MIDUS: MEMORY-INFUSED DEPTH UP-SCALING

Anonymous authors
Paper under double-blind review

ABSTRACT

Depth Up-Scaling (DUS) expands pre-trained language models by inserting additional Transformer blocks, but duplicating FFN-heavy structures increases both parameter count and dense per-token computation. Sparse retrieval offers an alternative by decoupling stored capacity from dense computation, but efficient scaling requires avoiding large memory overhead while preserving the heterogeneous representations of individual attention heads. We propose *Memory-Infused Depth Up-Scaling* (MIDUS), which organizes memory around attention heads using distinct product-key spaces and shared value storage with head-specific projections. Under common retrieval weights, we show that shared value realization incurs a loss gap when different heads require different outputs, while MIDUS recovers the independent optimum under a shared factorization. Empirically, MIDUS remains competitive with strong DUS baselines while substantially reducing cost. On Llama-3.1-8B, MIDUS uses 8.3× fewer trainable parameters and 42% less peak training GPU memory than OpT-DeUS, together with higher generation throughput. Head-importance analysis further shows a more concentrated distribution across attention heads, consistent with the proposed head-wise memory design. [1]

1 INTRODUCTION

Large language models (LLMs) exhibit predictable performance gains as parameters and data scale, but training larger architectures from scratch remains prohibitively expensive. Model up-scaling offers a practical alternative by expanding a pre-trained backbone and adapting the resulting model through continual pre-training (CPT). Among up-scaling strategies, Depth Up-Scaling (DUS) increases model capacity by inserting additional Transformer blocks into a pre-trained model (Gong et al., 2019; Yang et al., 2020; Wu et al., 2024). Recent work improves DUS by studying which blocks to copy, where to insert them, and how to initialize their parameters (Wu et al., 2024; Yang et al., 2025; Cao et al., 2025). Despite these advances, the form of the added capacity remains largely unchanged. Additional depth is still realized through Transformer blocks, replicating both attention and FFN layers. In particular, duplicating FFN-heavy structures increases both parameter count and dense computation as the model grows deeper.

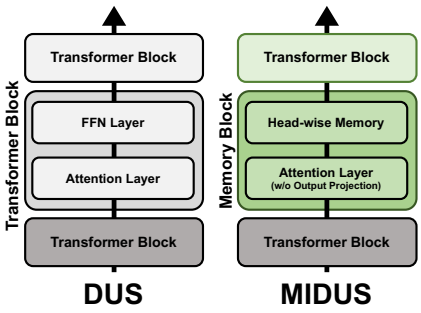

Figure 1: DUS expands model depth by duplicating Transformer blocks, while MIDUS inserts Memory blocks that perform head-wise retrieval through head-specific key spaces and value realization.

This motivates a different perspective on DUS. Rather than only asking how additional Transformer blocks should be selected or initialized, we ask whether added capacity should remain tied to full Transformer replication. Memory layers provide an alternative scaling mechanism by increasing stored capacity while retrieving only a small subset for each token (Lample et al., 2019; Kim & Jung, 2020; Huang et al., 2025). They therefore allow stored capacity to grow without introducing the same dense per-token computation as duplicated FFNs. We explore this property as an alternative source of capacity for the modules introduced during DUS.

[1] Code: `https://anonymous.4open.science/r/midus_hml`

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.

Introducing memory raises a second question of how this capacity should be organized within a Transformer. Before the output projection, multi-head attention produces multiple head-specific representations, and prior work shows that attention heads specialize in different contextual and linguistic patterns and vary in importance across layers, inputs, and tasks (Michel et al., 2019; Voita et al., 2019; Clark et al., 2019b; Fernandez et al., 2024; Yin & Steinhardt, 2025; Zhu et al., 2025). This motivates preserving head-specific memory access. However, maintaining a fully independent value bank for every head would substantially increase memory parameters. We therefore seek a memory organization that preserves head-specific access while limiting parameter growth.

Based on this perspective, we propose *Memory-Infused Depth Up-Scaling* (MIDUS). MIDUS replaces the FFN-heavy expansion of conventional DUS with sparse memory organized around individual attention heads. Each head directly uses its attention representation to query a distinct product-key space, while values are stored in a shared bank and transformed through head-specific projections. This design preserves head-specific memory access and value realization without maintaining an independent value bank for every head.

We further analyze head-wise value realization under fixed retrieval. Our analysis shows that shared value realization incurs a loss gap when different heads require different outputs, while MIDUS recovers the independent head-specific optimum under a shared factorization. This isolates the role of head-specific value realization from differences in retrieval. Empirically, MIDUS provides a favorable balance between performance and efficiency under CPT and supervised fine-tuning (SFT). Across general-purpose, mathematical, retrieval-heavy, and long-context evaluations, MIDUS remains competitive with strong DUS baselines while substantially reducing parameter, memory, and computational costs. We further observe a more concentrated distribution of head importance after CPT, consistent with organizing the added memory capacity around individual attention heads.

Our contributions are threefold. First, we introduce MIDUS, which rethinks the form of added capacity in DUS by replacing FFN-heavy expansion with sparse memory. Second, we develop a head-wise memory design that preserves head-specific retrieval and value realization while sharing value storage efficiently. Third, we provide a fixed-retrieval structural analysis together with extensive experiments demonstrating a favorable balance between performance and efficiency for DUS.

2 PRELIMINARIES

Depth up-scaling. Let $B\colon \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times d}$ denote a Transformer block, where T and d denote the sequence length and the model dimension, respectively. Given $x \in \mathbb{R}^{T\times d}$, it is defined as

$$B(x) = a(x) + \mathrm{FFN}\big(\mathrm{LN}(a(x))\big), \qquad a(x) = x + \mathrm{Attn}\big(\mathrm{LN}(x)\big). \tag{1}$$

Here, LN denotes layer normalization and Attn the attention layer. We denote the ℓ-th Transformer block by B_ℓ.Let the base model with L Transformer blocks be defined as

$$f_0 = B_{L-1} \circ \cdots \circ B_0. \tag{2}$$

DUS selects anchor indices $0 \le i_0 < \cdots < i_{D-1} \le L-1$ and interleaves D additional Transformer blocks. The placement of these blocks depends on the DUS policy. As a representative approach, Llama Pro (Wu et al., 2024) distributes anchor blocks uniformly across the model depth and inserts each duplicated block $\widetilde{B}_0, \ldots, \widetilde{B}_{D-1}$ directly after its corresponding anchor block, yielding

$$f_{\mathrm{DUS}}(x) = B_{L-1} \circ \cdots \circ \widetilde{B}_{D-1} \circ B_{i_{D-1}} \circ \cdots \circ \widetilde{B}_0 \circ B_{i_0} \circ \cdots \circ B_0(x). \tag{3}$$

Refer to Appendix A.4 for other DUS policies. Beyond simple anchor replication, recent studies such as LESA (Yang et al., 2025) and OpT-DeUS (Cao et al., 2025) explore initialization schemes tailored specifically for DUS. Section 5 provides a detailed overview of these approaches.

Linear memory layer. Given $a \in \mathbb{R}^{T\times d}$, a memory layer $\mathrm{Mem}\colon \mathbb{R}^{T\times d} \to \mathbb{R}^{T\times d}$ returns a retrieval-based representation $m = \mathrm{Mem}(a)$. It is specified by a query map and a key–value memory bank. To make this general formulation concrete, we consider a *linear* memory layer as a basic example. Specifically, define the query $q(a) = aW_q \in \mathbb{R}^{T\times d_q}$ with query projection $W_q \in \mathbb{R}^{d\times d_q}$, keys $K \in \mathbb{R}^{N\times d_q}$, and values $V \in \mathbb{R}^{N\times d}$ for N memory slots. The similarity scores between the queries and memory keys are given by $S(a) = q(a)K^\top$. For each token $t \in \{1, \ldots, T\}$, let $\Omega_t \subset \{1, \ldots, N\}$ be the Top-k index set of $S(a)_t \in \mathbb{R}^N$, with $|\Omega_t| = k$, and let $S(a)_{t,\Omega_t}$ denote

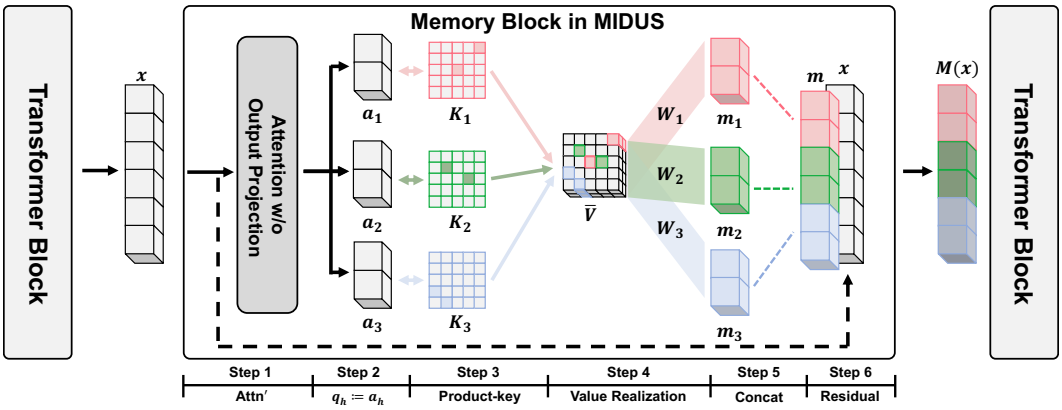

Figure 2: Overview of MIDUS with $H = 3$ attention heads and Top-k retrieval with $k = 2$. 1) The input x passes through Attn', an attention layer without the output projection, producing a'. 2) The resulting attention-head outputs are directly used as the memory queries q_h. 3) Each head performs product-key retrieval over its distinct key space and selects the Top-k memory slots. 4) The selected entries from the shared value bank $\bar{V}$ are aggregated and transformed by the head-specific projection W_h to produce m_h. 5) The head-wise outputs are concatenated to form m. 6) The resulting memory output m is added to the input x through the residual connection.

$S(a)_t$ restricted to the indices in Ω_t. We define $\alpha_t \in \Delta^{N-1}$ as the softmax over $S(a)_{t,\Omega_t}$ with zero mass outside Ω_t. Finally, each row of m is computed as

$$m_t = \sum_{j=1}^{N} \alpha_{t,j} V_j. \tag{4}$$

3 Methodology

3.1 Head-wise Memory Allocation

Conventional DUS couples additional capacity with additional dense computation by inserting FFN-heavy Transformer blocks. Sparse memory provides an alternative by increasing stored capacity while retrieving only a small subset for each token (Lample et al., 2019; Kim & Jung, 2020; Huang et al., 2025). A naive head-wise memory, however, introduces another scaling problem, as maintaining independent key and value banks for every attention head would substantially increase memory parameters.

