Mixture-of-Memory Embeddings for Context-Aware Sparse Lookup

Muchen Li[1,2] **Leonid Sigal**[1,2,3,4,†] **Renjie Liao**[1,2,3,†]

[1]University of British Columbia
[2]Vector Institute for AI [3]Canada CIFAR AI Chair [4]NSERC CRC Chair

Abstract

Scaling large language models efficiently has motivated sparse capacity mechanisms such as Mixture-of-Experts and, more recently, conditional memory: token-indexed embedding tables that augment the backbone with cheap parametric lookups. Existing memory-embedding methods retrieve via a deterministic function of the surface form, which collapses different contextual senses of the same token (*e.g.*, *python* the language vs. the animal) into a single fixed entry. We introduce Mixture of Memory Embeddings (MoME), a context-aware memory mechanism that replaces each token's single memory row with a mixture of M slots and uses a learned gate over the hidden state to choose which slots to read at each position. In controlled pretraining experiments across nanochat, Llama-3/MobileLLM, and Qwen3 backbones, MoME improves over Value Embedding, Bigram, and STEM baselines in iso-parameter and iso-training-FLOP settings, shows a more promising memory-size scaling trend at sub-billion scale, and remains efficient in training and inference. Qualitative routing analyses on polysemous tokens further suggest that the learned mixture exhibits a degree of semantic interpretability, dispatching the same surface token to distinct memory slots under different senses. The code and pretrained models are open-sourced.

1 Introduction

Efficient scaling for large language models has been a central challenge for extending their capability boundary. Dense scaling improves performance [21, 28], but it ties capacity growth to broadly active computation. Mixture-of-Experts (MoE) addresses this by routing each token to only a subset of experts, enabling conditional computation, and is now standard in frontier models [14, 15, 25, 32, 57].

More recently, a second axis of sparse scaling—conditional memory—has emerged as a complementary route to expanding model capacity [6, 54]. The motivation is that language modeling interleaves two sub-tasks: compositional reasoning, demanding dynamic computation, and retrieval of local, static patterns like named entities and formulaic phrases. Lacking a native lookup primitive, Transformers simulate retrieval through computation, spending early-layer capacity to reconstruct a static table. Conditional memory instead handles such regularities through sparse lookups: each token retrieves only a few memory entries that inject useful priors into the backbone [1, 24, 31, 58, 62]. Rather than storing all useful associations only in dense transformer weights, memory-embedding methods learn auxiliary tables that can be looked up and injected into the backbone, including Per-Layer Embedding in Gemma 3 [16], value-stream memory variants [26, 29, 30, 68], STEM [54], Engram [6], and Bigram [9]. The appeal is efficiency: only a small fraction of the table is active for any token, the lookup is lightweight, and the table can therefore be scaled with modest additional compute.

[†]Equal advising. Contact: muchenli@cs.ubc.ca

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

These methods differ in where memory is injected, but they share a common restriction in how it is indexed. Existing memory-embedding methods usually retrieve memory through deterministic token or local n-gram indexing: Per-Layer Embedding uses token identity [16]; Value Embedding and related value-stream variants use token identity [26, 29, 30, 68]; STEM uses token-indexed embedding modules [54]; and Engram uses a fixed n-gram hash [6]. Broader work on embedding, vocabulary, and n-gram scaling further motivates this direction [22, 35, 53, 60, 65]. This makes retrieval simple, but it also makes the retrieved memory largely context-blind. The same token can call for different associations in different contexts: *python* may refer to a programming language or an animal, and *spring* may refer to a season, a mechanical coil, or a verb. A single deterministic memory vector for such tokens forces these contextual modes to share one vector.

To address this shortcoming, we introduce Mixture of Memory Embeddings (MoME), a *context-aware* conditional memory mechanism that is more expressive but equally efficient, retaining the access pattern of token-indexed lookups. Instead of assigning each token row a single fixed memory vector, MoME stores a mixture of memory slots and uses the current hidden context to choose which components of that mixture to read. This gives memory retrieval a simple form of contextual adaptivity while keeping the mechanism close to the efficient lookup structure used by prior memory embeddings.

We evaluate MoME in controlled pretraining experiments across three architecture families: nanochat-style [29], Llama 3/MobileLLM-style [18, 36], and Qwen3-style backbones [63]. Across these settings, MoME improves over comparable memory-augmented transformer baselines in most matched comparisons, including Base, Value Embedding [30, 68], STEM [54], and Bigram [9] baselines. We also study memory-size scaling on the nanochat-style backbone in a low-compute regime, where MoME shows a more favorable observed scaling trend than Bigram [9] over the tested memory range. A complementary ablation further indicates that MoME is compatible with Bigram under compound scaling. In addition, qualitative and quantitative routing analyses on a curated set of polysemous tokens (e.g., *apple*, *bank*, *python*) show that MoME dispatches the same token to distinct memory slots under different semantic contexts on both nanochat- and Qwen3-style backbones, indicating that the learned mixture captures contextual variation rather than committing to one route per token.

Our contributions are:

- We design MoME, a mixture-of-memory module that brings context awareness into the memory table lookup of token-indexed memory embeddings.
- We show that MoME achieves competitive performance against prior state-of-the-art memory-embedding methods and transfers well across diverse model architectures.
- We further show that MoME exhibits a more favorable memory-size scaling trend than existing baselines, suggesting better returns as memory capacity grows; additionally, MoME can be applied together with Bigram to achieve better performance.
- Routing analyses show that the proposed module learns to dynamically index different memory slots in context, with the selected slots reflecting the semantics of the input.

2 Related Work

Learning memories of Large Language Models. Memory-augmented LMs span non-parametric retrieval [4, 19] and end-to-end memory networks [58, 62]. Both are orthogonal to MoME, which keeps memory parametric and token-indexed. Closer to our setting, FFN layers behave as key-value memories [10, 17, 38], motivating explicit memory layers with sub-linear lookup [1, 24, 31]. MoME shares this parametric premise but injects into the per-head value stream and indexes by token identity, trading content-addressable lookup for a lighter retrieval path.

Memory-Embedding Augmented LLMs. A recent convergent line attaches a learnable embedding table to the transformer and looks it up by a deterministic function of the input. Per-Layer Embedding in Gemma 3n [16] uses per-token-id rows at each layer's input; Value Embedding [26, 29, 30, 68] attaches the table to the value stream; STEM [54] fuses a token-indexed lookup into the SwiGLU hidden state; Engram [6] uses deterministic n-gram lookup followed by hidden-state-conditioned fusion; our Bigram baseline [9] is a separate implementation using only 2-gram lookup; MoWE [43]

Table 1: Comparison of how prior works retrieve and inject memory embeddings. Context aggregation describes memory addressing; canonical Engram additionally conditions fusion on the hidden state.

Method	Memory index	Index function	Injection position	Context aggregation
Per-Layer Embedding [16]	$V \times d_{\text{model}}$	x_t	Input embedding	None
Value Embedding [68]	$V \times H \times d_{\text{value}}$	x_t	Attention value	None
STEM [54]	$V \times d_{\text{ffn}}$	x_t	SwiGLU hidden	None
Engram (canonical) [6]	$N_{\text{hash}} \times d_{\text{model}}$	$h(x_t, \ldots, x_{t-n})$	Block input	n-gram
Ours	$N \times M \times d_{\text{value}}$	$[f(x_t),\ \mathcal{A}_{t,i}]$	Attention value	Hidden embedding

routes tokens deterministically into many small word-experts. Across this family, retrieval is a fixed function of the surface form, not of the hidden state – the gap MoME closes via a context-aware mixture-of-slots while keeping the cheap lookup. A parallel sub-line scales the embedding table itself. SCONE [65] learns frequent-n-gram embeddings via an auxiliary contextualizer and offloads them at inference; Over-Tokenized Transformer [22] feeds many overlapping n-gram embeddings, and Byte Latent Transformer [46] does so at the byte level. These extend earlier modular-hash precursors [23, 35, 53] and are motivated by vocabulary scaling laws [60]. Routing here also remains a fixed function of the surface form. MoME's value-stream injection relates to cross-layer value designs: ResFormer [68] adds a residual from the first layer's value vectors; NeuTRENO [42] regularizes value vectors across layers; DenseFormer [45] depth-weight-averages hidden states; Cross-Layer Attention variants [5, 41] share K/V across layers. MoME chooses this site so the memory branch runs in parallel with the value projection, joining only at a gated residual addition (Section 3.5).

Mixture of Experts. MoME's slot gate borrows from Mixture-of-Experts: the original sparsely-gated MoE [57] and its descendants [14, 15, 25, 32] decoupled capacity from per-token FLOPs. Closer to our setting are fine-grained designs that split each FFN into many small specialists with optional shared experts [11, 12, 20]. MoME applies the same idea on the *memory-table* side: each token-indexed row holds M slots, and the gate selects a subset at the current position.

3 Mixture of Memory Embeddings

3.1 Preliminaries – Token-Indexed Table as Learnable Memory

Prior works on memory-augmented transformers [6, 16, 54, 68] (Table 1) can be written as learning an embedding table $\mathbf{E} \in \mathbb{R}^{N \times D_{\text{inj}}}$, where N is the number of addressable memory entries and D_{inj} is the dimensionality required by the injection site. For each input token x_t, the memory table is indexed deterministically, either directly by token id ($N = V$) or by a hash function over an n-gram window ($N = N_{\text{hash}}$). These works further differ in where this retrieved embedding is injected into the backbone. Per-Layer Embedding [16] injects at the input embedding, Engram [6] fuses retrieved memory into block-input hidden states through a context-dependent gate, value embedding [26, 29, 68] injects into the attention value stream, and STEM fuses the retrieved embedding into the SwiGLU hidden state [56]. Table 1 organizes the representative methods along these axes; across the space the lookup remains a deterministic function of x_t alone (or of a fixed n-gram), so memory addressing does not depend on the model's hidden state at position t, even when fusion is context-dependent.

We argue that token-deterministic indexing has one underlying mismatch: capacity is allocated to tokens rather than to semantic content. This manifests as two compounding drawbacks:

(i) Misallocated capacity. One slot per token treats all tokens as if they carried equal semantic load, but they do not. Some tokens are near-redundant: variants like *cats*/*cat* largely overlap in meaning, yet each occupies its own row and the table duplicates the same content; other tokens are the opposite—a single row is asked to hold several distinct meanings, e.g., *bank* (financial institution vs. river edge), with no room to keep them apart.

(ii) Context-blind retrieval. Deterministic lookup makes memory retrieval blind to the full context.[1] Token semantics vary substantially across contexts—*python* as a programming language versus

[1] Canonical Engram uses 2- and 3-gram lookup; our Bigram baseline uses only 2-grams.

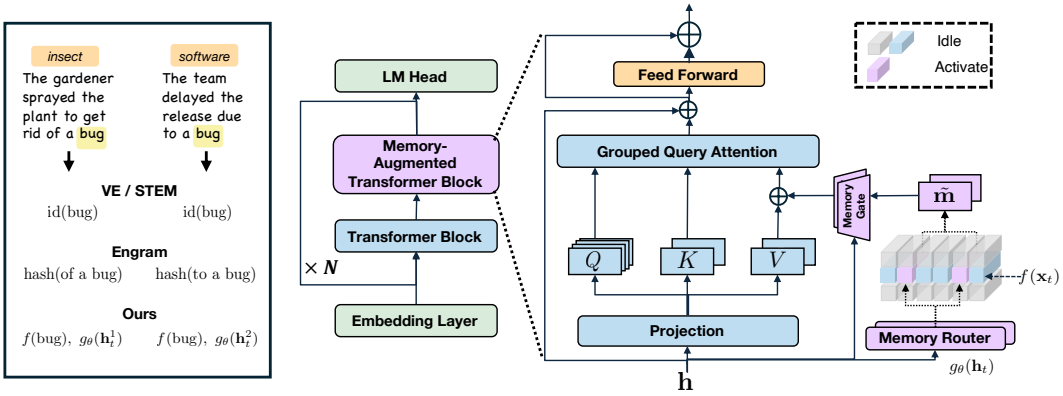


Figure 1: **Left:** the same token can carry different senses, and prior token-indexed methods (VE/STEM, canonical Engram) commit each row to a single fixed memory while MoME dispatches to different slots depending on the hidden state (see Figure 3 and Section A.11 for the learned routing). **Right:** MoME alternates regular transformer blocks with memory-augmented blocks; in each memory-augmented block, a context-aware router retrieves a memory vector and injects it into the attention value stream.

an animal—so memory for a token should be multi-modal across contexts, yet a deterministic table dedicates a single memory slot per token.

3.2 Learning Mixture of Memory Embeddings

Designing Mixture of Memory Embeddings Tables. Motivated by the limitations of current memory designs, we introduce Mixture of Memory Embeddings, which extends the prior token-indexed memory tables to resolve the limitation mentioned in Section 3.1.

Specifically, we introduce two designs: (i) **A mixture of M memory slots with a learned context-aware gate.** Motivated by Mixture-of-Experts models [57], we extend each token-indexed row to M memory slots with a learned context-aware gate to selectively activate the memory embedding. (ii) **Grouped token indexes f.** Aiming to reduce the redundancy in the memory table, we introduce an optional deterministic function to group tokens with similar semantics together. We find that this improves training efficiency.

Concretely, the memory is a learnable parameter tensor $\mathbf{E}^{\text{mem}} \in \mathbb{R}^{N \times M \times d_{\text{value}}}$, where N is the number of rows selected by the first-stage indexer f ($N = V$ for identity indexing and $N \leq V$ after grouping), M is the number of slots per row, and d_{value} is the per-head value dimension at the memory injection point. We write the stored memory vector at row n and slot a as $\mathbf{m}_{n,a} = \mathbf{E}^{\text{mem}}[n, a, :] \in \mathbb{R}^{d_{\text{value}}}$.

Figure 1 shows the overall Mixture of Memory Embeddings architecture: regular transformer blocks alternate with memory-augmented transformer blocks. Each memory-augmented block keeps the standard transformer computation and attaches a side memory module: the module reads the block input, retrieves a context-dependent memory vector, and injects it back into the attention value stream for each value head. Here, x_t denotes the input token at position t, $\mathbf{h}_t$ denotes the hidden state entering the memory-augmented block, and $\mathcal{A}_{t,i}$ denotes the active slot set selected for value head i. We also show the two memory indexers explicitly: $f(x_t) \to n_t$ selects the memory row, and $g_\theta(\mathbf{h}_t) \to (\mathcal{A}_{t,i}, \alpha)$ selects and weights slots within that row before value injection.

