy	0.03	$\mathbf{0.06}_{\uparrow .03}$
World knowledge		
Jeopardy	0.11	$\mathbf{0.12}_{\uparrow .01}$
BIG-bench WikiData	**0.50**	$0.47_{\downarrow .03}$
ARC-Easy	0.72	0.72
ARC-Challenge	0.40	0.40
Commonsense reasoning		
COPA	0.66	0.66
CommonsenseQA	0.26	$\mathbf{0.32}_{\uparrow .06}$
PIQA	**0.75**	$0.74_{\downarrow .01}$
OpenBookQA	**0.39**	$0.38_{\downarrow .01}$
Language understanding		
LAMBADA	0.43	$\mathbf{0.45}_{\uparrow .02}$
HellaSwag	0.57	0.57
Winograd	**0.69**	$0.64_{\downarrow .05}$
Winogrande	**0.57**	$0.56_{\downarrow .01}$
BIG-bench LangID	0.25	$\mathbf{0.26}_{\uparrow .01}$

Table 5: Full benchmark comparison of the add form ($\mathbf{V} + \mathbf{E}_v[i]$) and the replace form of BoV ($\mathbf{E}_v[i]$) on the 780M model across all 21 DCLM CORE tasks, grouped into the five DCLM categories, plus validation BPB and the aggregate CORE score.

	135M (12-layer)	780M (24-layer)
Architecture		
Layers	12	24
Model dimension d	768	1536
Attention heads (H / KV)	6 / 6	12 / 12
Head dimension d_h	128	128
Vocabulary size $\|\mathcal{V}\|$	32,768	32,768
Window pattern	SSSL	SSSL
Sequence length T	2048	2048
Parameters	135.3M	780.1M
Training budget		
Total batch size (tokens)	524,288 (2^{19})	1,048,576 (2^{20})
Device batch size	16	4
Gradient accumulation (4 GPUs)	4	32
Iterations	3,766	7,809
Total tokens	1.97B	8.19B
Target FLOPs	1.50×10^{18}	3.91×10^{19}
FLOPs per token	7.60×10^{8}	4.78×10^{9}
Tokens-to-parameters ratio	17.9	11.2
Compute dtype	bf16	bf16
Optimization		
Optimizer	Muon (matrices) + AdamW (rest)	
Matrix LR (Muon)	0.02	0.0283
Embedding LR	0.30	0.30
Unembedding LR	0.008	0.008
Scalar LR	0.50	0.707
Weight decay	0.28	0.0597
LR schedule	warmup 40 steps; warmdown from 0.65; final frac. 0.05	

Table 6: Training hyperparameters for the 135M (12-layer) and 780M (24-layer) models. The 780M learning rates and weight decay are scaled from the 135M values following batch-size ($\times\sqrt{2}$) and width scaling rules.