MIDUS addresses these two considerations jointly. Motivated by the heterogeneous roles of attention heads (Michel et al., 2019; Voita et al., 2019; Clark et al., 2019b; Fernandez et al., 2024; Yin & Steinhardt, 2025; Zhu et al., 2025), MIDUS preserves head-specific memory access through distinct retrieval spaces. Product-key factorization controls the key-side cost, while shared value storage with head-specific projections reduces the value-side cost without discarding head-specific value realization.

3.2 Memory-Infused Depth Up-Scaling

MIDUS realizes the added capacity through memory organized around individual attention heads. Each attention head directly queries a distinct product-key space, while the retrieved values are realized in the corresponding head space through shared storage and head-specific projections.

Head-wise product-key retrieval. Let H denote the number of attention heads, let $d_h = d/H$, and let $[H] = \{1, \ldots, H\}$. We use the attention output before the output projection

$$a'(x) = \mathrm{Attn}'\big(\mathrm{LN}(x)\big) = \big[a_1(x) \mid \cdots \mid a_H(x)\big], \tag{5}$$

where Attn' denotes multi-head attention without the output projection and $a_h(x) \in \mathbb{R}^{T \times d_h}$ is the output of head h. MIDUS directly uses each head output as its memory query, preserving

head-specific representations while avoiding the parameters and computation of an additional query projection. Accordingly, we set $q_h := a_h(x)$.

A naive head-wise linear memory would require each head to score against all memory keys, causing both lookup cost and key parameters to grow linearly with the number of memory slots. To scale the address space efficiently, we use product-key factorization (Lample et al., 2019). We split $q_h = \left[q_h^{\mathrm{row}} \mid q_h^{\mathrm{col}}\right]$, where $q_h^{\mathrm{row}}, q_h^{\mathrm{col}} \in \mathbb{R}^{T\times d_p}$ and $d_p = d_h/2$. For each head h, we maintain distinct sub-key banks $K_h^{\mathrm{row}}, K_h^{\mathrm{col}} \in \mathbb{R}^{n\times d_p}$, whose Cartesian product defines $N = n^2$ addressable memory slots. The row and column scores are

$$S_h^{\mathrm{row}} = q_h^{\mathrm{row}} K_h^{\mathrm{row}\top}, \qquad S_h^{\mathrm{col}} = q_h^{\mathrm{col}} K_h^{\mathrm{col}\top}. \tag{6}$$

For each head h and token t, let $I_{h,t}, J_{h,t} \subset \{1,\dots,n\}$ denote the Top-k index sets of $S_{h,t}^{\mathrm{row}}$ and $S_{h,t}^{\mathrm{col}}$, respectively. Their Cartesian product forms the candidate set $\widehat{\Omega}_{h,t} = I_{h,t} \times J_{h,t}$. For each $(i,j) \in \widehat{\Omega}_{h,t}$, we define

$$\sigma_{h,t}(i,j) = S_{h,t}^{\mathrm{row}}(i) + S_{h,t}^{\mathrm{col}}(j). \tag{7}$$

Let $\Omega_{h,t} \subset \widehat{\Omega}_{h,t}$ be the final Top-k pair set under $\sigma_{h,t}$. We define $\alpha_{h,t} \in \Delta^{N-1}$ by applying a softmax over the selected pair scores, with $\alpha_{h,t,\pi(i,j)} = 0$ for all $(i,j) \notin \Omega_{h,t}$, where $\pi(i,j) = (i-1)n + j$.

This factorization provides each head with N addressable keys without a full N-way lookup. Compared with independent linear key banks, the total key parameters decrease from $\mathcal{O}(Nd)$ to $\mathcal{O}(\sqrt{N}d)$, and score computation decreases from $\mathcal{O}(TNd)$ to $\mathcal{O}(T\sqrt{N}d)$.

Head-wise Value Realization. Distinct key spaces allow attention heads to retrieve independently, but maintaining a separate value bank for every head would substantially increase memory parameters. MIDUS instead shares the dominant value storage across heads and uses head-specific projections to realize the retrieved content separately in each head space. The retrieved representations remain in the head dimension $d_h = d/H$ and are concatenated to recover the model dimension.

We use a shared value bank $\bar{V} \in \mathbb{R}^{N\times d_h}$ and head-specific projections $W_h \in \mathbb{R}^{d_h\times d_h}$. For each head h and token t, the retrieved representation is

$$\bar{m}_{h,t} = \sum_{(i,j)\in\Omega_{h,t}} \alpha_{h,t,\pi(i,j)} \bar{V}_{\pi(i,j)}, \qquad m_{h,t} = \bar{m}_{h,t} W_h \in \mathbb{R}^{d_h}. \tag{8}$$

Stacking over tokens gives $m_h \in \mathbb{R}^{T\times d_h}$, and concatenating the head outputs yields

$$m(x) = \left[m_1 \mid \cdots \mid m_H\right] \in \mathbb{R}^{T\times d}. \tag{9}$$

This parameterization separates shared storage from head-specific utilization. The shared value bank provides a common content basis, while W_h realizes the retrieved content in the representation space of each head.

A linear memory with model-dimensional values requires Nd value parameters, whereas MIDUS uses $Nd_h + Hd_h^2 = (Nd + d^2)/H$. When $N \gg d$, the Nd_h term dominates, reducing the value parameter cost by approximately a factor of H. The parameter advantage therefore becomes more pronounced as the memory size increases.

Memory-Infused Depth Up-Scaling. With the head-wise memory output $m(x)$ in Eq.(9), MIDUS constructs an inserted block as

$$M(x) = x + m(x). \tag{10}$$

Given anchor indices $0 \le i_0 < \cdots < i_{D-1} \le L-1$, MIDUS inserts D blocks $M_0,\dots,M_{D-1}$ before the corresponding pre-trained Transformer blocks,

$$f_{\mathrm{MIDUS}}(x) = B_{L-1} \circ \cdots \circ B_{i_{D-1}} \circ M_{D-1} \circ \cdots \circ B_{i_0} \circ M_0 \circ \cdots \circ B_0(x). \tag{11}$$

This ordering differs from Eq.(3), where each duplicated Transformer block is placed after its anchor. In MIDUS, M_j first forms head-wise contextual representations and retrieves memory before B_{i_j} processes the resulting memory-infused representation. We initialize the query, key, and value

projections of M_j from those of B_{i_j}, providing a pre-trained initialization for its attention component. The shared value bank $\bar{V}$ is initialized to zero, giving $m(x) = 0$ and $M_j(x) = x$ at initialization. The expanded model therefore initially preserves the function of the pre-trained model while providing a structured starting point for learning memory augmentation. Figure 2 summarizes the MIDUS process.

3.3 Structural Analysis of Head-wise Value Realization

We analyze the role of head-wise value realization under fixed retrieval. By holding the selected memory slots and retrieval weights fixed, we isolate the value parameterization from changes in retrieval. We compare a shared realized value bank with independent head-specific value banks and characterize when the parameterization used by MIDUS can recover the head-specific solution.

For each head h, let $A_h \in \mathbb{R}^{T\times N}$ denote the fixed retrieval-weight matrix whose t-th row is $\alpha_{h,t}^\top$, and let $Y_h \in \mathbb{R}^{T\times d_h}$ denote the target memory output. Under the same retrieval weights, a shared realized value bank $V_{\mathrm{sh}} \in \mathbb{R}^{N\times d_h}$ produces $A_h V_{\mathrm{sh}}$, while independent head-specific banks $V_h \in \mathbb{R}^{N\times d_h}$ produce $A_h V_h$. MIDUS instead uses a shared bank $\bar{V} \in \mathbb{R}^{N\times d_h}$ with head-specific projections $W_h \in \mathbb{R}^{d_h\times d_h}$, producing $A_h \bar{V} W_h$. We define the corresponding optimal losses as

$$\mathcal{L}^\star_{\mathrm{Share}} = \inf_{V_{\mathrm{sh}}} \frac{1}{2}\sum_{h=1}^{H} \|A_h V_{\mathrm{sh}} - Y_h\|_F^2\,, \qquad \mathcal{L}^\star_{\mathrm{Ind}} = \inf_{\{V_h\}_{h=1}^H} \frac{1}{2}\sum_{h=1}^{H} \|A_h V_h - Y_h\|_F^2\,, \tag{12}$$

$$\mathcal{L}^\star_{\mathrm{MIDUS}} = \inf_{\bar{V},\{W_h\}_{h=1}^H} \frac{1}{2}\sum_{h=1}^{H} \left\|A_h \bar{V} W_h - Y_h\right\|_F^2\,. \tag{13}$$

Theorem 3.1. *Suppose $A_h = A$ for all $h \in [H]$, and let Π_A denote the orthogonal projection onto the column space of A. Define $\bar{Y}_A = \frac{1}{H}\sum_{h=1}^{H} \Pi_A Y_h$. Then*

$$\mathcal{L}^\star_{\mathrm{Share}} - \mathcal{L}^\star_{\mathrm{Ind}} = \frac{1}{2}\sum_{h=1}^{H} \left\|\Pi_A Y_h - \bar{Y}_A\right\|_F^2\,. \tag{14}$$

Furthermore, suppose an optimal independent solution $\{V_h^\star\}_{h=1}^H$ admits a shared factorization $V_h^\star = \bar{V} W_h$, where $\bar{V} \in \mathbb{R}^{N\times d_h}$ and $W_h \in \mathbb{R}^{d_h\times d_h}$. Then $\mathcal{L}^\star_{\mathrm{MIDUS}} = \mathcal{L}^\star_{\mathrm{Ind}}$. Consequently, if $\Pi_A Y_h \neq \Pi_A Y_{h'}$ for some $h \neq h'$, then $\mathcal{L}^\star_{\mathrm{MIDUS}} < \mathcal{L}^\star_{\mathrm{Share}}$.

Proofs and additional results are provided in Appendix A.10. The assumption $A_h = A$ deliberately removes retrieval differences across heads, so the gap in Eq.(14) arises solely from value realization. The gap is determined by variation among the projected head targets. When these targets differ, a shared realized value bank must compromise across heads. In contrast, the head-specific projections in MIDUS allow different outputs to be realized from the same stored content. When the optimal head-specific banks admit the shared factorization above, MIDUS achieves the same optimum as fully independent value banks without storing a separate bank for every head.

This result also connects to FFN-based DUS through the memory view of Transformer FFNs (Geva et al., 2021). Under this interpretation, an FFN provides value capacity over the full hidden representation rather than explicitly preserving separate attention-head channels. This analysis captures a value distinction in MIDUS without modeling the behavior of FFN-based expansion. More specifically, it isolates a setting where shared block-level value realization becomes restrictive when different heads require different outputs within the same retrieval span. MIDUS addresses this restriction by sharing stored content while allowing each attention head to realize that content differently.