3.3 Context-Aware Routing and Aggregation

Context-Aware Routing. Mixture of Memory Embeddings can be viewed as composing two memory indexers. The token indexer $f(x_t)$ selects the row n_t, while the context indexer $g_\theta(\mathbf{h}_t)$ selects and weights slots within that row. Given $n_t = f(x_t)$, the second-stage gate decides which of the M slots in that row are selected at the current position. This is where context enters the lookup: rather than committing the row to a single fixed vector, a learned gate over the hidden state selects the memory slots dynamically. For each value head $i \in [H]$, the gate is implemented by first mapping

the hidden state to slot logits

$$\boldsymbol{\ell}_{t,i} = \mathbf{W}_g^{(i)}\mathbf{h}_t + \mathbf{b}_g^{(i)} \in \mathbb{R}^M.$$

Here $\boldsymbol{\ell}_{t,i}$ are the routing logits used for memory-slot selection. Conditioning the slot gate on $\mathbf{h}_t$ is what makes routing context-aware, since $\mathbf{h}_t$ has already integrated context through the upstream transformer layers.

$$\mathcal{A}_{t,i} = \mathrm{TopK}(\boldsymbol{\ell}_{t,i}, K), \qquad \mathcal{A}_{t,i} \subseteq [M], \qquad |\mathcal{A}_{t,i}| = K.$$

Memory Aggregation. To aggregate memory, we use the sigmoid-norm gate for $K > 1$, converting slot logits into scores and normalizing only over the selected slots

$$\mathbf{s}_{t,i} = \sigma(\boldsymbol{\ell}_{t,i}), \qquad \alpha_{t,a}^{(i)} = \frac{s_{t,i,a}}{\sum_{b\in\mathcal{A}_{t,i}} s_{t,i,b}}, \quad a \in \mathcal{A}_{t,i}.$$

For $K = 1$, we use the softmax variant inline, $\alpha_{t,a}^{(i)} = [\mathrm{softmax}(\boldsymbol{\ell}_{t,i})]_a$.

Given $n_t = f(x_t)$ and the mixture weights $\{\alpha_{t,a}^{(i)}\}_{a\in\mathcal{A}_{t,i}}$, the aggregated memory for head i is constructed as the weighted sum of activated stored memory vectors:

$$\tilde{\mathbf{m}}_{t,i} = \sum_{a\in\mathcal{A}_{t,i}} \alpha_{t,a}^{(i)} \mathbf{m}_{n_t,a} \in \mathbb{R}^{d_{\mathrm{value}}}.$$

3.4 Token-Index Grouping Function

Optionally, training efficiency can be further improved by optimizing the token indexer itself: tokens with similar semantics are grouped into shared rows, reducing redundant token-indexed capacity while preserving context-aware slots within each row. We design the token-based indexer as a function

$$f : \mathcal{V} \to [N], \qquad N \le V,$$

that maps tokens with overlapping semantics into the same row. We tried several similarity heuristics and found that embedding-based matching works best: we instantiate f from a lightly pretrained token-embedding matrix $\mathbf{U}^{\mathrm{ref}} \in \mathbb{R}^{V\times d_{\mathrm{value}}}$ (obtained from a baseline run with $f = \mathrm{id}$) by offline kNN matching in pretrained embedding space,

$$f(x) = \mathrm{kNN}\big(\mathbf{U}_x^{\mathrm{ref}};\, \mathbf{U}^{\mathrm{ref}},\, c_{\mathrm{grp}}\big),$$

yielding a fixed slot map of grouping factor c_{grp} (*e.g.*, c_{grp}=2, 4 in our main configurations) that is frozen for the duration of training; here $\mathrm{kNN}(\cdot)$ denotes the offline row-map construction, not a runtime retrieval. Grouping shrinks the token-indexed row dimension by c_{grp}, and the saved capacity is offloaded to context-aware memory by increasing the number of slots M per row by c_{grp}.

3.5 Memory Injection

Gated Injection. The aggregated memory $\tilde{\mathbf{m}}_{t,i}$ is fused into the attention value stream through a separate per-head value-residual gate

$$\boldsymbol{\gamma}_\theta(\mathbf{h}_t) = 2\sigma(\mathbf{W}_\gamma\mathbf{h}_t + \mathbf{b}_\gamma) \in \mathbb{R}^H,$$
$$\tilde{\mathbf{v}}_{t,i} = \mathbf{v}_{t,i} + \gamma_{\theta,t,i}\tilde{\mathbf{m}}_{t,i}, \qquad \gamma_{\theta,t,i} = [\boldsymbol{\gamma}_\theta(\mathbf{h}_t)]_i.$$

Thus g_θ controls which memory slots are aggregated, while $\boldsymbol{\gamma}_\theta$ controls how strongly the aggregated memory is added to the value stream. The factor of 2 makes $\gamma_{\theta,t,i} = 1$ when the gate logit is initialized at zero, giving a neutral initial value-residual scale.

Latency-Friendly Design. The memory branch introduces extra per-head computation: the router projects hidden states to slot scores, performs top-K selection, and aggregates the selected memory vectors. Although this branch is lightweight in FLOPs, it can still add inference latency because the additional kernels sit on the critical path. A benefit of conditioning on $\mathbf{h}_t$ and injecting into the value stream is that much of the memory branch can be scheduled in parallel with the standard value projection that produces $\mathbf{v}_{t,i}$. After both branches finish, the only required join is the gated residual addition that produces $\tilde{\mathbf{v}}_{t,i}$. In the `qwen3_4b` two-memory-layer benchmark, this design adds 0.509 ms, while the hidden-state-injection variant of MoME adds 0.996 ms, nearly doubling the latency overhead. The headline numbers across backbones are summarized in Table 5; the full per-backbone latency sweep is in Appendix Table 21.

Table 2: Pretraining results on the nanochat-style backbone. *Memory* shape uses the notation in Table 7; *Param (M/T)* is memory-only / total parameters.

Run	Memory	Train bpb ↓	Val bpb ↓	CORE ↑	Param (M/T)	Train Throughput
Base	–	$0.8783_{\pm 0.0009}$	$0.8785_{\pm 0.0010}$	$0.1447_{\pm 0.0064}$	0M / 135M	1.15M tok/s
Bigram	$N_{\text{hash}} \times d_{\text{model}}$	$0.8632_{\pm 0.0009}$	$0.8636_{\pm 0.0009}$	$0.1533_{\pm 0.0116}$	151M / 286M	1.09M tok/s
VEmbedding	$V \times 6 \times d_{\text{value}}$	$0.8631_{\pm 0.0006}$	$0.8633_{\pm 0.0006}$	$0.1522_{\pm 0.0045}$	151M / 286M	1.10M tok/s
151M memory						
MoME (c_{grp}=1)	$V \times 6 \times d_{\text{value}}$	$0.8618_{\pm 0.0002}$	$0.8621_{\pm 0.0003}$	$0.1571_{\pm 0.0030}$	151M / 286M	1.07M tok/s
MoME (c_{grp}=2)	$\frac{V}{2} \times 12 \times d_{\text{value}}$	$\underline{0.8609_{\pm 0.0003}}$	$\underline{0.8611_{\pm 0.0003}}$	$\mathbf{0.1583_{\pm 0.0020}}$	151M / 287M	1.07M tok/s
MoME (c_{grp}=4)	$\frac{V}{4} \times 24 \times d_{\text{value}}$	$\mathbf{0.8608_{\pm 0.0004}}$	$\mathbf{0.8610_{\pm 0.0004}}$	$0.1554_{\pm 0.0029}$	151M / 287M	1.07M tok/s
302M memory						
MoME (c_{grp}=1)	$V \times 12 \times d_{\text{value}}$	$0.8567_{\pm 0.0002}$	$0.8569_{\pm 0.0002}$	$\underline{0.1635_{\pm 0.0067}}$	302M / 438M	1.03M tok/s
MoME (c_{grp}=2)	$\frac{V}{2} \times 24 \times d_{\text{value}}$	$\mathbf{0.8558_{\pm 0.0003}}$	$\mathbf{0.8561_{\pm 0.0003}}$	$\mathbf{0.1664_{\pm 0.0059}}$	302M / 438M	1.03M tok/s
MoME (c_{grp}=4)	$\frac{V}{4} \times 48 \times d_{\text{value}}$	$\underline{0.8563_{\pm 0.0005}}$	$\underline{0.8565_{\pm 0.0006}}$	$0.1515_{\pm 0.0110}$	302M / 439M	1.02M tok/s

Multi-Head Memory. As a parameter-efficient design choice, all value heads in a memory-augmented layer share a single memory table $\mathbf{E}^{\text{mem}}$, capping the per-layer memory parameter count at NMd_{value} rather than $NMHd_{\text{value}}$. To retain head-specific aggregation flexibility under this shared table, the slot router/gate output is per-head: each value head independently selects its activated slot subset and mixture weights over the shared row, so different heads can disagree on which slots to read while drawing from the same stored bank.

4 Experiments and results

4.1 Evaluation Setting

Evaluation setting. We follow the nanochat evaluation scripts [29][2] and report final train bpb, validation bpb, raw benchmark accuracies, and the CORE metric [34]. We compute $\text{bpb} = \frac{\sum_i -\log p_\theta(y_i|x_{<i})}{\log(2)\sum_i \text{bytes}(y_i)}$ over counted target tokens, and $\text{CORE} = \frac{1}{J}\sum_{j=1}^{J} \frac{a_j - r_j}{1 - r_j}$ over task accuracies a_j with task-specific random baselines r_j. Lower bpb is better and higher CORE is better; Appendix A.2 gives the detailed evaluation suite, and Appendix A.3 gives the shared training setup. For compactness, the main tables report representative raw accuracies and the CORE aggregate; see Appendix A.2 for the full evaluation suite and per-task definitions, with the complete 22-task Llama/MobileLLM-family and Qwen3-style breakdowns in Appendix Table 15.

4.2 Experiment Details

Training settings. For the experiments in Tables 2 and 3, we train on FineWeb-Edu data [37] with 524,288-token batches. The compound-scaling runs in Table 4 use the same data and batch size. Our 100B-token runs in Table 6 and the d24 control experiments in the appendix use ClimbMix [13] with 1,048,576-token batches. All use 2048-token sequences and the shared 32k BPE tokenizer. Training uses Muon for matrix-shaped transformer parameters and AdamW parameter groups for embeddings, unembeddings, scalars, value-memory tables, and other non-matrix parameters. Appendix A.3 summarizes the concrete training setup used by the reported families.

Base architectures. We evaluate three architecture families: a nanochat-style GPT family [29], a Llama/MobileLLM-family setting [18, 36] used for the STEM comparison [54], and a Qwen3-style 0.6B family [63]. The nanochat family is a compact decoder-only Transformer setting used for controlled iso-FLOP and iso-parameter comparisons, while the Llama/MobileLLM and Qwen3 families test whether the same memory mechanism transfers to stronger sub-billion-parameter backbone recipes. All three families use the same 32k-token tokenizer trained on FineWeb-Edu with 20B tokens using byte-level BPE [49].

Baseline implementations. We compare with Bigram, the best-performing Engram [6] variant tested at our scale, using the modded-nanogpt 2-gram implementation [9]. In our controlled comparison, it achieves lower validation bpb and higher CORE-22 than our canonical Engram adaptation (Appendix A.6). For STEM, we follow the original STEM paper and its released code [54]; we use

[2] Our nanochat reference point is commit `348fbb3`.

Table 3: Iso-training-token results for the Llama/MobileLLM and Qwen3 backbone families. Rows are grouped by backbone scale. *Mem* is the memory-table size, per-benchmark columns report raw accuracy (%), *BB-avg* is the unweighted mean of the BigBench tasks in our suite, and *Wall* is training time relative to the dense baseline within the same block. Full 22-task results appear in Table 15.

Method	Mem	Val bpb ↓	CORE ↑	ARC-E	ARC-C	BoolQ	PIQA	HSwag	OBQA	WinoG	BB-avg	Wall
Llama/MobileLLM 125M												
Base	0M	0.8852	0.1382	52.82	27.56	55.14	63.82	33.27	33.40	**53.04**	20.13	1×
STEM	755M	0.8633	0.1483	55.72	**29.35**	54.01	65.13	35.47	**34.20**	52.88	20.13	1×
VEmbedding	95M	0.8620	0.1530	56.82	28.50	53.67	**66.43**	**36.23**	31.20	52.09	22.21	1.03×
MoME-A1/3	95M	0.8623	0.1587	56.44	27.30	50.49	66.00	35.59	34.00	52.49	**23.31**	1.07×
MoME-A2/6	189M	**0.8560**	**0.1686**	**57.24**	27.65	**57.86**	65.18	**36.19**	32.00	51.93	21.98	1.07×
Llama/MobileLLM 350M												
Base	0M	0.7750	0.1988	64.23	34.04	**55.66**	70.13	47.87	**37.20**	54.14	16.68	1×
STEM	1342M	0.7697	0.2179	64.52	35.41	50.52	69.37	47.64	36.80	54.54	**25.94**	1.09×
VEmbedding	168M	0.7651	0.2214	**66.29**	36.26	47.06	70.62	**49.60**	37.00	56.35	23.87	1.04×
MoME-A2/5	168M	0.7647	0.2202	65.03	36.60	44.65	70.73	49.20	35.80	56.04	24.70	1.04×
MoME-A2/10	336M	**0.7610**	**0.2286**	65.28	**36.77**	55.44	**71.60**	**49.50**	36.00	**56.75**	21.15	1.08×
Qwen3 0.6B												
Base	0M	0.7594	0.2491	65.99	35.32	59.88	70.78	50.58	37.60	55.96	25.72	1×
STEM	1409M	0.7564	0.2556	**67.97**	36.52	56.39	**71.98**	51.01	39.00	56.59	26.01	1.07×
VEmbedding	470M	**0.7427**	0.2530	66.50	38.05	47.55	71.00	53.07	39.00	**57.14**	**26.82**	1.06×
MoME-A2/8	470M	0.7461	**0.2737**	67.09	**38.31**	**63.39**	71.76	**53.39**	**39.60**	56.04	26.19	1.06×

the same Llama/MobileLLM-family backbone for fair comparison and adopt the 1/2 STEM setting reported in their paper to align more closely with our setup. Value embedding follows the value-embedding line from modded-nanogpt/nanochat practice [29, 30] and the value residual learning formulation [68]. Appendix A.6 gives baseline implementation notes.

Implementation details for our method. For our method, we default to a single memory block per layer and alternate memory-augmented blocks and transformer blocks. Each memory block contains a memory table of size $N \times M \times d_{\text{value}}$, where by default we set $M = H$ for an iso-parameter setting against value embedding. We denote configurations as MoME-AK/M, where the prefix *A* stands for *Activated*: K activated slots are selected out of M slots per row (*e.g.*, MoME-A2/5 activates K=2 slots out of M=5). Unless otherwise specified, the default activated-slot count is K=2. By default we do *not* use the kNN row-grouping function, fixing $f(x_t)$=id(x_t) and c_{grp}=1 to isolate the effect of the memory mixture itself; the only place we sweep c_{grp} is the nanochat ablation in Section 4.3, where the reference embedding table for kNN grouping is taken from a lightly trained nanochat-d12 base run (∼1B tokens, ∼1e18 FLOPs). Appendix A.5 lists the run-specific MoME configurations across backbone families.

4.3 Pretraining Results

Iso-FLOP results on nanochat architecture. We compare value embedding, Bigram, and MoME on the 12-layer 135M nanochat-style backbone at a fixed 3×10^{18} training FLOPs (≈3.3B tokens), inserting one memory block at every odd layer and matching all augmented variants to an iso-151M memory-parameter budget via N_{hash}=$6V$ for Bigram (Table 2). First, at the iso-parameter budget MoME improves both validation bpb and CORE over Bigram and VEmbedding; doubling the memory to 302M further lowers validation bpb for all three grouping factors, while CORE improves for c_{grp}=1 and 2. Second, sweeping the row-grouping factor $c_{\text{grp}} \in \{1, 2, 4\}$ at fixed parameter count, c_{grp}=2 gives the highest CORE at both budgets and the lowest train and validation bpb at 302M; at 151M, c_{grp}=4 gives slightly lower train and validation bpb. Third, the mixture-of-memory routing adds negligible runtime overhead: MoME stays within $<2\%$ of the iso-parameter Bigram throughput.

Iso-token results on Llama/MobileLLM and Qwen3 architectures. We compare methods at a fixed token budget within each backbone scale: 5B tokens for Llama3-style MobileLLM 125M, and 20B tokens for MobileLLM 350M [18, 36] and Qwen3 0.6B [63]. Memory-parameter budgets vary

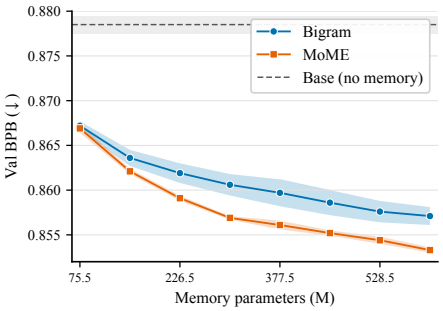


Figure 2: Scaling performance: Bigram vs. MoME. Colored regions denote error bars.

$N \times M$	Mem	Train bpb	Val bpb
Bigram	151	0.8632	0.8636
$6V \times 6$	151	0.8618	0.8624
$12V \times 6$	302	**0.8576**	**0.8582**
$6V \times 12$	302	0.8582	0.8588
$12V \times 12$	604	**0.8540**	**0.8546**
$24V \times 6$	604	0.8541	0.8547
$6V \times 24$	604	0.8544	0.8550

Table 4: Compound scaling of Bigram table size N with MoME slot count M (K=2).

Size	Value Inject	Hidden State Inject
Llama/MobileLLM		
350M	7.82%	8.05%
1B	5.47%	5.37%
Qwen3		
0.6B	7.90%	6.44%
4B	2.35%	4.60%
8B	1.56%	2.23%

Table 5: Inference-time increase relative to the Base model.

by method and configuration. Following the original STEM 1/2 setting [54], the memory module is inserted in half of the transformer layers; full per-family architecture details are in Appendix A.4. Table 3 supports three conclusions. First, the MoME gain over the no-memory base is stable and transfers across both families and across the 125M, 350M, and 0.6B scales. Second, MoME improves both validation bpb and CORE over STEM across all three scales while using fewer memory parameters. Third, MoME requires 1.04–1.08× the training wall time of the no-memory base.

Memory-size scaling. We further investigate the scaling trend of MoME compared with Bigram. We fix the backbone to nanochat d12 and scale the memory parameters under an iso-FLOP setting. As shown in Figure 2, MoME and Bigram start from similar performance at small memory sizes. MoME achieves lower validation bpb than the matched-memory Bigram row at every tested memory size. Within the tested range, scaling MoME therefore gives a consistently better bpb trajectory than scaling Bigram. We also observe that MoME has substantially lower run-to-run variance in both training and validation bpb curves (visible as narrower error bands in Figure 2 and per-budget standard deviations in Appendix Table 17), suggesting that MoME is more stable to train than Bigram in this setting.

Compound scaling with Bigram. Because MoME does not make assumptions about the first-stage token-based indexer $f(x_t)$, it is naturally complementary to methods such as Bigram. We therefore ask whether the two mechanisms can be combined. To test this, we use the baseline's bigram indexer as the first-stage token-based indexer while keeping the overall memory design of MoME. We also slightly adjust the configuration so that all memory-augmented layers share the same memory table, enabling an iso-parameter comparison with the Bigram baseline. As shown in Table 4, combining MoME with Bigram improves over Bigram under the same parameter budget. Moreover, at larger budgets, the best-performing variant comes from scaling both the Bigram table and the MoME memory slots. This suggests that the two mechanisms are compatible and can be scaled together for further performance gains.

Inference latency. Although we report training FLOPs and end-to-end wall time to evaluate the efficiency of MoME, we further probe the inference-latency overhead introduced by the memory mixture as the backbone scales. As shown in Table 5, we measure per-step inference latency for a memory-augmented transformer block and report the percentage increase relative to the Base model. We find that the routing and memory-retrieval cost is small compared with the rest of the transformer computation, so the relative latency overhead decreases as the backbone becomes larger. We also ablate the injection design: in MoME, the router takes the hidden state as input and injects the retrieved memory into the value stream, making routing and retrieval a parallel path to the standard value projection. Compared with a hidden-state-to-hidden-state injection path for MoME, this design shows a clear latency advantage as the model scales.

Scaled-up training on ClimbMix. We also evaluate MoME in a scaled-up training setting. We compare MoME, Dense, and VEmbedding after training on 104.858B ClimbMix tokens [13] in Table 6. We also include the Qwen3-0.6B [63] and Llama 3.2-1B [39] base models as external references trained on substantially larger corpora. We report bits per byte (bpb) on four corpora: the held-out ClimbMix validation set is in-domain, while FineWeb-Edu [37], enwik9, and Shakespeare serve as out-of-domain evaluation sets. Within the controlled 100B-token block of Table 6, MoME obtains lower bpb than Dense and VEmbedding on all three out-of-domain evaluations and a higher CORE-22 score, while its in-domain ClimbMix bpb is slightly higher than that of VEmbedding.

Table 6: 100B-token training results with the nanochat d24 architecture and external base-model references. Network parameters exclude memory parameters; bold marks the best result within each block.

Model	Train tokens	Network (B)	Memory (B)	Score tok/s	ClimbMix ↓	FineWeb-Edu ↓	enwik9 ↓	Shakespeare ↓	CORE-22 ↑
Qwen3-0.6B	36T	0.60	0.00	43.8k	0.7629	0.8125	0.8649	1.3855	**0.3753**
Llama-3.2-1B	$\leq$9T	1.24	0.00	51.7k	**0.7186**	**0.7378**	**0.7451**	**1.2169**	0.3629
Dense	100B	0.78	0.00	–	0.6639	0.7677	0.9037	1.4993	0.3441
VEmbedding	100B	0.78	0.60	64.0k	**0.6463**	0.7549	0.8860	1.4823	0.3484
MoME-A2/12	100B	0.78	0.61	58.4k	0.6504	**0.7548**	**0.8850**	**1.4789**	**0.3687**

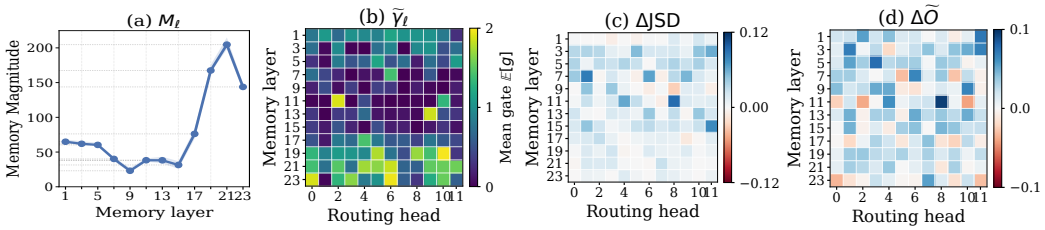


Figure 4: Model analysis. (a) CORE injected-memory norm by layer. (b) Mean CORE injection gate. (c) WiC routing-distribution effect. (d) WiC $\Delta\widetilde{O}$ effect. Blue indicates the hypothesis-consistent direction.

MoME also approaches Qwen3-0.6B on CORE-22 despite using substantially fewer pretraining tokens, and its measured full-sequence scoring throughput is in the same range as both external references.

4.4 Model Analysis

Memory usage per layer. To measure memory use across depth, we compute a task-balanced average over token positions and value heads at each memory layer ℓ with the model reported in Table 6. We summarize the per-head injection gate and the magnitude of the final memory residual added to the value head as

$$M_\ell = \mathbb{E}_{\substack{(x,t)\sim\mathrm{CORE} \\ i\in[H]}}\left[\left\|\gamma_{\ell,t,i}\tilde{\mathbf{m}}_{\ell,t,i}\right\|_2\right], \qquad \widetilde{\gamma}_\ell = \mathbb{E}_{\substack{(x,t)\sim\mathrm{CORE} \\ i\in[H]}}[\gamma_{\ell,t,i}], \tag{1}$$

where $\gamma_{\ell,t,i}\tilde{\mathbf{m}}_{\ell,t,i}$ is the gated memory vector ultimately infused into value head i. Panel (a) of Figure 4 plots M_ℓ, while panel (b) plots $\widetilde{\gamma}_\ell$. Both measurements show the same depth profile: memory use is strongest near the beginning and especially the end of the model, with substantially lower utilization through the middle layers. This pattern is consistent with memory contributing to early contextualization and then being re-engaged for late task-specific refinement, while the middle layers rely more heavily on the transformed backbone representations. Appendix A.9.4 provides the per-head and per-domain measurements.

Semantic interpretability in MoME. One initial motivation for MoME is to capture contextual variation in token semantics. This semantic structure does appear in the learned routing. We probe the learned router on a set of polysemous tokens and find that same-sense prompts tend to route to the same memory slot. Figure 3 visualizes this behavior for three polysemous tokens (*bank*, *bug*, and *drive*) at a single Qwen-0.6B head. Same-sense prompts route through the same memory slot, while the changed-sense prompt jumps to a different slot. More results are provided in Appendix A.11.

Quantitative analysis of semantic routing. Going beyond qualitative analysis, we quantify sense-sensitive routing using the Word-in-Context (WiC) dataset [48], which pairs two natural sentences containing the same target word and labels whether its sense is the same or different. We evaluate 670 strictly filtered pairs at all 144 memory layer–head sites in the d24 checkpoint. We use two complementary metrics to test a simple expectation: if routing reflects word sense, it should change more between different-sense pairs than between same-sense pairs. Jensen–Shannon divergence (JSD) compares the full routing distributions, capturing differences in weights even when the selected slots stay the same. The chance-corrected top-2 decision overlap $\widetilde{O}$ of Olson et al. [44] measures how much the two occurrences select the same slots. Using both checks whether the pattern holds for both routing weights and discrete slot choices. At each site, we report the mean JSD for different-sense

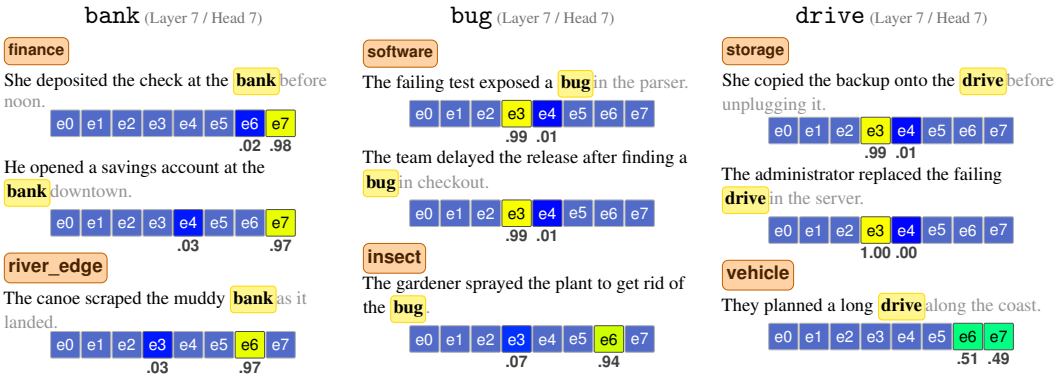


Figure 3: Semantic interpretability in MoME on a Qwen backbone (Layer 7 / Head 7). Highlighted slots are activated memory slots, with the number below each slot showing its mixture weight $\alpha_{t,a}^{(i)}$. Same-sense prompts route through the same memory slot, while the changed-sense prompt jumps to a different slot; text after the target word is greyed because routing only sees the prefix. In all three cases, n-gram-based routing has too short a window to disambiguate the senses, while MoME can condition on the full prefix.

pairs minus that for same-sense pairs (ΔJSD), and the mean overlap with the order reversed ($\Delta\widetilde{O}$). Positive values in either measure therefore indicate more similar routing for the same sense than for different senses. Panels (c)–(d) of Figure 4 show positive ΔJSD at 117 of 144 sites and positive $\Delta\widetilde{O}$ at 110 of 144 sites. Thus, most memory-routing heads exhibit similar routing for the same semantic context and different routing when the context changes the sense. Appendix A.12 provides the full filtering protocol, metric definitions, per-head statistics, gate-aware sensitivity analysis, and limitations.

5 Discussion

Limitations. The main limitation of this work is computational scale. Although the extended 100B-token experiment reaches approximately 1.4B total parameters, the underlying dense backbone has approximately 0.8B parameters and this scale check uses a single seed. The CORE activation and WiC analyses provide descriptive evidence of selective memory use and sense-sensitive routing, but causal interventions are needed to establish whether these behaviors improve downstream accuracy or robustness under domain shift.

Future work. Future work should investigate better compound-scaling strategies that combine efficient token routing with the mixture-of-memory mechanism. A second direction is suggested by the semantic structure in the learned routing: whether controlling memory routes can provide a practical handle for steering model behavior.