4 Results

Models and baselines. Following Cao et al. (2025), we use Llama-3.2-1B and Llama-3.1-8B (Grattafiori et al., 2024) as the main backbones and insert 8 and 16 additional modules, respectively. Each MIDUS Memory block uses $n = 64$ sub-keys per product-key codebook, giving $N = n^2 = 4096$ addressable slots per head with Top-k retrieval using $k = 4$. Both backbones have $H = 32$ attention heads, resulting in 1.05M and 2.10M head-specific addressable slots across

Table 1: Performance of DUS and MIDUS across Llama-3.2-1B and Llama-3.1-8B backbones under CPT and SFT settings. Llama-3.2-1B results report the mean over three runs, with standard deviations provided in Table 5. Llama-3.1-8B† baseline results are from Cao et al. (2025).

		Llama-3.2-1B									Llama-3.1-8B†								
		PPL ↓	Zero-shot Accuracy ↑								PPL ↓	Zero-shot Accuracy ↑							
	Method	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
CPT	Base	13.21	68.93	21.40	59.61	27.22	63.63	75.03	29.41	49.32	8.35	79.97	26.88	72.06	65.19	81.83	78.84	58.61	66.20
	SOLAR	13.39	**69.21**	**23.25**	60.19	25.28	61.09	75.08	29.70	49.12	9.90	79.88	26.88	71.59	57.41	80.70	78.56	54.37	64.20
	Llama Pro	12.08	67.34	21.25	60.54	38.69	64.85	74.81	32.44	51.42	7.81	81.61	**29.49**	73.72	70.93	81.65	79.98	62.56	68.56
	LESA	12.06	66.32	21.41	60.98	44.34	64.27	74.28	35.53	52.45	7.73	82.07	27.96	74.11	**72.40**	81.93	80.30	62.63	68.77
	OpT-DeUS	12.08	66.82	23.14	**61.01**	45.56	63.80	74.37	36.07	52.97	7.73	82.07	27.34	**74.74**	71.91	82.26	**80.79**	62.96	68.87
	Avg-DeUS	12.58	66.33	21.96	60.48	38.25	63.67	74.18	32.06	50.99	7.95	82.15	27.50	73.48	71.09	82.17	80.20	62.11	68.39
	MIDUS	**11.64**	65.91	22.84	60.85	**46.82**	**65.28**	**75.14**	**37.03**	**53.41**	7.35	**82.49**	28.42	**74.74**	71.91	**82.97**	80.30	**63.06**	**69.13**
SFT	Base	13.10	**69.61**	21.71	59.38	28.42	62.96	75.57	30.20	49.69	8.32	81.10	24.58	72.14	68.30	82.14	79.71	59.17	66.73
	SOLAR	13.20	69.57	23.50	59.38	26.89	60.17	75.77	30.48	49.39	9.68	80.68	25.19	71.19	61.18	81.19	79.16	55.03	64.80
	Llama Pro	12.19	68.07	21.61	**60.93**	43.52	64.32	75.53	33.97	52.57	7.81	83.33	27.19	74.11	72.07	82.26	80.79	62.32	68.87
	LESA	12.24	66.95	21.76	60.38	46.71	65.45	74.99	35.90	53.16	7.72	83.84	26.57	75.53	73.05	83.00	80.69	63.57	69.47
	OpT-DeUS	12.15	67.27	**23.55**	60.09	48.10	64.94	75.21	37.14	53.76	7.73	83.80	26.73	**76.09**	73.05	83.36	80.85	63.84	69.67
	Avg-DeUS	12.65	66.67	22.28	60.04	43.43	65.07	74.66	33.21	52.20	7.91	83.88	26.42	75.45	72.89	83.18	80.47	63.10	69.34
	MIDUS	**11.75**	66.36	21.71	60.40	**49.74**	**65.81**	**75.79**	**37.80**	**53.94**	**7.45**	**84.43**	**28.11**	74.82	**73.22**	**83.88**	**81.34**	**64.25**	**70.01**

Table 2: Retrieval-heavy and long-context evaluation of Llama-3.2-1B CPT models.

	Retrieval-heavy				Long-context (LongBench)						
Method	TriviaQA	NQ-Open	RACE	Average	Single-Doc QA	Multi-Doc QA	Summarization	Few-shot	Synthetic	Code	Average
Base	30.38	9.72	**38.28**	26.13	10.45	7.40	19.10	55.68	**4.15**	27.47	21.32
Llama Pro	34.33	10.28	37.99	27.53	10.52	7.32	21.09	58.08	3.35	35.70	23.07
OpT-DeUS	39.02	12.11	38.18	29.77	**10.92**	7.97	22.94	60.60	3.79	47.57	25.63
Avg-DeUS	38.25	11.75	37.42	29.14	10.66	7.74	21.04	60.38	3.54	46.96	25.03
MIDUS	**39.75**	**12.24**	37.51	**29.83**	10.60	**8.48**	**23.23**	**63.52**	2.62	**54.64**	**27.18**

all inserted blocks for Llama-3.2-1B and Llama-3.1-8B, respectively. We compare MIDUS with SOLAR (Kim et al., 2023), Llama Pro (Wu et al., 2024), LESA (Yang et al., 2025), and OpT-DeUS and Avg-DeUS (Cao et al., 2025), with the pre-trained backbone included as *Base*. For memory comparisons, we additionally consider Linear memory, PKM (Lample et al., 2019), and UltraMem (Huang et al., 2025). Further details on these methods are provided in Section 5.

Training. We perform CPT on `FineWeb-Edu` for general-purpose adaptation and `MathPile` for math-domain adaptation (Penedo et al., 2024; Wang et al., 2024). Following prior DUS studies (Wu et al., 2024; Yang et al., 2025; Cao et al., 2025), we train the inserted modules while freezing the pre-existing backbone blocks, except for Base and SOLAR where all parameters are updated. We apply SFT to the `FineWeb-Edu` CPT models using `Alpaca-GPT4` (Peng et al., 2023). Full optimization and dataset details are provided in Appendix A.2.

DUS placement policy. MIDUS uses the *Distributed* placement by default, following Eq.(11), while OpT-DeUS uses the more efficiency-oriented *Top-heavy* policy that concentrates inserted modules toward the upper layers. We additionally evaluate MIDUS with the *Top-heavy* and *Bottom-heavy* policies in Table 14. Exact placement definitions are provided in Appendix A.4.

Evaluation. We use `lm-evaluation-harness` (Gao et al., 2021). General-purpose evaluation includes ARC-Easy, LogiQA, Winogrande, CSQA, BoolQ, PIQA, MMLU, and WikiText perplexity (Clark et al., 2018; Liu et al., 2020; Sakaguchi et al., 2021; Talmor et al., 2018; Clark et al., 2019a; Bisk et al., 2020; Hendrycks et al., 2020; Merity et al., 2016). We additionally evaluate math-domain performance on GSM8K, GSM8K–CoT, MATH, and MathQA (Cobbe et al., 2021; Hendrycks et al., 2021; Amini et al., 2019), retrieval-heavy performance on TriviaQA, NQ-Open, and RACE (Joshi et al., 2017; Lee et al., 2019; Lai et al., 2017), and long-context performance on LongBench (Bai et al., 2024). General-purpose benchmarks use zero-shot evaluation, while math-domain and retrieval-heavy benchmarks use 5-shot evaluation.

4.1 Performance across Tasks and Adaptation Settings

General-purpose CPT and SFT. Table 1 reports performance after CPT on `FineWeb-Edu` and SFT on `Alpaca-GPT4`. For Llama-3.2-1B, we report the mean over three runs, with the corresponding standard deviations provided in Table 5. Note that ablation studies use single-seed runs to match the experimental protocol of the corresponding baselines. After CPT, MIDUS achieves

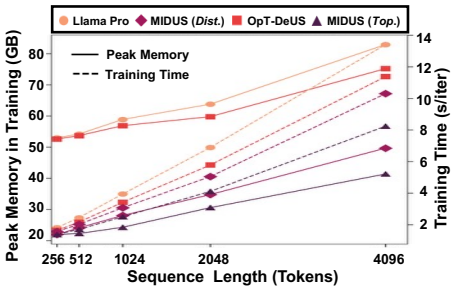

Figure 3: Efficiency of DUS baselines and MIDUS on Llama-3.1-8B across sequence length. *Dist.* denotes the *Distributed* and *Top.* the *Top-heavy* DUS policy.

Table 3: Efficiency for Llama-3.1-8B with the number of parameters (trainable/total), GPU memory (train/inference), training time, and throughput. Linear, PKM, and UltraMem use the same number of memory heads as MIDUS

Method	B ↓ Params	GB ↓ GPU Memory	s/iter ↓ Train	Tokens/s ↑ Throughput
Base	8.03 / 8.03	81.6 / 19.2	6.12	61.0
Llama Pro	3.49 / 11.52	63.8 / 27.0	6.89	41.3
OpT-DeUS	3.49 / 11.52	59.8 / 27.0	5.78	41.3
Linear	1.48 / 9.51	51.3 / 23.1	11.05	42.4
PKM	1.21 / 9.24	48.3 / 22.5	9.88	47.3
UltraMem	1.29 / 9.32	56.9 / 23.6	8.56	36.9
MIDUS	**0.42 / 8.45**	**34.8 / 20.1**	**5.05**	**50.5**

the lowest WikiText PPL and the highest average zero-shot accuracy on both backbones. After SFT, MIDUS remains competitive in individual benchmarks and achieves the highest average accuracy on both backbones. These results show that the efficiency of MIDUS does not come at the expense of overall downstream performance. The consistent pattern across the 1B and 8B backbones suggests that this performance–efficiency trade-off is not specific to a single model scale. We further evaluate Qwen-2.5-3B without model-specific hyperparameter tuning in Appendix A.6, where MIDUS shows a similar performance and efficiency pattern relative to OpT-DeUS.

Retrieval-heavy and long-context evaluation. Table 2 evaluates the same CPT-trained Llama-3.2-1B models on retrieval-heavy and long-context tasks. MIDUS achieves the highest average score across TriviaQA, NQ-Open, and RACE, while remaining close to the strongest baseline on the individual retrieval-heavy benchmarks. On LongBench, MIDUS obtains the highest overall score, with improvements on multi-document QA, summarization, few-shot learning, and code. Together with the head-importance analysis in Section 4.4, these results are consistent with our motivation for organizing memory capacity around heterogeneous attention heads.

Additional adaptation results. We further evaluate MIDUS under math-domain CPT and an additional SFT dataset in Appendix A.7. On `MathPile`, MIDUS achieves the highest average 5-shot accuracy on both Llama-3.2-1B and Llama-3.1-8B while remaining competitive across individual math benchmarks. After SFT on `Databricks-Dolly-15k`, MIDUS also achieves the lowest WikiText PPL and the highest average accuracy among the compared methods. These results show a similar performance pattern across different adaptation domains and instruction-tuning data. Together with the Qwen-2.5-3B results, these experiments provide additional evidence that the observed behavior extends across model families and adaptation settings.