Conclusion. We presented MoME, a conditional memory mechanism that keeps efficient token-indexed access while allowing context-dependent selection among multiple memory slots. Across controlled pretraining experiments on nanochat, Llama/MobileLLM, and Qwen3-style backbones, MoME improves over strong memory baselines and shows favorable scaling and latency behavior in the tested regimes. These results suggest that context-aware memory mixtures are a practical direction for expanding sparse model capacity.

Acknowledgments and Disclosure of Funding

This work was funded, in part, by the Vector Institute for AI, Canada CIFAR AI Chair, NSERC Canada Research Chair (CRC), and NSERC Discovery Grant. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, the Digital Research Alliance of Canada, companies[3] sponsoring the Vector Institute, and Advanced Research Computing at the University of British Columbia. Additional hardware support was provided by John R. Evans Leaders Fund CFI grant and Digital Research Alliance of Canada under the Resource Allocation Competition award. The authors have no competing interests.

References

[1] Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. In *International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2412.09764.

[2] BIG-bench authors. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. URL https://openreview.net/forum?id=uyTL5Bvosj.

[3] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7432–7439. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6239.

[4] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International Conference on Machine Learning*, 2022. URL https://arxiv.org/abs/2112.04426.

[5] William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelly. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems 37*, 2024. URL https://openreview.net/forum?id=M2UzLRoqic.

[6] Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Yukun Li, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026.

[7] Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. 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)*, pages 2924–2936, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

[8] 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. URL https://arxiv.org/abs/1803.05457.

[9] ClassicLarry. KellerJordan/modded-nanogpt pull request 201, 2026. URL https://github.com/KellerJordan/modded-nanogpt/pull/201. Bigram hash embedding pull request.

[10] Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022. URL https://arxiv.org/abs/2104.08696.

[11] Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, et al. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual*

[3] https://vectorinstitute.ai/#partners

Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.acl-long.70/.

[12] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024. URL https://arxiv.org/abs/2412.19437.

[13] 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. Nemotron-CLIMB: Clustering-based iterative data mixture bootstrapping for language model pre-training, 2025.

[14] Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. GLaM: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, 2022. URL https://arxiv.org/abs/2112.06905.

[15] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23 (120), 2022. URL https://arxiv.org/abs/2101.03961.

[16] Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[17] 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*, 2021. URL https://arxiv.org/abs/2012.14913.

[18] Aaron Grattafiori et al. The Llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[19] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. In *International Conference on Machine Learning*, 2020. URL https://arxiv.org/abs/2002.08909.

[20] Xu Owen He. Mixture of a million experts. *arXiv preprint arXiv:2407.04153*, 2024. URL https://arxiv.org/abs/2407.04153.

[21] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems 35*, 2022. URL https://arxiv.org/abs/2203.15556.

[22] Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Xun Zhou. Over-tokenized transformer: Vocabulary is generally worth scaling. In *Proceedings of the 42nd International Conference on Machine Learning*. PMLR, 2025. URL https://proceedings.mlr.press/v267/huang25bb.html.

[23] W. Ronny Huang, Tara N. Sainath, Cal Peyser, Shankar Kumar, David Rybach, and Trevor Strohman. Lookup-table recurrent language models for long tail speech recognition. In *Interspeech 2021*, 2021. doi: 10.21437/Interspeech.2021-340. URL https://www.isca-archive.org/interspeech_2021/huang21f_interspeech.html.

[24] Zihao Huang, Qiyang Min, Hongzhi Huang, Yutao Zeng, Defa Zhu, Ran Guo, and Xun Zhou. Ultra-sparse memory network. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2411.12364.

[25] 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. URL https://arxiv.org/abs/2401.04088.

[26] Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the NanoGPT baseline, 2024. URL https://github.com/KellerJordan/modded-nanogpt.

[27] Kaggle. 200,000+ Jeopardy! Questions. https://www.kaggle.com/datasets/tunguz/200000-jeopardy-questions, 2019.

[28] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020. URL https://arxiv.org/abs/2001.08361.

[29] Andrej Karpathy. nanochat: The best ChatGPT that $100 can buy, 2025. URL https://github.com/karpathy/nanochat.

[30] KoszarskyB. X post on value embeddings. X post, 2024. URL https://x.com/KoszarskyB/status/1864746625572257852.

[31] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1907.05242.

[32] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qrwe7XHTmYb.

[33] Hector Levesque, Ernest Davis, and Leora Morgenstern. The Winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012. URL https://aaai.org/papers/59-4492-the-winograd-schema-challenge.

[34] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, et al. DataComp-LM: In search of the next generation of training sets for language models. *Advances in Neural Information Processing Systems*, 37: 14200–14282, 2024.

[35] Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. In *First Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2401.17377.

[36] Zechun Liu, Changsheng Zhao, Forrest N. Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. MobileLLM: Optimizing sub-billion parameter language models for on-device use cases. In *Proceedings of the 41st International Conference on Machine Learning*. PMLR, 2024. URL https://arxiv.org/abs/2402.14905.

[37] Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. FineWeb-Edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

[38] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2202.05262.

[39] Meta. Llama 3.2 Model Card, 2024. URL https://github.com/meta-llama/llama-models/blob/main/models/llama3_2/MODEL_CARD.md.

[40] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2381–2391, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1260. URL https://aclanthology.org/D18-1260.

[41] Yongyu Mu, Yuzhang Wu, Yuchun Fan, Chenglong Wang, Hengyu Li, Qiaozhi He, Murun Yang, Tong Xiao, and Jingbo Zhu. Cross-layer attention sharing for large language models. *arXiv preprint arXiv:2408.01890*, 2024. URL https://arxiv.org/abs/2408.01890.

[42] Tam Nguyen, Tan Nguyen, and Richard Baraniuk. Mitigating over-smoothing in transformers via regularized nonlocal functionals. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2312.00751.

[43] Cicero Nogueira dos Santos, James Lee-Thorp, Isaac Noble, Chung-Ching Chang, and David Uthus. Memory augmented language models through mixture of word experts. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2024. URL https://arxiv.org/abs/2311.10768.

[44] Matthew Lyle Olson, Neale Ratzlaff, Musashi Hinck, Man Luo, Sungduk Yu, Chendi Xue, and Vasudev Lal. Probing semantic routing in large mixture-of-expert models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 18263–18278, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.991. URL https://aclanthology.org/2025.findings-emnlp.991/.

[45] Matteo Pagliardini, Amirkeivan Mohtashami, Francois Fleuret, and Martin Jaggi. DenseFormer: Enhancing information flow in transformers via depth weighted averaging. In *Advances in Neural Information Processing Systems 37*, 2024. URL https://openreview.net/forum?id=kMnoh7CXrq.

[46] Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E. Weston, Luke Zettlemoyer, et al. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.453/.

[47] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144.

[48] Mohammad Taher Pilehvar and Jose Camacho-Collados. WiC: the word-in-context dataset for evaluating context-sensitive meaning representations. 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)*, pages 1267–1273, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1128. URL https://aclanthology.org/N19-1128/.

[49] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. OpenAI technical report, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[50] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

[51] Siva Reddy, Danqi Chen, and Christopher D. Manning. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://aclanthology.org/Q19-1016.

[52] Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S. Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, 2011. URL https://people.ict.usc.edu/~gordon/copa.html.

[53] Aurko Roy, Rohan Anil, Guangda Lai, Benjamin Lee, Jeffrey Zhao, Shuyuan Zhang, Shibo Wang, Ye Zhang, Shen Wu, Rigel Swavely, et al. N-Grammer: Augmenting transformers with latent n-grams. *arXiv preprint arXiv:2207.06366*, 2022. URL https://arxiv.org/abs/2207.06366.

[54] Ranajoy Sadhukhan, Sheng Cao, Harry Dong, Changsheng Zhao, Attiano Purpura-Pontoniere, Yuandong Tian, Zechun Liu, and Beidi Chen. STEM: Scaling transformers with embedding modules. *arXiv preprint arXiv:2601.10639*, 2026.

[55] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8732–8740. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/6399.

[56] Noam Shazeer. GLU variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020. URL https://arxiv.org/abs/2002.05202.

[57] 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. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

[58] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *Advances in Neural Information Processing Systems*, 2015. URL https://arxiv.org/abs/1503.08895.

[59] Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. 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)*, pages 4149–4158, Minneapolis, Minnesota, 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421.

[60] Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2407.13623.

[61] Karan Uppal, Nagarajan Natarajan, and Manik Varma. MoVE: Mixture-of-vocabulary-experts for improved representation learning, 2026. URL https://openreview.net/forum?id=xEgjOxM5dZ. Submitted to the International Conference on Learning Representations.

[62] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015. URL http://arxiv.org/abs/1410.3916.

[63] An Yang et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[64] Yebin Yang, Huaijin Wu, Fu Guo, Lin Yao, Xiaohan Qin, Jingzhi Wang, Debing Zhang, and Junchi Yan. JTok: On token embedding as another axis of scaling law via joint token self-modulation. *arXiv preprint arXiv:2602.00800*, 2026. doi: 10.48550/arXiv.2602.00800. URL https://arxiv.org/abs/2602.00800.

[65] Da Yu, Edith Cohen, Badih Ghazi, Yangsibo Huang, Pritish Kamath, Ravi Kumar, Daogao Liu, and Chiyuan Zhang. Scaling embedding layers in language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=gH4BRa4ZP3.

[66] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL https://aclanthology.org/P19-1472.

[67] Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. AGIEval: A human-centric benchmark for evaluating foundation models. In *Findings of the Association for Computational Linguistics: NAACL 2024*. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.findings-naacl.149/.

[68] Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.acl-long.1375/.

A Appendix

A.1 Notation

Table 7: Notation used for Mixture of Memory. We reserve V for vocabulary size, N for first-stage memory rows, M for slots per row, K for activated slots, and d_{value} for the per-head value dimension.

Object	Symbol	Meaning
Index shorthand	$[q]$	The set $\{1, \ldots, q\}$.
Input token	$x_t \in \mathcal{V}$	Token at sequence position t.
Vocabulary size	$V = \lvert\mathcal{V}\rvert$	Number of tokens in the vocabulary.
Hidden state	$\mathbf{h}_t \in \mathbb{R}^{d_{\text{model}}}$	Transformer hidden state used by the slot gate and value-residual gate.
Value heads	H	Number of key/value heads at the memory-injection layer.
Value dimension	d_{value}	Per-head value dimension; $\mathbf{v}_{t,i} \in \mathbb{R}^{d_{\text{value}}}$ and $\mathbf{v}_t \in \mathbb{R}^{H d_{\text{value}}}$.
Augmented value	$\tilde{\mathbf{v}}_{t,i} \in \mathbb{R}^{d_{\text{value}}}$	Value vector after adding the gated retrieved memory for position t and value head i.
Row indexer	$f : \mathcal{V} \to [N]$, $n_t = f(x_t)$	Fixed first-stage map from token id to memory row.
Slot gate	$g_\theta^{(i)}(\mathbf{h}_t)$	Second-stage context indexer returning the activated slot set $\mathcal{A}_{t,i}$ and mixture weights $\alpha_{t,a}^{(i)}$ for head i.
Grouping factor	c_{grp}	Target token grouping/compression factor for the kNN slot map; this is distinct from the active-slot count K.
Memory bank	$\mathbf{E}^{\text{mem}} \in \mathbb{R}^{N \times M \times d_{\text{value}}}$	Learnable memory tensor with M candidate slots per row, shared across value heads.
Stored memory vector	$\mathbf{m}_{n,a} = \mathbf{E}^{\text{mem}}[n, a, :]$	Memory vector stored at row n and slot a, with $\mathbf{m}_{n,a} \in \mathbb{R}^{d_{\text{value}}}$.
Gate logits	$\boldsymbol{\ell}_{t,i} \in \mathbb{R}^M$	Pre-sigmoid slot logits for position t and value head i.
Gate scores	$\mathbf{s}_{t,i} = \sigma(\boldsymbol{\ell}_{t,i})$	Post-sigmoid slot scores used for top-K selection.
Activated slots	$\mathcal{A}_{t,i} \subseteq [M]$	Top-K memory slots activated for position t and value head i.
Mixture weights	$\alpha_{t,a}^{(i)}$	Sigmoid-norm weight for slot a after normalization over $\mathcal{A}_{t,i}$.
Retrieved memory	$\tilde{\mathbf{m}}_{t,i} \in \mathbb{R}^{d_{\text{value}}}$	Per-head memory vector added to the value stream after scaling by $\gamma_{t,i}$.
Value-residual gate	$\boldsymbol{\gamma}_t \in \mathbb{R}^H$	Per-head scalar gate produced from $\mathbf{h}_t$ to control memory injection strength.

A.2 Evaluation Protocol

Table 8 lists the task suite used by the local nanochat CORE evaluator. The evaluator shuffles each task with a fixed seed, evaluates raw accuracy, reads the random baseline for each task from the evaluation metadata, and reports the unweighted mean of centered accuracies as CORE. For bpb, the evaluator sums target negative log-likelihood over counted tokens, masks ignored and special-token targets, converts nats to bits, and normalizes by the UTF-8 byte count of the same targets.

Table 8: The 22-task CORE evaluation suite used by the local nanochat evaluation bundle. The evaluator reports raw task accuracy and then averages task-specific random-baseline-centered accuracies for the CORE metric.

Task	Shots	Evaluation type	Source
HellaSwag-0	0	multiple choice	[66]
Jeopardy	10	language modeling	[27]
BB-QA Wikidata	10	language modeling	[2]
ARC-Easy	10	multiple choice	[8]
ARC-Challenge	10	multiple choice	[8]
COPA	0	multiple choice	[52]
CommonsenseQA	10	multiple choice	[59]
PIQA	10	multiple choice	[3]
OpenBookQA	0	multiple choice	[40]
LAMBADA OpenAI	0	language modeling	[47]
HellaSwag	10	multiple choice	[66]
Winograd	0	schema	[33]
WinoGrande	0	schema	[55]
BB-Dyck Languages	10	language modeling	[2]
AGI Eval LSAT-AR	3	multiple choice	[67]
BB-CS Algorithms	10	language modeling	[2]
BB-Operators	10	language modeling	[2]
BB-Repeat Copy Logic	10	language modeling	[2]
SQuAD	10	language modeling	[50]
CoQA	0	language modeling	[51]
BoolQ	10	multiple choice	[7]
BB-Language Identification	10	multiple choice	[2]

A.3 Training Setup

Table 9 summarizes the training settings. Tables 10 and 11 give the backbone and memory configurations for each family.

Table 9: Training setup for the experiments in Section 4.3. Architecture, memory configuration, device batch size, and training budget vary by comparison.

Item	Setting
Data	The experiments in Tables 2–4 use FineWeb-Edu [37]. Our 100B-token runs in Table 6 and the d24 appendix controls use ClimbMix [13].
Tokenizer	Shared byte-level BPE tokenizer [49] with vocabulary size $32{,}768$, trained on the FineWeb-Edu training stream; special tokens have zero byte count for bpb.
Context length	2048
Total batch	Tables 2–4 use $524{,}288$ tokens; our 100B-token runs in Table 6 and the d24 appendix controls use $1{,}048{,}576$ tokens. Gradient accumulation preserves the total batch size as device batch size and world size vary.
Optimizer groups	Muon is used for matrix-shaped transformer parameters; AdamW groups are used for embeddings, unembeddings, scalar parameters, value-memory tables, and other non-matrix parameters. Value-memory tables may use a separate AdamW learning rate and weight decay, listed in Table 11.
Learning-rate schedule	Run launchers use a warmup followed by cosine decay or warmdown. The nanochat scaling runs use target-FLOP stopping; token-matched Llama/MobileLLM and Qwen3-style runs use fixed token budgets.
Evaluation tokens	BPB evaluation uses the run-specified token budget: $20{,}971{,}520$ tokens per split for nanochat/scaling reports and $2{,}097{,}152$ tokens per split for token-matched Llama/MobileLLM and Qwen3 reports. CORE uses all examples unless a run explicitly sets a per-task cap.
Seed policy	Nanochat scaling tables report three-seed means over seeds 42, 43, 44 unless otherwise noted. Token-matched family tables use the completed run(s) available for each configuration.
Distributed training	Runs use DDP. The local reports use world size 4 for most nanochat and Llama/MobileLLM runs and world size 8 for Qwen3-style and larger nanochat d24 runs.
Hardware	Training hardware varies by run, including NVIDIA RTX PRO 6000 Blackwell (96GB) and H100 (80GB) GPUs. The d24 control reruns use four H100 GPUs; the sparse-MoE throughput comparison in Table 13 uses eight H100 GPUs.

A.4 Backbone Family Details

Table 10 lists the backbone configurations for the nanochat, Llama/MobileLLM, and Qwen3-style experiments.

Table 10: Backbone configurations for the reported experiments. Asterisks mark dimensions derived from the configured aspect ratio and head dimension.

Family	Scale	Model type	Depth	d_{model}	Head dim	Attn. heads	KV heads	Tie emb.	Ctx.	Training budget
nanochat	d12	`gpt`	12	768*	128	6*	6*	No	2048	3×10^{18} FLOPs; 3.3B tokens
nanochat	d24	`gpt`	24	1536*	128	12*	12*	No	2048	6×10^{19} FLOPs; 12B tokens
Llama/MobileLLM	125M	`stemgpt_350m`	30	576	64	9*	3	Yes	2048	5B tokens
Llama/MobileLLM	350M	`stemgpt_350m`	32	960	64	15*	5	No	2048	20B tokens
Qwen3-style	0.6B	`qwen3_0p5b`	28	1024	128	16	8	Yes	2048	20B tokens

A.5 Memory Module Configurations Across Families

Table 11 specifies the memory layers, row indexer, router input, gate function, and value-table optimizer for each backbone family.

Table 11: Memory configurations by backbone family. AK/M means that the router selects K of M slots per row. "Launcher-default wd" uses the default weight decay for the value-memory table in that run.

nanochat d12. Memory layers: memory blocks with span 1. Shape: $N \times M \times d_{\text{value}}$. MoME setting: A2/6, A2/12, A2/24 in main rows, with M swept in the scaling study. Row indexer f: identity or frozen kNN table, depending on row. Router input: hidden. Gate: sigmoid-norm. Value-table optimizer: AdamW group, lr 0.2, wd 0.001.

nanochat d24. Memory layers: odd layers $1, 3, \ldots, 23$. Shape: $N \times M \times d_{\text{value}}$. MoME setting: A2/12. Row indexer f: identity. Router input: hidden. Gate: sigmoid-norm. Value-table optimizer: AdamW group, lr 0.2, wd 0.001.

Llama/MobileLLM 125M. Memory layers: odd layers $1, 3, \ldots, 29$. Shape: $V \times M \times d_{\text{value}}$. MoME setting: A1/3 and A2/6 reported variants. Row indexer f: identity. Router input: value. Gate: softmax for A1/3, sigmoid-norm for A2/6. Value-table optimizer: AdamW group, lr 0.1, wd 0.001.

Llama/MobileLLM 350M. Memory layers: odd layers $1, 3, \ldots, 31$. Shape: $V \times M \times d_{\text{value}}$. MoME setting: A2/5 and A2/10. Row indexer f: identity. Router input: value. Gate: sigmoid-norm. Value-table optimizer: AdamW group, lr 0.1, launcher-default wd.

Qwen3-style 0.6B. Memory layers: odd layers $1, 3, \ldots, 27$. Shape: $V \times M \times d_{\text{value}}$. MoME setting: A2/8. Row indexer f: identity. Router input: hidden. Gate: sigmoid-norm. Value-table optimizer: AdamW group, lr 0.2, wd 0.001.

A.6 Baseline Implementation Details

Bigram. Our Bigram baseline follows modded-nanogpt practice [9], rather than the canonical Engram architecture of Cheng et al. [6]. Memory is a single table of N rows with row dimension d_{model}, indexed at each position by a deterministic hash of the previous token together with the current token (the bigram key). The same table is shared across the memory-augmented layers, which are the odd-indexed blocks in the default nanochat configuration. At each such layer, the retrieved row is added to the residual stream through hidden-state-conditioned per-head gates and a learned per-layer scalar. The lookup itself remains a deterministic function of the two token IDs. The dictionary size is written as a multiplier of the tokenizer vocabulary size V; memory-scaling sweeps vary this multiplier while keeping depth, target FLOPs, context length, batch size, and seed policy aligned with the matched MoME runs. The Mobile/Llama Bigram row uses the matched-size 350M run when it is reported.

At approximately matched total parameters and 6×10^{19} training FLOPs on d24 ClimbMix, Bigram achieves lower validation bpb and higher CORE-22 than our canonical Engram adaptation (Table 16). This single-seed comparison uses an adaptation that retains 2-/3-gram lookup and hidden-state-conditioned fusion but omits Engram's multi-stream integration for the single-stream nanochat backbone.

Value Embedding. The Value Embedding (VE) baseline follows modded-nanogpt and nanochat [29, 30] and the value-residual formulation of Zhou et al. [68]. A token-indexed table of size $V \times H \times d_{\text{value}}$ supplies one embedding slice per attention head, which is added to the attention value stream. By default, VE is injected into the odd-indexed transformer blocks; its table size is set by the vocabulary and value-head dimensions.

STEM. The STEM baseline follows the original architecture construction and released implementation [54]. We adapt the reported one-half-layer STEM setting to the same training harness, tokenizer, context length, and batch size used by the Llama/MobileLLM-family comparisons.

A.7 Compound Scaling

A.7.1 Bigram Memory and Slot Scaling

Table 12 expands Table 4 with total parameters and CORE. Each fixed-budget group compares allocations to Bigram table size, MoME slot count, or both.

Table 12: Full compound scaling table for Bigram table size N and MoME slot count M on the 12-layer nanochat backbone, averaged over three seeds. The compact main table reports only memory size and bpb; this table additionally lists total parameters and CORE. Bold and underline mark the best and second-best train/validation bpb within each fixed memory-budget group.

Bigram N	MoME setting	Mem (M)	Total params (M)	Train bpb ↓	Val bpb ↓	CORE ↑
$6V$	A2/6	151	286.4	$0.8618_{\pm 0.0010}$	$0.8624_{\pm 0.0010}$	$0.1647_{\pm 0.0023}$
$12V$	A2/6	302	437.4	$\mathbf{0.8576_{\pm 0.0008}}$	$\mathbf{0.8582_{\pm 0.0007}}$	$0.1549_{\pm 0.0084}$
$8V$	A2/9	302	437.5	$\underline{0.8579_{\pm 0.0010}}$	$\underline{0.8585_{\pm 0.0010}}$	$0.1551_{\pm 0.0088}$
$6V$	A2/12	302	437.6	$0.8582_{\pm 0.0008}$	$0.8588_{\pm 0.0008}$	$0.1544_{\pm 0.0095}$
$12V$	A2/12	604	739.6	$\mathbf{0.8540_{\pm 0.0002}}$	$\mathbf{0.8546_{\pm 0.0001}}$	$0.1604_{\pm 0.0050}$
$24V$	A2/6	604	739.4	$\underline{0.8541_{\pm 0.0011}}$	$\underline{0.8547_{\pm 0.0012}}$	$0.1590_{\pm 0.0096}$
$6V$	A2/24	604	739.9	$0.8544_{\pm 0.0012}$	$0.8550_{\pm 0.0012}$	$0.1614_{\pm 0.0108}$

A.7.2 Combination with Sparse-FFN MoE

Table 13 tests whether MoME can complement a sparse-FFN MoE. The matched-total E6/K2 MoE-FFN obtains the lowest validation bpb, while combining it with MoME gives the highest CORE-22. The combined model has a similar active-parameter count but more stored parameters than either component, and trains more slowly.

Table 13: Sparse-MoE interaction on d24/w1536/h12, ClimbMix, seed 42, at approximately 6×10^{19} training FLOPs. E6/K2 activates two of six routed FFN experts in every Transformer layer and retains one always-on shared expert; "+ MoME" additionally places MoME at every odd layer. Throughput is measured on eight H100 GPUs with the same global batch and native Flash Attention 3. The models have similar active-parameter counts, but the combined model has more stored parameters than either component. Active counts include full input/output embedding matrices, routers, other dense/shared parameters, selected FFN experts, and distinct retrieved memory entries; unselected FFN experts and unaccessed memory entries are excluded.

Model	Stored parameters (M)	Active parameters/token (M)	Training tokens/s	Validation bpb ↓	CORE-22 ↑
MoME	1,386.8	782.8	**796,258**	0.6990	0.2941
MoE-FFN, E6/K2/24	1,384.3	780.4	653,033	**0.6903**	0.2886
MoE-FFN, E6/K2/24 + MoME	1,991.0	783.0	611,249	0.6913	**0.2986**

A.8 Supplementary Quantitative Results

A.8.1 Matched 100B checkpoints and external base-model references

Table 14 reports the full 22-task accuracies for the matched 100B dense value-embedding and MoME checkpoints, alongside Qwen and Llama references up to the 3B model class. The native checkpoints form a controlled comparison; the external models differ in training data, token exposure, and tokenizer and serve as reference points.

Table 14: Raw accuracy (%) on the 22 CORE tasks for the matched 100B dense value-embedding and MoME checkpoints and external Qwen and Llama references up to the 3B model class. Appendix A.2 defines CORE aggregation.

CORE task	Dense VE	MoME	Qwen3-0.6B	Qwen3-1.7B	Llama-3.2-1B	Llama-3.2-3B
HellaSwag, zero-shot	66.36	66.75	52.13	64.86	63.59	73.43
Jeopardy	26.12	28.34	25.22	40.58	34.91	48.18
BigBench QA Wikidata	56.44	57.27	59.93	68.81	69.34	72.15
ARC-Easy	74.96	74.24	73.19	78.87	68.69	75.34
ARC-Challenge	47.27	46.50	43.69	54.44	38.14	48.04
COPA	69.00	70.00	71.00	75.00	74.00	80.00
CommonsenseQA	50.12	59.38	66.01	80.10	36.77	71.42
PIQA	78.07	78.18	71.16	76.12	75.52	79.11
OpenBookQA	45.20	43.20	34.80	41.80	38.80	43.80
LAMBADA	52.78	52.49	54.59	62.51	62.16	69.57
HellaSwag	67.38	67.97	52.45	65.55	65.16	75.49
Winograd	80.95	80.22	75.46	80.59	82.42	85.71
WinoGrande	61.80	60.46	58.25	63.77	60.22	69.53
BigBench Dyck Languages	4.40	7.80	12.20	31.50	13.70	22.40
AGI Eval LSAT-AR	24.78	25.22	24.35	26.09	23.91	24.35
BigBench CS Algorithms	37.35	43.64	47.73	73.79	46.21	62.20
BigBench Operators	18.57	23.33	60.95	73.33	40.00	58.10
BigBench Repeat Copy Logic	3.12	3.12	15.62	18.75	9.38	21.88
SQuAD	52.12	52.53	57.75	65.51	51.97	61.00
CoQA	38.13	39.30	39.45	44.91	36.34	43.76
BoolQ	62.51	69.36	74.25	81.28	64.98	74.28
BigBench Language Identification	25.42	26.31	36.58	42.71	24.77	49.00

A.8.2 Combined full token-matched results

Table 15 combines the Llama/MobileLLM and Qwen3-style rows from Table 3. It adds train bpb and reports raw accuracy for every benchmark in the evaluation suite, including all BigBench tasks as separate rows.

Table 15: Transposed full token-matched results for the runs summarized in Table 3. Columns are runs grouped by backbone family; rows report memory, relative wall time, train/validation bpb, CORE, and raw benchmark accuracy (%) for every benchmark in the evaluation suite. All MobileLLM 125M columns report seed 42 from the September 2026 reproduction. BigBench rows are QA Wikidata (QAW), Dyck Languages (Dyck), CS Algorithms (CS), Operators (Op), Repeat Copy Logic (RCL), and Language Identification (LI). Bold marks the best value and underline marks the second-best value within each family block; accuracy values within 0.1 percentage points share the same rank.