4.2 Efficiency of MIDUS

Efficiency against DUS baselines. Table 3 compares the parameter and runtime costs of MIDUS with DUS baselines on Llama-3.1-8B. Relative to OpT-DeUS, MIDUS reduces trainable parameters from 3.49B to 0.42B while also reducing total parameters and peak GPU memory in both training and inference. MIDUS further achieves lower training time and higher generation throughput. These gains arise from complementary design choices. Product-key factorization and shared value storage limit the parameter growth of the inserted memory, while sparse retrieval replaces the dense per-token FFN computation introduced by conventional DUS. Together, these properties substantially reduce the cost of depth expansion while maintaining competitive downstream performance. The simultaneous reductions in parameter, memory, and runtime costs suggest that the efficiency gain is structural rather than a trade-off along a single resource dimension.

Figure 3 shows that the training-memory and training-time advantages persist as sequence length increases. Although MIDUS uses the *Distributed* placement by default while OpT-DeUS uses the more efficiency-oriented *Top-heavy* placement, MIDUS remains more efficient across the evaluated sequence lengths. Using the same *Top-heavy* placement further improves MIDUS efficiency. This consistency across sequence lengths and placement settings indicates that the observed advantage is not specific to a particular evaluation configuration. Additional results across placement policies,

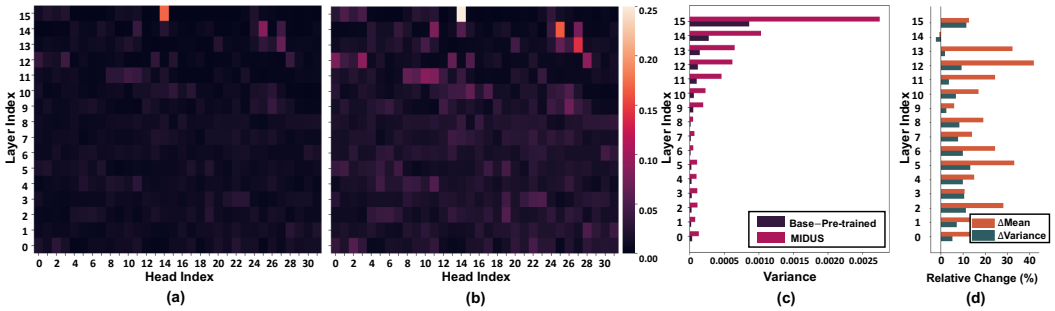

Figure 4: Head-importance analysis on PIQA for Llama-3.2-1B. (a) Head importance across the backbone layers of the pre-trained model. (b) Head importance after CPT with MIDUS. (c) Within-layer variance of head importance for the pre-trained model and MIDUS. (d) Relative changes in the mean and within-layer variance of head importance from OpT-DeUS to MIDUS under identical CPT.

batch sizes, and backbone scales are provided in Appendices A.4 and A.9. Implementation details are provided in Appendix A.3.

4.3 Memory Parameterization

Performance across memory parameterizations. Table 4 compares MIDUS with Linear memory, PKM (Lample et al., 2019), and UltraMem (Huang et al., 2025) under the same head-wise interface. All variants use attention-head outputs as queries and maintain distinct key spaces across heads. MIDUS achieves the highest average accuracy while matching the lowest WikiText PPL, indicating that its performance is not explained by sparse memory alone.

Table 4: CPT results on `FineWeb-Edu` with Llama-3.2-1B for MIDUS variants with different memory layers. Linear, PKM, and UltraMem use the same number of memory heads as MIDUS.

	PPL ↓	Zero-shot Accuracy ↑							
Mem	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
Linear	11.82	65.61	22.73	**62.04**	42.42	64.71	75.08	35.89	52.64
PKM	11.97	65.70	21.35	61.40	42.18	**65.47**	74.65	33.93	52.10
UltraMem	**11.64**	**66.46**	21.81	61.09	44.31	64.83	**75.63**	36.49	52.94
MIDUS	**11.64**	66.16	**23.20**	61.56	**46.27**	65.29	75.08	**36.91**	**53.50**

Controlled comparisons in Appendix A.8.1 further isolate the effects of head-wise retrieval, distinct key spaces, and head-specific value realization. Removing head-specific value realization reduces the average accuracy from 53.50 to 52.52, while sharing the key map gives 52.94 and a conventional block-level PKM gives 52.36. These results support preserving attention-head identity in retrieval and value realization while sharing the dominant value storage across heads.

Efficiency across memory parameterizations. Table 3 shows that MIDUS uses the fewest trainable and total parameters and the lowest training and inference memory among memory variants. It also achieves the fastest training and highest throughput. These results show that head-specific memory access does not require independent storage for every head, allowing MIDUS to retain the computational benefits of sparse retrieval while reducing memory parameter and storage costs.

4.4 Head-wise Analysis

To examine how the head-wise memory organization of MIDUS affects the original attention heads, we use the gradient-based head importance score IS_h proposed by Bansal et al. (2023). For examples $(x, y) \sim \mathcal{D}$, let $\ell(x, y)$ denote the autoregressive negative log-likelihood and let $a_{h,t}([x \mid y])$ denote the output of head h at position t. We define

$$IS_h(\mathcal{D}) := \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\left|\sum_t a_{h,t}([x \mid y])^\top \frac{\partial \ell(x, y)}{\partial a_{h,t}([x \mid y])}\right|\right].$$

Figures 4(a)–(c) visualize how the head-importance profile changes from the pre-trained backbone to MIDUS after CPT on `FineWeb-Edu`. MIDUS exhibits a more concentrated distribution of head importance, with larger within-layer variation across attention heads.

To separate this effect from the influence of CPT itself, Figure 4(d) directly compares MIDUS with OpT-DeUS after the same CPT procedure. We evaluate the 16 original backbone layers, whose parameters remain frozen in both models. Relative to OpT-DeUS, MIDUS increases both the mean head importance and its within-layer variance in 15 of 16 layers, with maximum increases of 13.37% and 41.94%, respectively. This controlled comparison shows a more differentiated head-importance profile under MIDUS than OpT-DeUS, consistent with organizing the added memory capacity around individual attention heads.

5 Related Work

5.1 Depth Up-Scaling

Model up-scaling expands capacity along different axes, such as the width-wise route taken by MoE that selectively activates expert FFNs (Shazeer et al., 2017; Jiang et al., 2024). DUS instead expands models depth-wise by duplicating and reorganizing Transformer blocks. Early work studies block selection and growth schedules through stacking and progressive training (Gong et al., 2019; Yang et al., 2020; Du et al., 2024; Saunshi et al., 2024; Pan et al., 2024; Yano et al., 2025; Zhang et al., 2026). Llama Pro selectively copies blocks under a stabilizing recipe (Wu et al., 2024), while SOLAR stacks contiguous block groups (Kim et al., 2023). Recent methods focus on initialization. LESA learns per-layer initializers (Yang et al., 2025), while OpT-DeUS uses transport-guided initialization and Avg-DeUS initializes new layers by averaging adjacent blocks (Cao et al., 2025). These methods improve how additional Transformer blocks are selected, placed, or initialized, while the form of the added capacity remains largely unchanged. Increasing depth therefore still adds FFN parameters and dense per-token computation. Token-routing methods (Raposo et al., 2024; Tan et al., 2024) reduce executed computation but retain duplicated FFN parameters at inference. MIDUS instead changes the inserted capacity by replacing duplicated FFN branches with retrieval-based memory.

5.2 Memory Layer

A memory layer stores key–value slots and retrieves values through dot-product lookup. Linear memories scale poorly because lookup cost and key parameters grow linearly with memory size. PKM (Lample et al., 2019) factorizes the key space into two codebooks, enabling lookup over many composite keys with fewer dot products and lower key-side cost. Lample et al. (2019) replace selected FFNs with PKM layers for language modeling, while Kim & Jung (2020) extend PKM to pre-trained language models through residual memory augmentation.

MIDUS builds on these memory scaling techniques but addresses a different question of how retrieval-based capacity should be organized for depth up-scaling. PKM reduces key-side lookup and parameter costs but does not impose attention-head-specific memory organization. MIDUS instead ties distinct retrieval spaces directly to attention-head representations and shares value storage across heads, using head-specific projections to realize the retrieved content in each head space. This also differs from the implicit value expansion of UltraMem. MIDUS does not use its value projections to create additional virtual memory addresses. Instead, the projections are tied to attention-head identity and determine how the same shared stored content is realized across heads.

6 Conclusion

We introduced MIDUS, a depth up-scaling approach that replaces the FFN-heavy capacity added by conventional DUS with sparse memory organized around individual attention heads. MIDUS directly uses attention-head outputs for retrieval and combines shared value storage with head-specific projections, providing an efficient way to add capacity without duplicating dense FFN structures. Our fixed-retrieval analysis isolates the role of head-specific value realization, while experiments across CPT, SFT, retrieval-heavy, long-context, and math-domain settings show that MIDUS remains competitive with strong DUS baselines while substantially reducing parameter, memory, and computational costs. Head-importance analysis further shows a more concentrated distribution across attention heads after CPT, consistent with the head-wise organization of the added memory capacity. Overall, MIDUS provides a practical memory-based alternative for efficient DUS.

AI USE STATEMENT

In this work, we used generative AI tools to assist with feedback on experimental design, interpretation of experimental results, and refinement of the manuscript and mathematical argument. We also used generative AI tools for language editing, code assistance, and LaTeX formatting. All AI-assisted content, analyses, and code were reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

ETHICS STATEMENT

This work does not involve human subjects or private user data. MIDUS inherits the broader risks of the underlying language models and should be deployed with appropriate evaluation and safeguards.

REPRODUCIBILITY STATEMENT

Training hyperparameters, dataset construction, implementation details, placement policies, additional experiments, ablation studies, efficiency analyses, and complete theoretical proofs are provided in the Appendix. Anonymous source code is provided with the submission.

REFERENCES

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pp. 2357–2367, 2019.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Hritik Bansal, Karthik Gopalakrishnan, Saket Dingliwal, Sravan Bodapati, Katrin Kirchhoff, and Dan Roth. Rethinking the role of scale for in-context learning: An interpretability-based case study at 66 billion scale. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11833–11856, 2023.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Mingzi Cao, Xi Wang, and Nikolaos Aletras. Progressive depth up-scaling via optimal transport. *arXiv preprint arXiv:2508.08011*, 2025.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019a.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. In *Proceedings of the 2019 ACL workshop BlackboxNLP: analyzing and interpreting neural networks for NLP*, pp. 276–286, 2019b.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instructiontuned llm. 2023.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Wenyu Du, Tongxu Luo, Zihan Qiu, Zeyu Huang, Yikang Shen, Reynold Cheng, Yike Guo, and Jie Fu. Stacking your transformers: A closer look at model growth for efficient llm pre-training. *Advances in Neural Information Processing Systems*, 37:10491–10540, 2024.