	Llama/MobileLLM 125M/5B					Llama/MobileLLM 350M/20B					Qwen3 0.6B/20B			
Metric	Base	STEM	VEmbedding	MoME-A1/3	MoME-A2/6	Base	STEM	VEmbedding	MoME-A2/5	MoME-A2/10	Base	STEM	VEmbedding	MoME-A2/8
Mem	0M	755M	94.4M	94.4M	188.7M	0M	1342.2M	167.8M	167.8M	335.5M	0M	1409.3M	469.8M	469.8M
Wall	1×	1×	1.03×	1.07×	1.07×	1×	1.09×	1.04×	1.04×	1.08×	1×	1.07×	1.06×	1.06×
Train bpb	0.8904	0.8683	0.8671	0.8673	**0.8613**	0.7807	**0.7645**	0.7710	0.7704	0.7675	0.7577	0.7484	**0.7374**	0.7381
Val bpb	0.8852	0.8633	0.8620	0.8623	**0.8560**	0.7750	0.7697	0.7651	0.7647	**0.7610**	0.7594	0.7564	**0.7427**	0.7461
CORE	0.1382	0.1483	0.1530	0.1587	**0.1686**	0.1988	0.2179	0.2214	0.2202	**0.2286**	0.2491	0.2556	0.2530	**0.2737**
HSwag-0	34.11	35.81	36.28	36.15	**36.59**	48.00	47.65	49.35	49.11	**49.79**	50.12	50.81	**52.66**	**52.66**
Jeop	0.76	2.36	0.90	0.85	**3.83**	13.79	15.45	**17.48**	16.34	16.67	18.19	22.34	**23.29**	22.82
ARC-E	52.82	55.72	56.82	56.44	**57.24**	64.23	64.52	**66.29**	65.03	65.28	65.99	**67.97**	66.50	67.09
ARC-C	27.56	**29.35**	28.50	27.30	27.65	34.04	35.41	36.26	36.60	**36.77**	35.32	36.52	38.05	**38.31**
COPA	58.00	65.00	64.00	63.00	**66.00**	65.00	66.00	67.00	68.00	**70.00**	**72.00**	69.00	71.00	68.00
CSQA	**31.61**	27.19	22.11	29.48	30.71	19.74	**22.44**	20.72	21.95	20.31	22.36	23.10	22.28	**28.99**
PIQA	63.82	65.13	**66.43**	66.00	65.18	70.13	69.37	70.62	70.73	**71.60**	70.78	**71.98**	71.00	71.76
OBQA	33.40	**34.20**	31.20	34.00	32.00	**37.20**	36.80	37.00	35.80	36.00	37.60	39.00	39.00	**39.60**
LAMB	27.58	28.68	29.54	29.38	**30.72**	39.41	38.19	39.94	41.51	**42.05**	43.16	42.03	42.98	**43.45**
HSwag	33.27	35.47	**36.23**	35.59	**36.19**	47.87	47.64	**49.60**	49.20	**49.50**	50.58	51.01	53.07	**53.39**
Winog	60.81	58.61	57.51	**62.64**	60.44	65.20	62.64	67.40	66.67	**68.50**	67.03	**71.43**	69.23	69.96
WinoG	**53.04**	52.88	52.09	52.49	51.93	54.14	54.54	56.35	56.04	**56.75**	55.96	56.59	**57.14**	56.04
LSAT	23.91	23.91	25.65	**26.09**	25.65	**27.83**	23.91	21.30	23.04	23.04	24.35	24.78	24.35	**26.09**
SQuAD	4.81	6.14	13.61	8.96	**14.31**	26.35	28.98	**31.43**	30.57	30.61	28.05	29.78	**35.87**	35.62
CoQA	10.12	12.31	13.52	14.17	**14.73**	21.48	23.19	**25.23**	24.14	23.60	24.60	25.96	28.18	**28.50**
BoolQ	55.14	54.01	53.67	50.49	**57.86**	**55.66**	50.52	47.06	44.65	55.44	59.88	56.39	47.55	**63.39**
BB-QAW	31.48	37.23	40.29	**41.36**	39.13	53.54	54.24	51.66	53.64	**54.54**	54.93	**56.58**	54.43	53.89
BB-Dyck	6.20	8.40	**12.60**	10.60	8.40	0.00	8.20	**9.70**	8.40	8.50	**16.70**	14.10	13.20	9.90
BB-CS	**43.11**	38.33	**43.03**	39.92	39.77	0.00	**43.03**	34.24	41.97	18.26	40.98	36.89	**45.45**	44.17
BB-Op	11.90	10.95	12.38	**16.19**	15.71	17.14	15.71	**21.90**	18.57	20.00	16.19	19.52	22.38	**23.81**
BB-RCL	3.12	0.00	0.00	**6.25**	3.12	3.12	**9.38**	0.00	0.00	0.00	0.00	**3.12**	0.00	0.00
BB-LI	24.96	**25.88**	24.97	25.52	25.72	**26.30**	25.07	25.70	25.64	25.62	25.50	**25.87**	25.48	25.35

A.8.3 Token-Indexed Memory Baselines

Table 16 compares MoME with token-indexed memory baselines, including adaptations of JTok and JTok-M [64] and MoVE-Input [61] to our backbone and training setup. Bigram gives the lowest validation bpb, while MoME gives the highest CORE-22.

Table 16: Token-indexed memory comparisons on d24/w1536/h12, ClimbMix, seed 42, at approximately 6×10^{19} training FLOPs. Bigram is the 2-gram baseline used in the main comparisons. Canonical Engram, JTok, JTok-M, and MoVE-Input are adapted to our backbone at approximately matched parameter counts; their architectures and training settings differ from the originals. Appendix A.6 describes the Engram adaptation.

Method	Parameters (M)	Validation bpb ↓	CORE-22 ↑
Bigram	1,384.1	**0.6943**	0.2871
Canonical Engram adaptation	1,389.6	0.7029	0.2698
JTok adaptation	1,384.1	0.6974	0.2764
JTok-M adaptation	1,384.1	0.7003	0.2756
MoVE-Input adaptation	1,386.7	0.7016	0.2744
MoME	1,386.8	0.6990	**0.2941**

A.9 Scaling and Ablations

A.9.1 Memory-size scaling

Table 17 and Figure 5 compare Bigram and MoME as memory size increases on the nanochat d12 backbone, with the no-memory Base model from Table 2 as a reference.

Table 17: Memory-size scaling sweep on the 12-layer nanochat backbone trained at 3e18 FLOPs ($\sim$3.3B tokens), with block counts $\{3, 6, 9, 12, 15, 18, 21, 24\}$. *Blocks* is the memory expansion factor; each block adds ≈ 25.2M parameters to the memory table. Values are 3-seed means $\pm$ std. The Base-model row is the no-memory baseline imported from Table 2. Bigram and MoME share the same memory budget at each block count, so the rows are directly comparable. Figure 2 plots the val bpb column vs. memory.

Method	Blocks	Mem (M)	Train BPB ↓	Val BPB ↓	CORE ↑	Wall (min)
Base	–	0	$0.8783_{\pm 0.0009}$	$0.8785_{\pm 0.0010}$	$0.1447_{\pm 0.0064}$	–
Bigram	3	75.5	$0.8668_{\pm 0.0006}$	$0.8672_{\pm 0.0005}$	$0.1513_{\pm 0.0160}$	$106.72_{\pm 0.70}$
Bigram	6	151.0	$0.8632_{\pm 0.0009}$	$0.8636_{\pm 0.0009}$	$0.1533_{\pm 0.0116}$	$108.80_{\pm 0.47}$
Bigram	9	226.5	$0.8615_{\pm 0.0012}$	$0.8619_{\pm 0.0011}$	$0.1562_{\pm 0.0080}$	$110.55_{\pm 0.50}$
Bigram	12	302.0	$0.8602_{\pm 0.0013}$	$0.8606_{\pm 0.0012}$	$0.1497_{\pm 0.0035}$	$112.49_{\pm 0.38}$
Bigram	15	377.5	$0.8592_{\pm 0.0015}$	$0.8597_{\pm 0.0015}$	$0.1551_{\pm 0.0105}$	$114.35_{\pm 0.43}$
Bigram	18	453.0	$0.8581_{\pm 0.0015}$	$0.8586_{\pm 0.0014}$	$0.1608_{\pm 0.0052}$	$116.34_{\pm 0.60}$
Bigram	21	528.5	$0.8571_{\pm 0.0013}$	$0.8576_{\pm 0.0012}$	$0.1494_{\pm 0.0091}$	$118.28_{\pm 0.40}$
Bigram	24	604.0	$0.8564_{\pm 0.0011}$	$0.8571_{\pm 0.0010}$	$0.1613_{\pm 0.0034}$	$120.17_{\pm 0.45}$
MoME	3	75.5	$0.8667_{\pm 0.0005}$	$0.8669_{\pm 0.0005}$	$0.1476_{\pm 0.0035}$	$109.77_{\pm 0.82}$
MoME	6	151.0	$0.8618_{\pm 0.0002}$	$0.8621_{\pm 0.0003}$	$0.1571_{\pm 0.0030}$	$111.89_{\pm 0.59}$
MoME	9	226.5	$0.8589_{\pm 0.0003}$	$0.8591_{\pm 0.0003}$	$0.1627_{\pm 0.0063}$	$113.48_{\pm 0.59}$
MoME	12	302.0	$0.8567_{\pm 0.0002}$	$0.8569_{\pm 0.0002}$	$0.1635_{\pm 0.0067}$	$115.68_{\pm 0.54}$
MoME	15	377.5	$0.8558_{\pm 0.0005}$	$0.8561_{\pm 0.0005}$	$0.1599_{\pm 0.0068}$	$117.59_{\pm 0.57}$
MoME	18	453.0	$0.8550_{\pm 0.0003}$	$0.8552_{\pm 0.0003}$	$0.1602_{\pm 0.0027}$	$119.76_{\pm 0.51}$
MoME	21	528.5	$0.8541_{\pm 0.0004}$	$0.8544_{\pm 0.0004}$	$0.1584_{\pm 0.0003}$	$121.85_{\pm 0.60}$
MoME	24	604.0	$0.8530_{\pm 0.0003}$	$0.8533_{\pm 0.0003}$	$0.1586_{\pm 0.0025}$	$123.70_{\pm 0.51}$

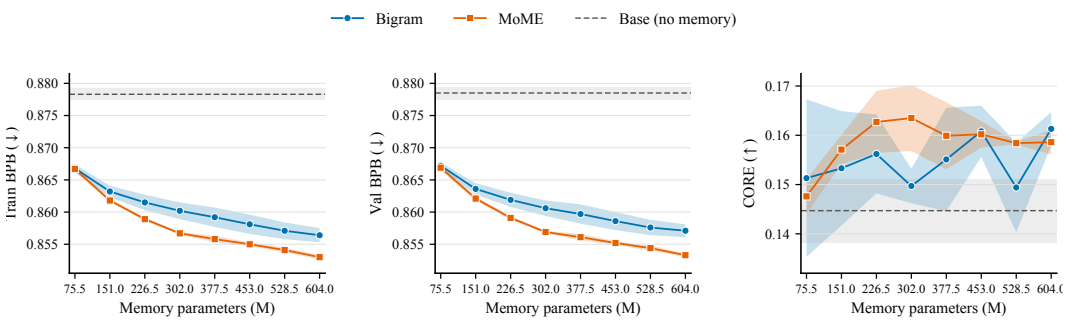


Figure 5: Full memory-size scaling sweep on the 12-layer nanochat backbone (3e18 FLOPs, 3-seed mean $\pm$ std). From left to right: training bpb, validation bpb, and CORE vs. memory parameter count for Bigram and MoME for block counts $\{3, 6, 9, 12, 15, 18, 21, 24\}$. The dashed line and shaded band are the no-memory Base model from Table 2.

A.9.2 Context-Conditioned Routing

Table 18 compares routing rules while retaining multiple trainable slots at nearly matched parameter and compute budgets. Hidden-state routing improves validation bpb and CORE-22 over learned token-only and fixed-random routing.

Table 18: Routing controls on d24/w1536/h12, ClimbMix, seed 42, at approximately 6×10^{19} training FLOPs. All variants retain multiple slots at nearly matched parameter budgets; only hidden-state routing uses context. Both delta columns report the control value minus the MoME value.

Variant	Context-conditioned	Parameters (M)	Validation bpb ↓	Δ bpb	CORE-22 ↑	Δ CORE
MoME, hidden-state routing	Yes	1,386.8	**0.6990**	—	**0.2941**	—
MoME, learned token-only routing	No	1,388.8	0.7044	+0.0054	0.2853	-0.0088
MoME, fixed-random routing	No	1,384.1	0.7018	+0.0028	0.2768	-0.0173

A.9.3 Dense Scaling Controls

Table 19 compares MoME with pure dense backbones at the original width, at a width selected to match measured inference latency, and at approximately matched total parameters. Under the same training-FLOP budget, MoME has the lowest bpb on the native validation set and all three shifted-domain evaluations and the highest CORE-22.

Table 19: Dense-scaling controls on ClimbMix, seed 42, at approximately 6×10^{19} training FLOPs. The latency-matched width is selected using measured inference latency before training; the matched-total dense model is compute-matched rather than token-matched because its larger dense backbone processes fewer tokens under the fixed FLOP budget. Validation bpb uses the native evaluator, while the three additional corpora use the standardized document-reset protocol.

Model	Parameters (M)	Validation bpb ↓	FineWeb-Edu bpb ↓	enwik9 bpb ↓	Shakespeare bpb ↓	CORE-22 ↑
Pure dense d24/w1536	780	0.7042	0.8105	0.9775	1.5610	0.2843
Latency-matched dense d24/w1728	973	0.6998	0.8097	0.9757	1.5632	0.2773
Matched-total dense d25/w2048	1,393	0.7033	0.8092	0.9780	1.5584	0.2871
MoME d24/w1536	1,387	**0.6990**	**0.8045**	**0.9712**	**1.5531**	**0.2941**

A.9.4 Test-time memory use across CORE domains

We evaluate the step-100,000 $d24$ checkpoint used in Figure 4. The model has 12 odd-numbered memory layers, 12 routing heads per memory layer, 12 slots per head, and top-2 routing. The ClimbMix reference contains 1,048,576 executed token positions; the CORE audit evaluates all examples in the 22-task bundle. For multiple-choice tasks, repeated candidate executions are counted because they are part of the model's actual test-time computation. Cross-task summaries give every task equal weight.