Jared Fernandez, Yonatan Bisk, and Emma Strubell. Gradient localization improves lifelong pretraining of language models. *arXiv preprint arXiv:2411.04448*, 2024.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, 2021.

Linyuan Gong, Di He, Zhuohan Li, Tao Qin, Liwei Wang, and Tieyan Liu. Efficient training of bert by progressively stacking. In *International conference on machine learning*, pp. 2337–2346. PMLR, 2019.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Zihao Huang, Qiyang Min, Hongzhi Huang, Yutao Zeng, Defa Zhu, Ran Guo, and zhou Xun. Ultra-sparse memory network. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=zjeHLSiNv1.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, et al. Solar 10.7 b: Scaling large language models with simple yet effective depth up-scaling. *arXiv preprint arXiv:2312.15166*, 2023.

Gyuwan Kim and Tae-Hwan Jung. Large product key memory for pretrained language models. *arXiv preprint arXiv:2010.03881*, 2020.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pp. 785–794, 2017.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32, 2019.

Kenton Lee, Ming-Wei Chang, and Kristina Toutanova. Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 6086–6096, 2019.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*, 2020.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

Yu Pan, Ye Yuan, Yichun Yin, Jiaxin Shi, Zenglin Xu, Ming Zhang, Lifeng Shang, Xin Jiang, and Qun Liu. Preparing lessons for progressive training on language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 18860–18868, 2024.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 30811–30849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0970. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Nikunj Saunshi, Stefani Karp, Shankar Krishnan, Sobhan Miryoosefi, Sashank Jakkam Reddi, and Sanjiv Kumar. On the inductive bias of stacking towards improving reasoning. *Advances in Neural Information Processing Systems*, 37:71437–71464, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

Zhen Tan, Daize Dong, Xinyu Zhao, Jie Peng, Yu Cheng, and Tianlong Chen. Dlo: Dynamic layer operation for efficient vertical scaling of llms. *arXiv preprint arXiv:2407.11030*, 2024.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 5797–5808, 2019.

Zengzhi Wang, Xuefeng Li, Rui Xia, and Pengfei Liu. Mathpile: A billion-token-scale pretraining corpus for math. *Advances in Neural Information Processing Systems*, 37:25426–25468, 2024.

Chengyue Wu, Yukang Gan, Yixiao Ge, Zeyu Lu, Jiahao Wang, Ye Feng, Ying Shan, and Ping Luo. Llama pro: Progressive llama with block expansion. *arXiv preprint arXiv:2401.02415*, 2024.

Cheng Yang, Shengnan Wang, Chao Yang, Yuechuan Li, Ru He, and Jingqiao Zhang. Progressively stacking 2.0: A multi-stage layerwise training method for bert training speedup. *arXiv preprint arXiv:2011.13635*, 2020.

Yifei Yang, Zouying Cao, Xinbei Ma, Yao Yao, Zhi Chen, Libo Qin, and Hai Zhao. Lesa: Learnable llm layer scaling-up. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22463–22476, 2025.

Kazuki Yano, Sho Takase, Sosuke Kobayashi, Shun Kiyono, and Jun Suzuki. Efficient construction of model family through progressive training using model expansion. *arXiv preprint arXiv:2504.00623*, 2025.

Kayo Yin and Jacob Steinhardt. Which attention heads matter for in-context learning? *arXiv preprint arXiv:2502.14010*, 2025.

Jinyang Zhang, Yue Fang, Hongxin Ding, Weibin Liao, Muyang Ye, Junfeng Zhao, Yasha Wang, and Xu Chu. ADEPT: Continual pretraining via adaptive expansion and dynamic decoupled tuning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=vcWDDfA4Ev.

Youxiang Zhu, Ruochen Li, Danqing Wang, Daniel Haehn, and Xiaohui Liang. Focus directions make your language models pay more attention to relevant contexts. *arXiv preprint arXiv:2503.23306*, 2025.

A Appendix

A.1 Limitations and Broader Impact

While we evaluate MIDUS across multiple tasks and model families, further validation at larger scales and under different training regimes remains future work. Our Llama-3.2-1B results are averaged over three runs, whereas the Llama-3.1-8B and Qwen-2.5-3B experiments are based on single runs due to their higher computational cost. Our structural analysis isolates head-specific value realization under fixed retrieval and does not characterize the full optimization dynamics of MIDUS. MIDUS reduces the computational cost of depth up-scaling, while inheriting the broader risks associated with the underlying language models.

A.2 Experimental Details

A.2.1 Hyperparameter Settings

We conduct `FineWeb-Edu` experiments on NVIDIA RTX 6000 Blackwell GPUs and `MathPile` experiments on NVIDIA H200 GPUs. All models are trained in `bfloat16` using AdamW (Loshchilov & Hutter, 2017), a cosine learning rate schedule with a warmup ratio of 0.1, and `FlashAttention-2` (Dao, 2023). We sweep weight decay over $\{0, 10^{-6}, 10^{-2}\}$. For MIDUS, memory key and value parameters use the maximum learning rate without scheduling and zero weight decay. For Llama-3.2-1B on `FineWeb-Edu`, we follow the search protocol of prior work for all methods, using a fixed learning rate of 10^{-4} for CPT and 10^{-5} for SFT. For Llama-3.1-8B, baseline results are taken from Cao et al. (2025); MIDUS uses 5×10^{-5} for CPT and searches SFT learning rates over $\{10^{-6}, 5 \times 10^{-6}\}$. Unless otherwise stated, the global batch size is 64 and the sequence length is 2,048. During SFT, all model parameters are updated following prior work (Yang et al., 2025; Cao et al., 2025).

A.2.2 Dataset Details

Following Cao et al. (2025), we construct a 1.5B-token `FineWeb-Edu` subset from the CC-MAIN-2024-51 slice (Penedo et al., 2024) after the base model's pre-training cutoff. For SFT, we use `Alpaca-GPT4` with 52k examples (Peng et al., 2023) and `Databricks-Dolly-15k` with 15k examples (Conover et al., 2023). For `MathPile` (Wang et al., 2024), we use approximately 1.1B tokens from Textbooks, Wikipedia, ProofWiki, CommonCrawl, and StackExchange, excluding arXiv.

A.3 Implementation Details

MIDUS replaces dense FFN computation in the inserted blocks with sparse retrieval and weighted value aggregation. To improve practical efficiency, we use optimized product-key selection, inference-time value caching, and deduplicated gradient accumulation.

Fused Cartesian Top-k. The product-key retrieval in MIDUS first selects Top-k row and column sub-keys and then chooses the final Top-k pairs from their Cartesian product, as described in Section 3.2. For short-query inference, this hierarchical procedure can introduce non-trivial kernel-launch overhead. We therefore construct the Cartesian score grid

$$(\mathcal{S}_{h,t})_{ij} = S^{\mathrm{row}}_{h,t}(i) + S^{\mathrm{col}}_{h,t}(j) \tag{15}$$

and directly select the Top-k pairs from the flattened grid. Although this uses $\mathcal{O}(N)$ rather than $\mathcal{O}(\sqrt{N})$ pair-selection work, it improves GPU utilization for short queries and small batches. For longer sequences and larger batches, we use the standard two-stage product-key selection.

Head-wise value caching. During inference, the parameters are fixed, so repeatedly applying the head-specific projection W_h after value aggregation is unnecessary. We therefore precompute

$$V^{\mathrm{cache}}_h = \bar{V} W_h \in \mathbb{R}^{N \times d_h}. \tag{16}$$

At runtime, MIDUS retrieves directly from V^{cache}_h using the selected indices and retrieval weights. By linearity, this produces the same output as retrieving from $\bar{V}$ and subsequently applying W_h. The cached tables are constructed only for inference and do not change the learned parameterization. All reported inference memory, generation throughput, and prefill measurements use this value cache.

A.3.1 Deduplicated Backward Aggregation

A practical bottleneck in training sparse memory layers arises during the backward pass of value aggregation. When multiple retrieval queries access the same value-table row, standard implementations may issue concurrent updates to the same memory address, leading to atomic contention on GPUs.

We therefore use a custom backward implementation based on deduplication and pre-aggregation. For value aggregation

$$y_p = \sum_{u=1}^{k} \lambda_{p,u} V_{\mathcal{I}_{p,u}}, \qquad p = 1, \ldots, P, \tag{17}$$

where $\mathcal{I}_{p,u} \in \{1, \ldots, N\}$ denotes the retrieved value index and $\lambda_{p,u}$ its corresponding retrieval weight, the gradient for value row j is

$$G_V[j] = \sum_{(p,u):\, \mathcal{I}_{p,u}=j} \lambda_{p,u} G_Y[p]. \tag{18}$$

Rather than issuing one global update for every retrieved entry, Algorithm 1 first forms the individual contributions, deduplicates the retrieved indices, and aggregates all contributions associated with the same value row before writing to the global gradient table.

Algorithm 1 Deduplicated Backward Pass for Value Aggregation

Require: Output gradient $G_Y \in \mathbb{R}^{P \times d_v}$, value indices $\mathcal{I} \in \{1, \ldots, N\}^{P \times k}$, retrieval weights $\Lambda \in \mathbb{R}^{P \times k}$, value-table size N

Step 1: Form per-retrieval gradient contributions
$G_{\text{exp}} \leftarrow G_Y.\text{unsqueeze}(1) \odot \Lambda.\text{unsqueeze}(-1)$
$G_{\text{flat}} \leftarrow G_{\text{exp}}.\text{reshape}(Pk, d_v)$
$\mathcal{I}_{\text{flat}} \leftarrow \mathcal{I}.\text{reshape}(Pk)$
Step 2: Deduplicate retrieved value indices
$\mathcal{I}_{\text{uniq}}, \eta \leftarrow \text{unique}(\mathcal{I}_{\text{flat}}, \text{return_inverse} = \text{True})$
Step 3: Pre-aggregate contributions
$U \leftarrow |\mathcal{I}_{\text{uniq}}|$
$G_{\text{uniq}} \leftarrow \text{zeros}(U, d_v)$
$G_{\text{uniq}}.\text{index_add_}(0, \eta, G_{\text{flat}})$
Step 4: Write one gradient per touched value row
$G_V \leftarrow \text{zeros}(N, d_v)$
$G_V[\mathcal{I}_{\text{uniq}}] \leftarrow G_{\text{uniq}}$

Ensure: G_V

This procedure computes the same value-table gradient as Eq.(18), up to floating-point accumulation order, while reducing repeated global atomic updates to the same value row. Gradients with respect to the retrieval weights follow the standard weighted-embedding rule,

$$\frac{\partial \mathcal{L}}{\partial \lambda_{p,u}} = \left\langle G_Y[p], V_{\mathcal{I}_{p,u}} \right\rangle. \tag{19}$$

A.4 DUS Placement Policy

The Llama-3.2-1B and Llama-3.1-8B backbones contain 16 and 32 Transformer blocks, respectively. We insert 8 and 16 additional modules, resulting in final depths of 24 and 48 blocks.

DUS and MIDUS may use the same anchor schedule while producing different final-stack positions because they follow different insertion rules. In conventional DUS, an added Transformer block is inserted after its anchor as in Eq.(3), whereas MIDUS inserts a Memory block before the corresponding pre-trained block as in Eq.(11). Accordingly, the *Llama Pro* and *Distributed* policies below use the same underlying anchor schedule but differ by one position in the expanded stack.

We consider four placement policies.

- *Top-heavy*: added modules are concentrated in the upper half of the stack, following Cao et al. (2025).

- *Llama Pro*: added Transformer blocks follow the distributed placement of Wu et al. (2024) under the DUS insertion rule.
- *Distributed*: MIDUS uses the same anchor schedule as Llama Pro under the before-anchor insertion rule in Eq.(11).
- *Bottom-heavy*: added modules are concentrated in the lower half of the stack.

Llama-3.2-1B with 24 blocks.

- *Top-heavy*: $\{8, 10, 12, 14, 16, 18, 20, 22\}$
- *Llama Pro*: $\{2, 5, 8, 11, 14, 17, 20, 23\}$
- *Distributed*: $\{1, 4, 7, 10, 13, 16, 19, 22\}$
- *Bottom-heavy*: $\{0, 2, 4, 6, 8, 10, 12, 14\}$

Llama-3.1-8B with 48 blocks.

- *Top-heavy*: $\{16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46\}$
- *Llama Pro*: $\{2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47\}$
- *Distributed*: $\{1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46\}$
- *Bottom-heavy*: $\{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30\}$

Among the MIDUS placements, *Top-heavy* reduces training cost, whereas *Distributed* gives the highest average accuracy in our ablation. The corresponding means are reported in Table 5.

A.5 Multi-Seed Results on Llama-3.2-1B

Table 5: Three-run results on Llama-3.2-1B under CPT and SFT. We report mean ± standard deviation across three runs. The corresponding means without standard deviations are reported in Table 1 of the main paper.

	Method	PPL ↓	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average ↑
CPT	Base	13.21±0.01	68.93±0.23	21.40±0.85	59.61±0.54	27.22±1.72	63.63±0.43	75.03±0.20	29.41±0.38	49.32±0.13
	SOLAR	13.39±0.02	**69.21±0.37**	**23.25±0.69**	60.19±0.12	25.28±0.92	61.09±0.48	75.08±0.14	29.70±0.47	49.12±0.23
	Llama Pro	12.08±0.22	67.34±0.53	21.25±1.05	60.54±0.93	38.69±3.11	64.85±0.42	74.81±0.24	32.44±1.06	51.42±0.67
	LESA	12.06±0.01	66.32±0.06	21.41±0.62	60.98±0.69	44.34±1.38	64.27±0.56	74.28±0.36	35.53±1.31	52.45±0.32
	OpT-DeUS	12.08±0.31	66.82±0.32	23.14±0.18	**61.01±0.59**	45.56±0.80	63.80±1.20	74.37±0.24	36.07±0.45	52.97±0.08
	Avg-DeUS	12.58±0.47	66.33±0.51	21.96±0.70	60.48±0.18	38.25±3.28	63.67±0.30	74.18±0.09	32.06±2.02	50.99±0.94
	MIDUS	**11.64±0.00**	65.91±0.30	22.84±0.26	60.85±0.51	**46.82±0.44**	**65.28±0.24**	**75.14±0.12**	**37.03±0.12**	**53.41±0.07**
SFT	Base	13.10±0.01	**69.61±0.15**	21.71±0.64	59.38±0.37	28.42±2.18	62.96±0.65	75.57±0.19	30.20±0.35	49.69±0.40
	SOLAR	13.20±0.02	69.57±0.34	23.50±0.53	59.38±0.96	26.89±0.47	60.17±0.41	75.77±0.14	30.48±0.72	49.39±0.12
	Llama Pro	12.19±0.31	68.07±0.34	21.61±1.11	**60.93±0.83**	43.52±0.82	64.32±1.37	75.53±0.13	33.97±0.31	52.57±0.36
	LESA	12.24±0.12	66.95±0.89	21.76±1.38	60.38±0.08	46.71±1.28	65.45±0.80	74.99±0.22	35.90±1.22	53.16±0.52
	OpT-DeUS	12.15±0.29	67.27±0.57	**23.55±0.23**	60.09±0.40	48.10±1.26	64.94±1.31	75.21±0.27	37.14±0.46	53.76±0.06
	Avg-DeUS	12.65±0.40	66.67±0.37	22.28±1.20	60.04±0.28	43.43±3.12	65.07±0.49	74.66±0.27	33.21±2.09	52.20±1.10
	MIDUS	**11.75±0.01**	66.36±0.05	21.71±0.23	60.40±0.78	**49.74±0.26**	**65.81±0.39**	**75.79±0.19**	**37.80±0.05**	**53.94±0.17**

A.6 Evaluation on Qwen-2.5-3B

Table 6: Evaluation on Qwen-2.5-3B using the hyperparameter configuration selected on Llama-3.2-1B without additional tuning. Results are from a single run. The Base model, shown in gray, updates all pre-trained parameters and is included as a reference rather than a directly matched comparison.

	Performance									Efficiency					
Method	PPL	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Avg.	Train Params (B)	Total Params (B)	Train Mem. (GB)	Infer. Mem. (GB)	Train Time (s/iter)	Throughput (tokens/s)
Base	10.88	80.18	31.03	70.24	70.35	75.23	77.91	62.75	66.81	3.18	3.18	83.31	10.14	11.60	29.9
OpT-DeUS	12.24	78.24	30.57	71.27	75.10	**78.93**	**77.86**	64.23	68.03	1.39	4.47	69.75	12.99	11.09	20.2
MIDUS	**10.72**	**79.08**	**33.49**	**71.35**	**76.17**	78.38	77.80	**64.73**	**68.71**	**0.11**	**3.20**	**56.33**	**11.10**	**10.33**	**22.5**

To evaluate MIDUS beyond the Llama model family, we conduct CPT on Qwen-2.5-3B using the hyperparameter configuration selected on Llama-3.2-1B without additional tuning. Qwen-2.5-3B contains 36 Transformer blocks with 16 attention heads, and we insert 18 additional modules for both OpT-DeUS and MIDUS.

MIDUS achieves a higher average accuracy and lower PPL than OpT-DeUS while using substantially fewer trainable and total parameters. It also reduces training and inference memory, shortens training time, and improves generation throughput. Although this experiment is based on a single run without Qwen-specific hyperparameter tuning, it provides additional evidence that the performance and efficiency characteristics of MIDUS extend beyond the Llama model family.

A.7 Additional Adaptation Results

A.7.1 CPT on MathPile

Table 7: Performance of CPT on MathPile for Llama-3.2-1B and Llama-3.1-8B models.

	Llama-3.2-1B (5-shot Accuracy ↑)					Llama-3.1-8B (5-shot Accuracy ↑)				
Method	GSM8K	GSM8K–CoT	MATH	MathQA	Average	GSM8K	GSM8K–CoT	MATH	MathQA	Average
Base	4.32	4.93	5.66	29.88	11.20	35.33	33.36	11.42	37.99	29.53
Llama Pro	5.61	4.40	5.64	30.62	11.57	41.93	41.77	13.10	40.87	34.42
OpT-DeUS	6.07	6.22	**5.80**	30.95	12.26	**49.36**	50.04	13.84	**42.58**	38.96
MIDUS	**6.60**	**6.60**	5.62	**31.39**	**12.55**	48.90	**51.78**	**14.34**	**42.58**	**39.40**

MIDUS achieves the highest average 5-shot accuracy on both Llama-3.2-1B and Llama-3.1-8B, while remaining competitive across the individual math benchmarks.

A.7.2 SFT on Databricks-Dolly-15k

Table 8: SFT results on the Databricks-Dolly-15k (Conover et al., 2023) with Llama-3.2-1B.

		Llama-3.2-1B								
		PPL ↓	Zero-shot Accuracy ↑							
	Method	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
SFT	Base	13.07	68.64	21.35	60.30	27.52	63.27	**75.24**	30.49	49.55
SFT	SOLAR	13.19	**69.40**	23.04	59.04	27.52	60.03	75.14	30.77	49.28
SFT	Llama Pro	12.47	68.10	**23.81**	60.54	41.93	63.70	**75.24**	34.03	52.48
SFT	LESA	11.77	66.46	23.66	60.77	49.06	64.98	74.86	38.22	54.00
SFT	OpT-DeUS	11.78	67.21	**23.81**	61.48	**49.14**	63.76	75.19	37.83	54.06
SFT	Avg-DeUS	12.10	67.51	22.73	60.30	46.60	65.02	74.37	36.79	53.33
SFT	MIDUS	**11.72**	66.62	22.12	**61.72**	48.65	**66.51**	75.14	**38.59**	**54.19**

MIDUS achieves the lowest WikiText PPL and the highest average zero-shot accuracy among the compared methods, showing a similar pattern to the Alpaca-GPT4 SFT results in the main paper.

A.8 Ablation Studies

A.8.1 Component-wise Ablation of MIDUS

We isolate the main architectural choices of MIDUS by selectively modifying its query, retrieval, and value realization mechanisms while keeping the remaining configuration fixed.

Removing head-specific value realization reduces the average accuracy from 53.50 to 52.52, while sharing the key map across heads yields 52.94 and replacing head-wise retrieval with global retrieval yields 53.21. The block-level PKM baseline obtains 52.36, indicating that the result cannot be attributed to product-key retrieval alone. Overall, these controlled variants support preserving attention-head identity across both retrieval and value realization.

A.8.2 Ablation study on the number of memory slots

In Table 10, we vary the total number of product-key memories from tens of thousands to one million, i.e., n=16, 32, 64 in the PKM factorization. Average zero-shot accuracy rises steadily with capacity, while PPL stays essentially flat. We hypothesize that a larger number of memory slots mainly helps tasks that benefit from retrieving facts or patterns, without disrupting the core language-modeling behavior. These results indicate that memory size provides an additional capacity axis for MIDUS without increasing model depth.

Table 9: Component-wise ablation of MIDUS on Llama-3.2-1B after CPT on `FineWeb-Edu`. Head Query denotes direct use of attention-head outputs as memory queries. Distinct Keys denotes separate product-key spaces across heads. Head Realization denotes shared value storage with head-specific projections.