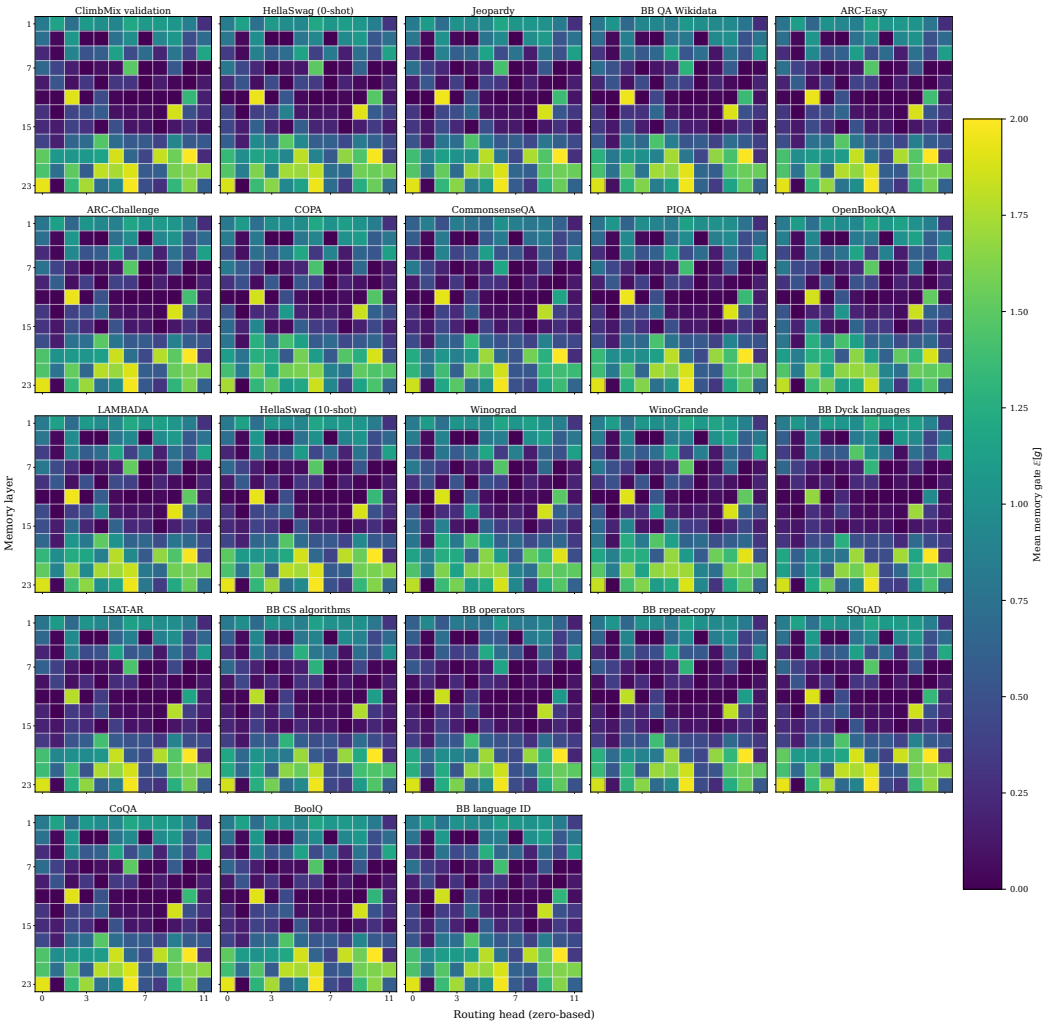


Figure 6: Mean injection gates for ClimbMix validation and all 22 CORE tasks. Each panel shows 12 memory layers by 12 routing heads on the same $[0, 2]$ color scale.

Figure 6 shows similar gate patterns across CORE tasks, with substantial variation between heads within each layer. Some heads have nearly closed injection gates, while others have gates near their upper bound; layer averages obscure these differences in injection strength.

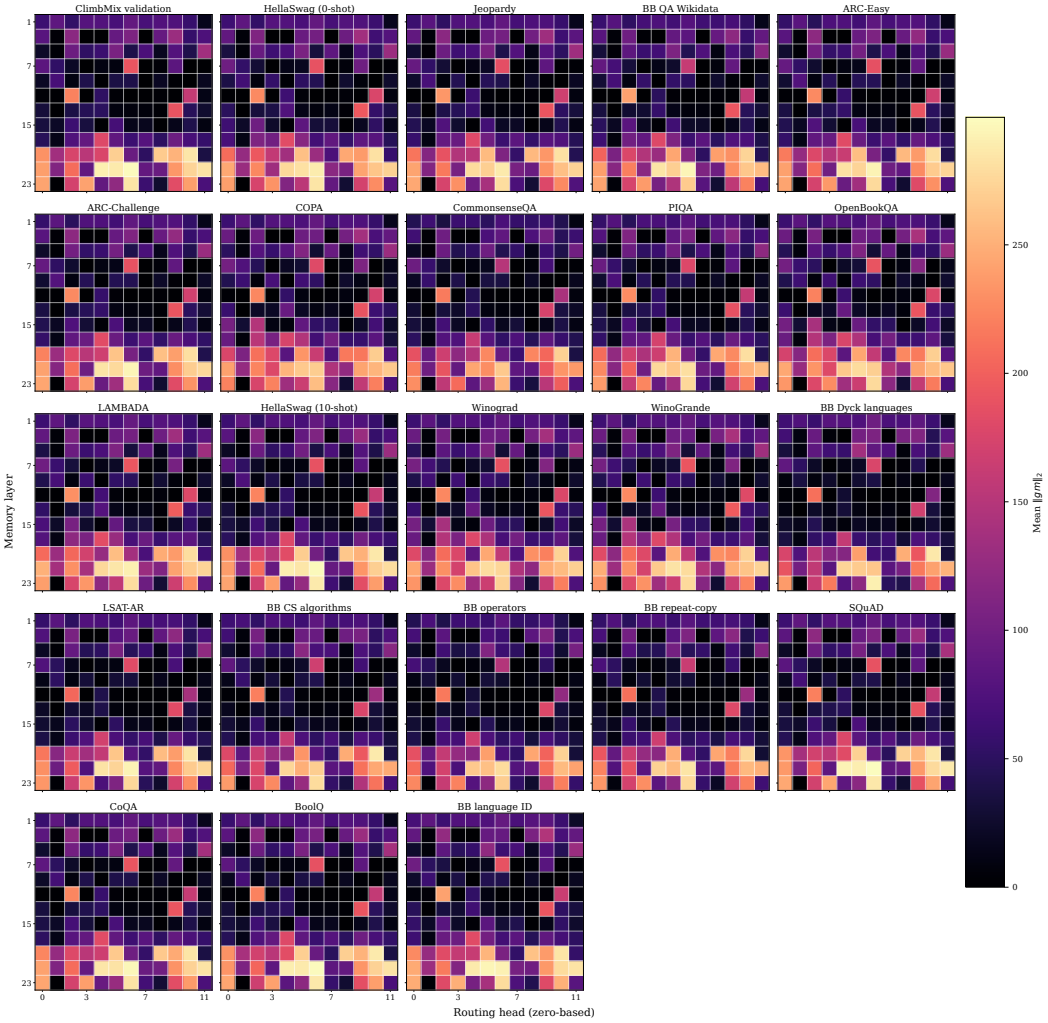


Figure 7: Actual injected memory-branch magnitude for ClimbMix validation and all 22 CORE tasks. Each cell reports $\mathbb{E}[\|\gamma \mathbf{m}\|_2]$; all panels share one absolute color scale.

The injected-memory norms in Figure 7 are largest in the late layers on both ClimbMix and the CORE tasks. These norms measure injection magnitude; they do not establish how much the memory contributes to predictions.

A.9.5 Router slot utilization across domains

We evaluate the top-2 router of the 100B-token checkpoint on 1,048,576 source-token positions from each of ClimbMix validation, FineWeb-Edu validation, enwik9, and Shakespeare. Beginning-of-sequence and padding positions are excluded. Statistics are computed at each of the 12 memory layers and 12 routing heads, then averaged over the 144 sites.

For a slot distribution p, the effective slot count is $M_{\text{eff}} = \exp(H(p))$. We compute it using either unweighted top-2 membership or routing weights. We also compute the membership distribution separately for each token ID observed at least 32 times, to assess whether repeated occurrences use different slot pairs.

Table 20: Router slot-utilization audit for the 100B-token MoME checkpoint. Effective counts are macro-averaged over 12 memory layers and 12 routing heads. Per-token counts retain token IDs with at least 32 occurrences. Inactive cells count layer–head–slot combinations never selected on the evaluated corpus. JSD compares each site's routing-weight distribution with ClimbMix. These statistics measure router concentration, not learned memory-table coverage.

Domain	Marginal effective slots	Weight-effective slots	Largest-slot mass	Per-token effective slots	Inactive site–slot cells	Mean/max JSD
ClimbMix validation	4.614	3.265	61.66%	3.550	4 / 1,728	0 / 0
FineWeb-Edu validation	4.576	3.222	61.85%	3.491	9 / 1,728	0.0026 / 0.0282
enwik9	4.528	3.209	60.95%	3.353	12 / 1,728	0.0235 / 0.1412
Shakespeare	4.230	2.959	64.01%	3.082	42 / 1,728	0.0305 / 0.2975

Table 20 shows that routing mass concentrates on a subset of slots, while repeated occurrences of the same token use different slot pairs across contexts. In every domain, the token-conditional effective slot count exceeds the number of slots selected per occurrence. Routing distributions differ more from ClimbMix on enwik9 and Shakespeare than on FineWeb-Edu.

A.10 Full Inference Latency Results

Table 21 reports the full latency sweep corresponding to the compact main-text summary in Table 5.

Table 21: Full inference latency overhead of memory-augmented value retrieval across backbone scales. Baseline is the absolute latency in milliseconds; memory columns report absolute deltas relative to the baseline, with percentage overhead in parentheses. L_{mem} denotes the number of memory-injected layers in this benchmark. *MoME (hidden-state injection)* denotes the MoME variant that gates from the input and injects back into the input stream. It is an injection-site ablation of MoME, not the canonical Engram model.

Arch	L_{mem}	Baseline ms	VE	Bigram	MoME	MoME (hidden-state injection)
`mobilellm350`	2	3.069	$+0.017$ ($+0.55\%$)	-0.036 (-1.18%)	$+0.240$ ($+7.82\%$)	$+0.247$ ($+8.05\%$)
`mobilellm350`	3	4.526	$+0.025$ ($+0.55\%$)	$+0.073$ ($+1.61\%$)	$+0.249$ ($+5.50\%$)	$+0.299$ ($+6.61\%$)
`mobilellm1b`	2	5.065	$+0.024$ ($+0.47\%$)	-0.016 (-0.31%)	$+0.277$ ($+5.47\%$)	$+0.272$ ($+5.37\%$)
`mobilellm1b`	3	7.481	$+0.035$ ($+0.47\%$)	$+0.151$ ($+2.02\%$)	$+0.295$ ($+3.95\%$)	$+0.317$ ($+4.24\%$)
`qwen3_0p6b`	2	4.768	$+0.053$ ($+1.11\%$)	-0.038 (-0.80%)	$+0.377$ ($+7.90\%$)	$+0.307$ ($+6.44\%$)
`qwen3_0p6b`	3	7.065	$+0.050$ ($+0.71\%$)	$+0.089$ ($+1.26\%$)	$+0.378$ ($+5.35\%$)	$+0.361$ ($+5.11\%$)
`qwen3_4b`	2	21.665	$+0.099$ ($+0.46\%$)	$+0.088$ ($+0.41\%$)	$+0.509$ ($+2.35\%$)	$+0.996$ ($+4.60\%$)
`qwen3_4b`	3	32.532	$+0.129$ ($+0.40\%$)	$+0.236$ ($+0.73\%$)	$+0.503$ ($+1.55\%$)	$+1.125$ ($+3.46\%$)
`qwen3_8b`	2	37.906	$+0.079$ ($+0.21\%$)	$+0.025$ ($+0.07\%$)	$+0.590$ ($+1.56\%$)	$+0.846$ ($+2.23\%$)
`qwen3_8b`	3	56.724	$+0.085$ ($+0.15\%$)	$+0.267$ ($+0.47\%$)	$+0.592$ ($+1.04\%$)	$+0.948$ ($+1.67\%$)

A.11 Supplementary Semantic-Routing Examples

We show selected examples for *apple*, *bank*, *cell*, and *python* on nanochat d24 and Qwen 0.6B. Each panel compares two prompts with the same target-word sense and a third prompt with a different sense.

Reading the panels. Each panel shows one inspected token at one probe location (Layer / Head). Prompts are grouped by semantic sense: a **sense** header is followed by the prompts that share that sense. Each prompt's heatmap is a row of memory slots; the cell colour shows the routed weight (blue $= 0$, yellow $= 1$, inactive slots are muted blue), and the numeric weight of every active slot is printed directly below its cell. The target token in each prompt is highlighted as **target**. In these selected examples, same-sense prompts favor the same slots, while the changed-sense prompt favors a different slot.

Colour scale:  $0 \rightarrow 1$.

nanochat_d24

`apple` (token id 10504) — Layer 5 / Head 10

fruit

A cook walks through a farmers market looking for fruit to bake into a pie. The vendor points to a red item from the orchard that is sweet, firm, and ready to eat with cinnamon. She sliced the **apple** into the pie filling before adding cinnamon.



A parent packs a school lunch and chooses something healthy, juicy, and easy to eat after recess. The fruit is washed at the sink and placed beside the sandwich. He packed an **apple** beside the sandwich for lunch.



company

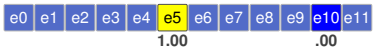
A student compares iPhone models, laptop syncing, app store purchases, device trade-ins, and warranty support before buying a new cellphone. The laptop lid showed an **apple** logo during the keynote.



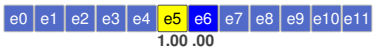
`bank` (token id 4345) — Layer 3 / Head 0

finance

A restaurant owner brings tax records, deposit history, collateral documents, and a repayment plan to a lending officer inside a downtown branch. She deposited the check at the **bank** before noon.



A family opens a savings account, asks about interest rates, checks the routing number, and deposits a paycheck with the teller. He opened a savings account at the **bank** downtown.



river_edge

After heavy rain, a fisherman walks below willow trees, watches the current, and sets his tackle box near reeds where the ground slopes into the water. The canoe scraped the muddy **bank** as it landed.



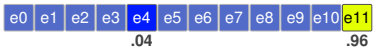
`cell` (token id 1133) — Layer 19 / Head 2

biology

In a tissue lab, a researcher studies a tiny living unit under a microscope. The notes mention membrane proteins, a nucleus, gene expression, and how the unit divides after a chemical signal. Under the microscope, each stained **cell** showed a bright nucleus.



A biology teacher draws receptors, a nucleus, and a membrane on the board. The class discusses how one living unit recognizes infection and changes its gene program during an immune response. During the immune response, the activated **cell** began dividing.



phone

During a train commute, a passenger loses wireless signal underground. The conversation is about a handset battery, carrier coverage, text messages, and whether the nearest tower supports the device. The commuter's **cell** lost service inside the tunnel.



`python` (token id 24552) Layer 5 / Head 4

programming

A data scientist is preparing a model run. She imports torch, checks a virtual environment, writes a small training script, and fixes a package version before launching the job on a GPU. The team rewrote the data loader in **python** before launch.



A backend engineer reviews an API service. The team discusses decorators, unit tests, type hints, dependency pins, and how the scripting language handles JSON requests in production. The backend team shipped the API in **python** after the review.



snake

At the reptile house, visitors watch a large snake coil under a heat lamp. The keeper explains how the animal sheds its skin, senses warmth, and swallows prey inside a humid enclosure. The keeper fed the **python** after sunset.