Method	Head Query	Distinct Keys	Head Realization	PPL ↓	Average ↑
Block-level PKM	–	–	–	11.67	52.36
Head-wise PKM w/ Shared Values	✓	✓	–	11.86	52.52
Global Retrieval w/ Head Realization	–	–	✓	11.65	53.21
Head-wise Queries w/ Shared Key Map	✓	–	✓	11.76	52.94
MIDUS	✓	✓	✓	**11.64**	**53.50**

Table 10: Ablation study on the number of total memory slots and its effect on performance. The 66K, 262K, and 1M settings denote the total addressable slots across all inserted blocks and attention heads; each head uses $N = n^2$ composite keys.

	PPL ↓	Zero-shot Accuracy ↑							
# Memories	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
66K ($n = 16$)	**11.63**	66.12	22.89	60.06	46.27	65.02	74.81	36.74	53.13
262K ($n = 32$)	11.64	65.95	**23.35**	60.85	**47.17**	**65.44**	74.76	36.41	53.42
1M ($n = 64$)	11.64	**66.16**	23.20	**61.56**	46.27	65.29	**75.08**	**36.91**	**53.50**

A.8.3 Ablation study on Top-k

In Table 11, we vary the number of retrieved memory slots per token from k=1 to k=8 while keeping the memory size and all other hyperparameters fixed. Average zero-shot accuracy improves steadily as k increases from 1 to 4, with modest or no further gains at k=8, and PPL remains essentially unchanged across all settings. We therefore use k=4 as the default Top-k setting in all other experiments.

Table 11: Effect of the Top-k value k for Llama-3.2-1B with MIDUS.

	Train Time ↓	TTFT ↓	PPL ↓	Zero-shot Accuracy ↑							
Top-k	s/iter	ms	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
1	**4.31**	9.33	11.67	65.74	21.97	60.54	46.19	64.95	74.86	36.95	53.03
2	4.41	9.40	**11.63**	65.87	**23.20**	60.69	46.11	65.11	74.54	37.04	53.22
4	4.54	**9.31**	11.64	**66.16**	**23.20**	**61.56**	46.27	**65.29**	**75.08**	36.91	**53.50**
8	4.81	10.71	11.64	66.04	23.04	60.46	**46.93**	64.98	75.03	**37.11**	53.37

A.8.4 Ablation study on the learning rate and weight decay

Table 12: Effect of learning rate (LR) and weight decay (WD) on the performance of Llama-3.2-1B with MIDUS

		PPL ↓	Zero-shot Accuracy ↑							
LR	WD	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
1e-4	1e-6	11.64	66.16	**23.20**	**61.56**	46.27	65.29	**75.08**	36.91	**53.50**
1e-4	0	11.64	66.25	21.97	60.77	46.60	65.32	75.03	36.99	53.27
1e-4	1e-2	**11.62**	65.78	23.04	59.91	45.45	**65.35**	74.81	**37.25**	53.09
1e-5	1e-6	11.64	65.57	21.51	61.01	46.85	63.58	74.48	37.12	52.87
5e-5	1e-6	11.63	**66.46**	22.89	60.14	**47.58**	64.86	74.54	37.23	53.39

In Table 12, we vary the learning rate and weight decay over a range of reasonable values while keeping all other settings fixed. Across these configurations, PPL remains tightly clustered around 11.62–11.64 and the average zero-shot accuracy stays in a narrow band, indicating that MIDUS is relatively robust to these optimization hyperparameters.

A.8.5 Additional Structural Ablations

In Table 13, we evaluate four additional structural variants of MIDUS. The first applies BatchNorm1d to the queries, following PKM (Lample et al., 2019). The second applies LayerNorm to the queries. The third adds an internal attention residual, $a' = x + \mathrm{Attn}'(\mathrm{LN}(x))$. The fourth enables the attention output projection. Neither query-normalized variant improves over MIDUS, suggesting that

normalization may disturb the score structure used for Top-k selection. The internal residual does not improve aggregate performance, which indicates that directly using separated head outputs may provide more suitable memory queries than mixing them with the input representation. Although adding the output projection gives a marginal gain, we retain the projection-free design as the default because it preserves head-separated queries and avoids the additional projection cost.

Table 13: Additional structural ablations of MIDUS.

	PPL ↓	Zero-shot Accuracy ↑							
Method	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
MIDUS	**11.64**	66.16	23.20	**61.56**	46.27	**65.29**	75.08	36.91	53.50
w/ BatchNorm1d	11.74	**66.54**	22.43	60.77	**47.34**	64.80	74.92	36.94	53.39
w/ LayerNorm	11.68	66.08	22.43	**61.56**	46.68	65.17	75.19	**37.34**	53.49
w/ Residual	**11.64**	66.04	22.89	61.48	46.36	65.17	**75.35**	36.75	53.43
w/ Output Projection	**11.64**	66.12	**23.66**	61.09	46.52	65.23	74.86	37.07	**53.51**

A.8.6 Ablation study on DUS policy

Table 14: MIDUS under different DUS placement policies. Peak GPU Memory and Time are training requirements. *Top-heavy* places expanded blocks toward the top, *Distributed* interleaves, and *Bottom-heavy* places them near the input.

	GB ↓	s/iter ↓	PPL ↓	Zero-shot Accuracy ↑							
DUS Policy	GPU Memory	Time	Wiki	ARC	LogiQA	Wino	CSQA	BoolQ	PIQA	MMLU	Average
Top-heavy	**24.1**	**3.89**	11.63	66.12	22.27	61.25	**46.44**	64.28	74.92	36.23	53.07
Distributed	28.0	4.56	11.64	**66.16**	**23.20**	**61.56**	46.27	**65.29**	75.08	**36.91**	**53.50**
Bottom-heavy	28.6	4.67	**11.60**	66.12	21.81	60.30	45.05	65.05	**75.30**	36.35	52.85

Table 14 shows the efficiency and performance of MIDUS with Llama-3.2-1B under different DUS placement policies. *Top-heavy* placement, as used by OpT-DeUS (Cao et al., 2025), is most efficient in memory and time, since many frozen lower blocks can be skipped during backpropagation. *Bottom-heavy* placement slightly improves PPL but does not maximize average zero-shot accuracy. *Distributed* policy yields the best overall accuracy.

A.9 Additional efficiency experiments

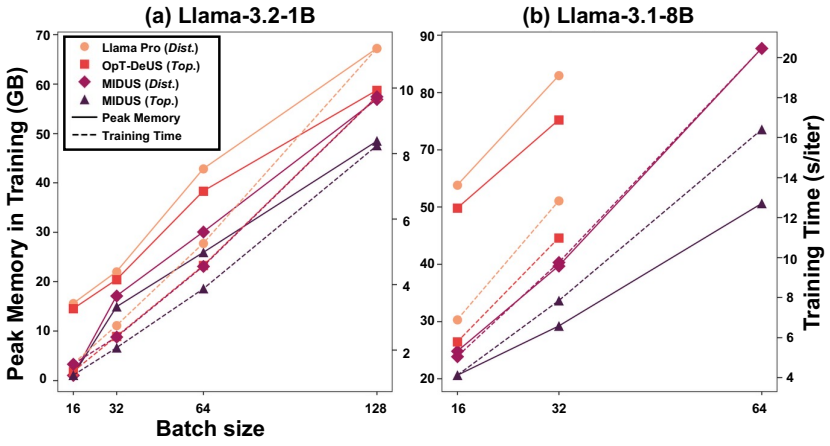

Figure 5: Efficiency of DUS baselines and MIDUS as a function of batch size. *Dist.* denotes the *Distributed* and *Top.* the *Top-heavy* DUS placement policy.

We measure all efficiency metrics on a single NVIDIA RTX 6000 Blackwell GPU. For Table 3, we use a global batch size of 16, 16 gradient-accumulation steps, and a sequence length of 2,048 tokens.

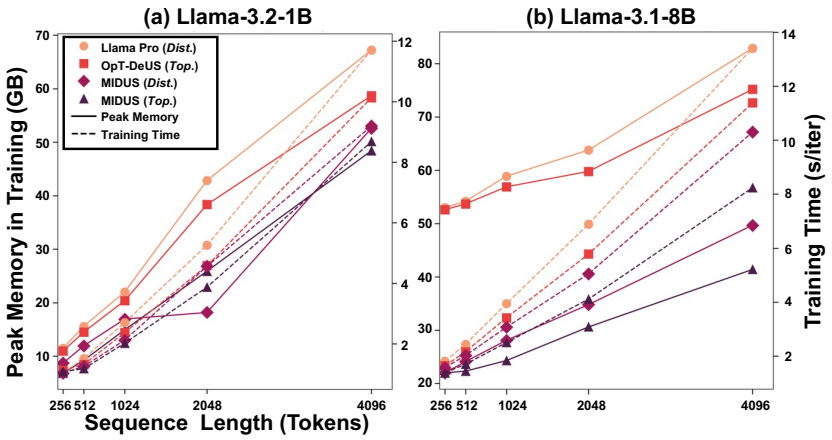

Figure 6: Efficiency of DUS baselines and MIDUS as a function of sequence length. *Dist.* denotes the *Distributed* and *Top.* the *Top-heavy* DUS placement policy.

Figure 5 varies the global batch size while fixing the sequence length at 2,048. Figure 6 varies the sequence length under the default batch configuration.

For Llama-3.1-8B, we omit training-time results for Llama Pro and OpT-DeUS at the largest evaluated batch size because they exceed the available GPU memory. Across both batch-size and sequence-length settings, MIDUS requires less peak training memory than the DUS baselines. MIDUS with the *Top-heavy* policy also remains faster than OpT-DeUS under the same placement policy, while the default *Distributed* configuration provides a favorable balance between performance and efficiency.

A.10 Structural Analysis of Head-wise Value Realization

This appendix provides the detailed analysis of head-wise value realization while treating retrieval as fixed. We first characterize the loss incurred by sharing a single realized value bank across heads. We then analyze when the shared storage and head-specific projections used by MIDUS can recover the optimum of fully independent head-specific value banks.

A.10.1 Fixed Retrieval Setup

For each head h, let $A_h \in \mathbb{R}^{T\times N}$ denote the fixed retrieval-weight matrix whose t-th row is $\alpha_{h,t}^\top$. Let $Y_h \in \mathbb{R}^{T\times d_h}$ denote the target memory output for head h.

We consider three value parameterizations under the same retrieval weights. A shared realized value bank uses $V_{\mathrm{sh}} \in \mathbb{R}^{N\times d_h}$ and produces $A_h V_{\mathrm{sh}}$ for every head. Independent head-specific banks use $V_h \in \mathbb{R}^{N\times d_h}$ and produce $A_h V_h$. MIDUS uses a shared bank $\bar{V} \in \mathbb{R}^{N\times d_h}$ with head-specific projections $W_h \in \mathbb{R}^{d_h\times d_h}$ and produces $A_h \bar{V} W_h$.

The corresponding optimal losses are

$$\mathcal{L}^\star_{\mathrm{Share}} = \inf_{V_{\mathrm{sh}}} \frac{1}{2}\sum_{h=1}^{H} \|A_h V_{\mathrm{sh}} - Y_h\|_F^2, \qquad \mathcal{L}^\star_{\mathrm{Ind}} = \inf_{\{V_h\}_{h=1}^H} \frac{1}{2}\sum_{h=1}^{H} \|A_h V_h - Y_h\|_F^2, \tag{20}$$

$$\mathcal{L}^\star_{\mathrm{MIDUS}} = \inf_{\bar{V},\{W_h\}_{h=1}^H} \frac{1}{2}\sum_{h=1}^{H} \left\|A_h \bar{V} W_h - Y_h\right\|_F^2. \tag{21}$$

A.10.2 Loss Gap from Shared Value Realization

Under fixed retrieval, the independent value banks can be optimized separately for each head. Their optimal loss is

$$\mathcal{L}^\star_{\mathrm{Ind}} = \frac{1}{2}\sum_{h=1}^{H} \|(I_T - \Pi_{A_h})Y_h\|_F^2, \tag{22}$$

where Π_{A_h} denotes the orthogonal projection onto $\mathrm{col}(A_h)$.

Proposition A.1. *For arbitrary fixed retrieval matrices $\{A_h\}_{h=1}^H$,*

$$\mathcal{L}^\star_{\mathrm{Share}} = \mathcal{L}^\star_{\mathrm{Ind}} + \frac{1}{2}\sum_{h=1}^{H} \|A_h V^\star_{\mathrm{sh}} - \Pi_{A_h} Y_h\|_F^2, \tag{23}$$

where a minimum-norm optimal shared bank is

$$V^\star_{\mathrm{sh}} = \left(\sum_{h=1}^{H} A_h^\top A_h\right)^\dagger \sum_{h=1}^{H} A_h^\top Y_h,$$

where $\dagger$ denotes the Moore–Penrose pseudoinverse. Therefore, $\mathcal{L}^\star_{\mathrm{Share}} \geq \mathcal{L}^\star_{\mathrm{Ind}}$. Equality holds if and only if there exists a common V_{sh} such that $A_h V_{\mathrm{sh}} = \Pi_{A_h} Y_h$ for every $h \in [H]$.

Proof. For each head, the independent optimization $\inf_{V_h} \frac{1}{2}\|A_h V_h - Y_h\|_F^2$ is a least-squares problem. Its fitted output is $\Pi_{A_h} Y_h$, which gives Eq.(22).

For the shared realized value bank, the normal equation is

$$\left(\sum_{h=1}^{H} A_h^\top A_h\right) V_{\mathrm{sh}} = \sum_{h=1}^{H} A_h^\top Y_h,$$

which gives the stated minimum-norm solution.

For every head, $A_h V^\star_{\mathrm{sh}} - \Pi_{A_h} Y_h$ lies in $\mathrm{col}(A_h)$, while $(I_T - \Pi_{A_h})Y_h$ lies in its orthogonal complement. Therefore,

$$\|A_h V^\star_{\mathrm{sh}} - Y_h\|_F^2 = \|A_h V^\star_{\mathrm{sh}} - \Pi_{A_h} Y_h\|_F^2 + \|(I_T - \Pi_{A_h})Y_h\|_F^2. \tag{24}$$

Summing over heads gives Eq.(23). The inequality and equality condition follow directly. □

When retrieval is identical across heads, the gap has a simpler form.

Corollary A.2. *Suppose $A_h = A$ for all $h \in [H]$ and define $\bar{Y}_A = \frac{1}{H}\sum_{h=1}^{H} \Pi_A Y_h$. Then*

$$\mathcal{L}^\star_{\mathrm{Share}} - \mathcal{L}^\star_{\mathrm{Ind}} = \frac{1}{2}\sum_{h=1}^{H} \left\|\Pi_A Y_h - \bar{Y}_A\right\|_F^2 . \tag{25}$$

Proof. Under $A_h = A$, the optimal shared output is $AV^\star_{\mathrm{sh}} = \bar{Y}_A$. Substituting this into Eq.(23) gives Eq.(25). □

Eq.(25) shows that the penalty from sharing a realized value bank is determined by variation among the projected head targets. The gap vanishes when all heads require the same output within the common retrieval span and becomes positive when their projected targets differ.

A.10.3 Shared Storage with Head-specific Value Realization

MIDUS does not store an independent value bank for every head. Instead, it represents each realized head-specific bank as $\bar{V}W_h$ using shared storage $\bar{V}$ and a head-specific projection W_h. We next characterize how closely this parameterization can match fully independent banks.

Proposition A.3. *Let $\{V_h^\star\}_{h=1}^H$ be an optimal independent solution. Suppose there exist $\bar{V} \in \mathbb{R}^{N\times d_h}$, $W_h \in \mathbb{R}^{d_h\times d_h}$, and $R_h \in \mathbb{R}^{N\times d_h}$ such that $V_h^\star = \bar{V}W_h + R_h$ for every $h \in [H]$ and $\sum_{h=1}^H \|R_h\|_F^2 \le \varepsilon$. Define $\rho = \max_{h\in[H]} \|A_h\|_{\mathrm{op}}^2$, where $\|\cdot\|_{\mathrm{op}}$ denotes the operator norm. Then*

$$\mathcal{L}^\star_{\mathrm{MIDUS}} \le \mathcal{L}^\star_{\mathrm{Ind}} + \frac{\rho}{2}\varepsilon. \tag{26}$$

Proof. For the candidate MIDUS parameterization given by $\bar{V}$ and $\{W_h\}_{h=1}^H$, define $E_h = V_h^\star - \bar{V}W_h = R_h$. The difference between the optimal independent output and the MIDUS output is A_hE_h. By the operator-Frobenius bound,

$$\|A_hE_h\|_F^2 \le \|A_h\|_{\mathrm{op}}^2\|E_h\|_F^2 \le \rho\|E_h\|_F^2.$$

Since $A_hV_h^\star = \Pi_{A_h}Y_h$, the independent residual $A_hV_h^\star - Y_h$ is orthogonal to A_hE_h. Therefore,

$$\|A_h\bar{V}W_h - Y_h\|_F^2 = \|A_hV_h^\star - Y_h\|_F^2 + \|A_hE_h\|_F^2.$$

Summing over heads and using $\sum_h \|E_h\|_F^2 \le \varepsilon$ gives Eq.(26). Since $\mathcal{L}^\star_{\mathrm{MIDUS}}$ is the optimum over all MIDUS parameterizations, it cannot exceed the loss of this candidate. □

Proposition A.3 shows that the loss relative to independent value banks is controlled by how well those banks admit a common shared-storage factorization. In the exact factorization case, the residual term vanishes.

A.10.4 Proof of Theorem 3.1

Theorem 3.1. *Suppose $A_h = A$ for all $h \in [H]$, and let Π_A denote the orthogonal projection onto the column space of A. Define $\bar{Y}_A = \frac{1}{H}\sum_{h=1}^{H} \Pi_A Y_h$. Then*

$$\mathcal{L}^\star_{\mathrm{Share}} - \mathcal{L}^\star_{\mathrm{Ind}} = \frac{1}{2}\sum_{h=1}^{H} \left\|\Pi_A Y_h - \bar{Y}_A\right\|_F^2 . \tag{14}$$

Furthermore, suppose an optimal independent solution $\{V_h^\star\}_{h=1}^H$ admits a shared factorization $V_h^\star = \bar{V}W_h$, where $\bar{V} \in \mathbb{R}^{N\times d_h}$ and $W_h \in \mathbb{R}^{d_h\times d_h}$. Then $\mathcal{L}^\star_{\mathrm{MIDUS}} = \mathcal{L}^\star_{\mathrm{Ind}}$. Consequently, if $\Pi_A Y_h \ne \Pi_A Y_{h'}$ for some $h \ne h'$, then $\mathcal{L}^\star_{\mathrm{MIDUS}} < \mathcal{L}^\star_{\mathrm{Share}}$.

Proof. Suppose the optimal independent banks satisfy $V_h^\star = \bar{V}W_h$ for every $h \in [H]$. This is the zero-residual case of Proposition A.3. Therefore, $\mathcal{L}^\star_{\mathrm{MIDUS}} \le \mathcal{L}^\star_{\mathrm{Ind}}$.

Every MIDUS realization $\bar{V}W_h$ is also a valid choice in the independent family. Hence $\mathcal{L}^\star_{\mathrm{MIDUS}} \geq \mathcal{L}^\star_{\mathrm{Ind}}$, which gives $\mathcal{L}^\star_{\mathrm{MIDUS}} = \mathcal{L}^\star_{\mathrm{Ind}}$.

Under the common-retrieval assumption $A_h = A$, Corollary A.2 gives

$$\mathcal{L}^\star_{\mathrm{Share}} - \mathcal{L}^\star_{\mathrm{Ind}} = \frac{1}{2} \sum_{h=1}^{H} \left\| \Pi_A Y_h - \bar{Y}_A \right\|_F^2 . \tag{27}$$

If $\Pi_A Y_h \neq \Pi_A Y_{h'}$ for some $h \neq h'$, the projected targets cannot all equal their mean $\bar{Y}_A$. The right-hand side is therefore strictly positive. Combining this with $\mathcal{L}^\star_{\mathrm{MIDUS}} = \mathcal{L}^\star_{\mathrm{Ind}}$ gives

$$\mathcal{L}^\star_{\mathrm{MIDUS}} = \mathcal{L}^\star_{\mathrm{Ind}} < \mathcal{L}^\star_{\mathrm{Share}}.$$

□

The theorem isolates the role of head-specific value realization independently of retrieval. Even when all heads use the same retrieval-weight matrix, a shared realized value bank incurs a positive gap whenever the projected target outputs differ across heads. MIDUS can retain the independent-bank optimum when those head-specific banks admit the shared factorization used by its value parameterization. This formalizes the separation between shared storage and head-specific utilization described in Section 3.2.

Connection to FFN-based DUS. Prior work interprets Transformer FFNs as implicit key–value memories in which the first transformation and nonlinearity induce key-like activations and the second transformation provides value-like outputs (Geva et al., 2021). Under this view, duplicating FFN branches in DUS increases memory-like capacity over the full hidden representation.

Our analysis does not claim that an arbitrary FFN is equivalent to the shared value family above. FFNs include nonlinear activation and dense feature mixing that are absent from the fixed-retrieval setup. Instead, the analysis isolates a value-side distinction that is relevant to MIDUS. A shared block-level realization must use the same stored value representation across heads, whereas MIDUS shares the stored content while allowing each attention head to realize that content through its own projection.