Qwen 0.6B

`apple` (token id 10504) Layer 9 / Head 7

fruit

A cook walks through a farmers market looking for fruit to bake into a pie. The vendor points to a red item from the orchard that is sweet, firm, and ready to eat with cinnamon. She sliced the **apple** into the pie filling before adding cinnamon.



A parent packs a school lunch and chooses something healthy, juicy, and easy to eat after recess. The fruit is washed at the sink and placed beside the sandwich. He packed an **apple** beside the sandwich for lunch.



company

A student compares iPhone models, laptop syncing, app store purchases, device trade-ins, and warranty support before buying a new cellphone. The laptop lid showed an **apple** logo during the keynote.



`bank` (token id 4345) Layer 7 / Head 7

finance

A restaurant owner brings tax records, deposit history, collateral documents, and a repayment plan to a lending officer inside a downtown branch. She deposited the check at the **bank** before noon.



A family opens a savings account, asks about interest rates, checks the routing number, and deposits a paycheck with the teller. He opened a savings account at the **bank** downtown.



river_edge

After heavy rain, a fisherman walks below willow trees, watches the current, and sets his tackle box near reeds where the ground slopes into the water. The canoe scraped the muddy **bank** as it landed.



`cell` (token id 1133) Layer 27 / Head 1

biology

In a tissue lab, a researcher studies a tiny living unit under a microscope. The notes mention membrane proteins, a nucleus, gene expression, and how the unit divides after a chemical signal. Under the microscope, each stained **cell** showed a bright nucleus.



A biology teacher draws receptors, a nucleus, and a membrane on the board. The class discusses how one living unit recognizes infection and changes its gene program during an immune response. During the immune response, the activated **cell** began dividing.



phone

During a train commute, a passenger loses wireless signal underground. The conversation is about a handset battery, carrier coverage, text messages, and whether the nearest tower supports the device. The commuter's **cell** lost service inside the tunnel.



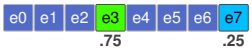
`python` (token id 24552) Layer 7 / Head 7

programming

A data scientist is preparing a model run. She imports torch, checks a virtual environment, writes a small training script, and fixes a package version before launching the job on a GPU. The team rewrote the data loader in **python** before launch.



A backend engineer reviews an API service. The team discusses decorators, unit tests, type hints, dependency pins, and how the scripting language handles JSON requests in production. The backend team shipped the API in **python** after the review.



snake

At the reptile house, visitors watch a large snake coil under a heat lamp. The keeper explains how the animal sheds its skin, senses warmth, and swallows prey inside a humid enclosure. The keeper fed the **python** after sunset.



A.12 Quantifying Sense-Sensitive Routing on WiC

Dataset and filtering. We complement the curated examples above with a quantitative probe on the union of the train and test splits of WiC [48]; no WiC examples are used to fit or tune the model or a probe. Each WiC item contains two natural sentences with the same marked target word and a binary label: T if the target has the same sense in both sentences and F otherwise. Starting from all 6,828 pairs (5,428 train and 1,400 test), we retain a *strict-natural* pair only when the marked surface in both sentences matches the canonical target case-insensitively, each occurrence is represented by exactly one tokenizer token, and the two occurrences have the same token ID. This leaves 3,206 pairs (2,655 train and 551 test).

Because the memory router is causal and can only use the prefix preceding the target, we further require both marked occurrences to lie strictly after the midpoint of their tokenized sentence. Let $p(x)$ denote the target position in a sequence that includes the beginning-of-sequence token and let $L(x)$ denote the corresponding sequence length. We retain occurrence x when

$$r(x) = \frac{p(x) - \frac{1}{2}}{L(x) - 1} > \frac{1}{2}, \tag{2}$$

and retain a pair only if both occurrences pass. The resulting evaluation set contains 670 pairs (549 train and 121 test; 354 T and 316 F), 362 unique target-token IDs, and 1,340 target occurrences. The minimum preceding-context length has a median of 5 tokens and an interquartile range of 4–6; 547 pairs have at least four preceding tokens. The retained set is heavily noun-skewed (634 noun pairs and 36 verb pairs).

Per-head metrics. We analyze routing in the $d24$ MoME model at each layer–head site $s = (\ell, h)$. The model has memory at the 12 odd-numbered layers $\{1, 3, \ldots, 23\}$, with 12 routing heads per layer and top-2 selection from 12 slots. For WiC pair i, let $\mathbf{q}_{i1,s}$ and $\mathbf{q}_{i2,s}$ be the full routing distributions for its two target occurrences, and let $\mathbf{m}_{i,s} = (\mathbf{q}_{i1,s} + \mathbf{q}_{i2,s})/2$. We compute the base-2 Jensen–Shannon divergence

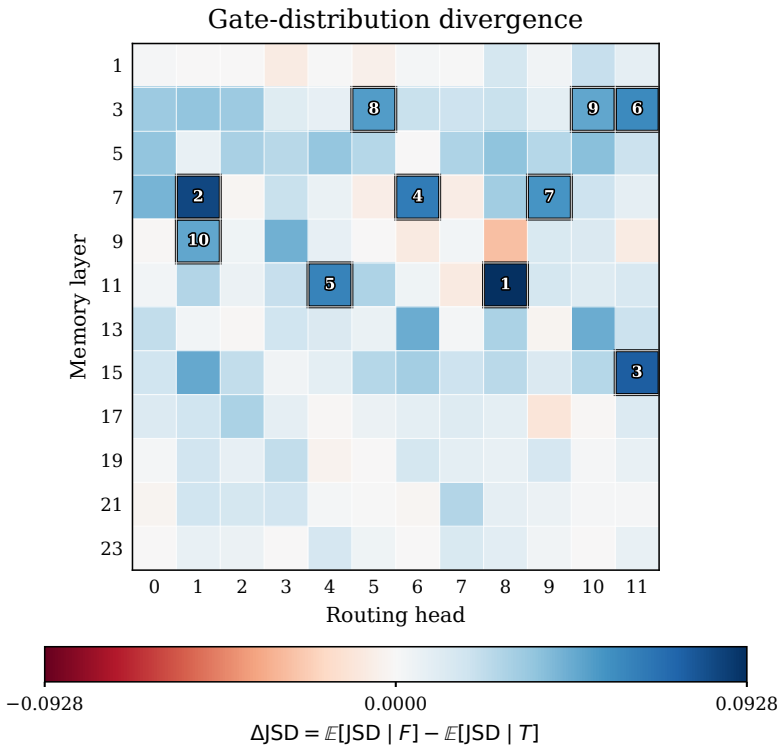
$$d_{i,s} = \frac{1}{2}\mathrm{KL}_2(\mathbf{q}_{i1,s} \| \mathbf{m}_{i,s}) + \frac{1}{2}\mathrm{KL}_2(\mathbf{q}_{i2,s} \| \mathbf{m}_{i,s}), \qquad \Delta\mathrm{JSD}_s = \mathbb{E}[d_{i,s} \mid F] - \mathbb{E}[d_{i,s} \mid T]. \tag{3}$$

Positive $\Delta\mathrm{JSD}_s$ indicates greater average routing divergence for different-sense pairs than for same-sense pairs.

We also measure chance-corrected decision overlap following Olson et al. [44]. Let $A_{i1,s}$ and $A_{i2,s}$ be the executed top-K slot sets, let $o_{i,s} = |A_{i1,s} \cap A_{i2,s}|$, and let M be the number of slots. We compute

$$\widetilde{O}_{i,s} = \frac{o_{i,s} - K^2/M}{K - K^2/M}, \qquad \Delta\widetilde{O}_s = \mathbb{E}[\widetilde{O}_{i,s} \mid T] - \mathbb{E}[\widetilde{O}_{i,s} \mid F]. \tag{4}$$

Positive $\Delta\widetilde{O}_s$ indicates greater slot overlap for same-sense pairs, after correcting for uniform-random top-K selection.



Blue: expected sense-sensitive direction. Red: reverse direction. Numbers mark the top ten sites.

Figure 8: Routing-distribution differences on the 670 filtered WiC pairs. Cells correspond to layer–head sites. Blue indicates greater divergence for different-sense pairs ($\Delta\text{JSD}_s > 0$), and red indicates the reverse. Outlined numbers rank the ten largest positive effects. Head indices are zero-based.

Results. In Figure 8, different-sense pairs have greater routing divergence at 117 of 144 sites. Most of the largest differences occur in the middle layers.

Layer 11/head 8 has the largest effect, with $\Delta\text{JSD} = 0.0928$ and a pair-stratified bootstrap 95% interval of $[0.0489, 0.1357]$. All ten largest effects have pointwise bootstrap intervals above zero.

Decision overlap is greater for same-sense pairs at most sites. The largest effect again occurs at layer 11/head 8, with $\Delta\widetilde{O} = 0.0939$ and a pair-stratified bootstrap 95% interval of $[0.0360, 0.1520]$.

Injection-gate-aware sensitivity. To account for injection strength, we measure the per-head injection gate $\gamma_{ij,s} \in [0, 2]$ for occurrence $j \in \{1, 2\}$ of pair i at site s. We weight each pair by $w_{i,s} = \sqrt{\gamma_{i1,s}\gamma_{i2,s}}/2 \in [0, 1]$, which decreases when either occurrence has a small injection gate, and compute

$$\Delta\text{JSD}_s^{\text{gate}} = \mathbb{E}[w_{i,s}d_{i,s} \mid F] - \mathbb{E}[w_{i,s}d_{i,s} \mid T]. \tag{5}$$

We apply the weights before computing the class means. We also recompute both routing metrics after retaining only pairs with $\min(\gamma_{i1,s}, \gamma_{i2,s}) \geq \tau$, for $\tau \in \{0.1, 0.2, 0.25\}$. Thresholded cells with fewer than ten pairs in either sense class are shown in gray. Figure 9 shows the first two JSD thresholds; Figure 4 reports the unfiltered effects for both metrics.

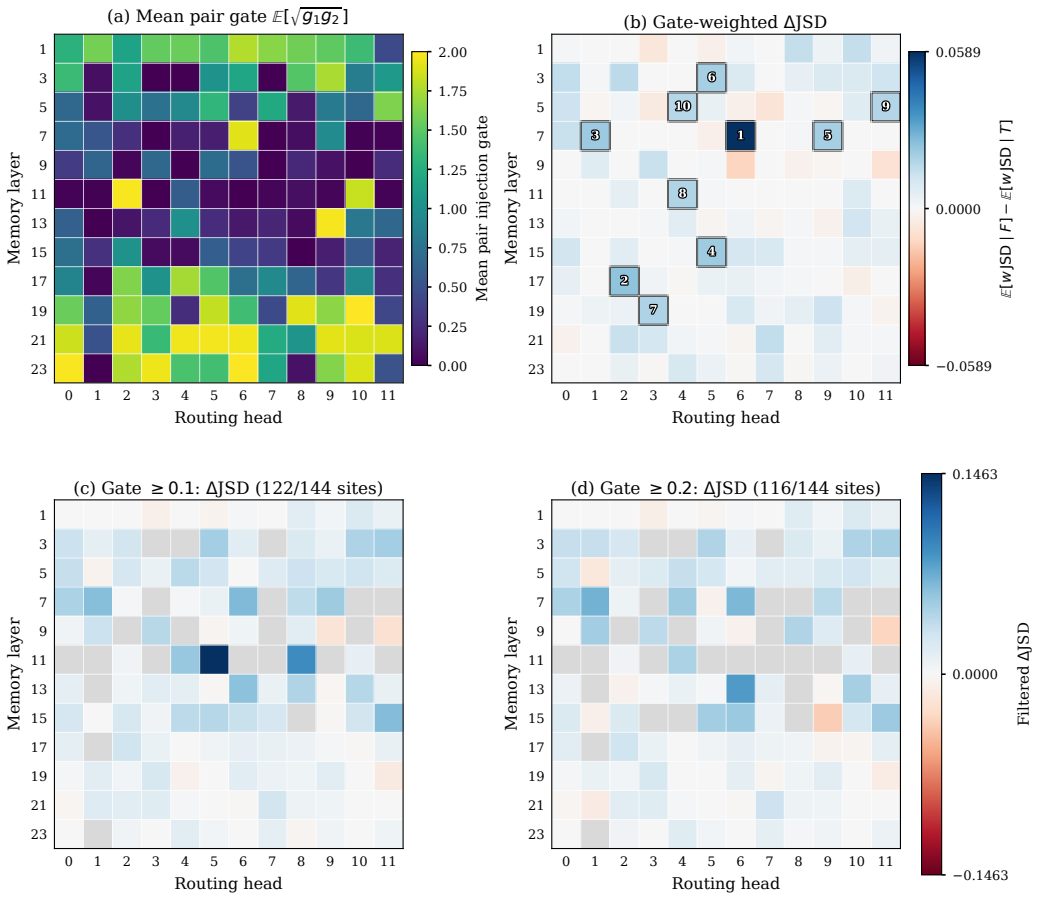


Figure 9: Injection-gate sensitivity analysis on the same WiC pairs as Figure 8. (a) Mean unnormalized pair gate $\mathbb{E}[\sqrt{\gamma_{i1,s}\gamma_{i2,s}}]$. (b) Gate-weighted effect from Equation 5, with the ten largest positive effects ranked. (c–d) Unweighted ΔJSD_s for pairs whose two injection gates exceed 0.1 or 0.2, using a shared color scale. Gray cells have fewer than ten retained pairs in either class. Blue indicates positive effects; head indices are zero-based.

Accounting for injection strength changes which heads have the largest effects (Figure 9). Layer 11/head 8 has the largest unweighted routing difference but a nearly closed injection gate. Layer 7/head 6, which has a nearly open gate, has the largest gate-weighted effect.

At layer 7/head 6, $\Delta\text{JSD}^{\text{gate}} = 0.0589$, with a pair-stratified bootstrap 95% interval of $[0.0167, 0.1016]$. Fifteen sites have pointwise weighted intervals above zero. The unweighted metric measures routing divergence; the weighted metric shows how that divergence changes when pairs with small injection gates contribute less.

At the strictest gate threshold, $\tau = 0.25$, most retained sites still show greater routing divergence for different-sense pairs and greater slot overlap for same-sense pairs.

Scope and limitations. Routing differences are associated with word sense, including at heads with open injection gates, but this analysis does not measure their effect on predictions. Only 81 target-token groups contain both sense labels, so the pooled comparison may reflect differences in lexical composition despite holding the target token fixed within each pair. The analysis combines WiC train and test examples and does not estimate held-out generalization. Bootstrap intervals are pointwise, and the largest effects are selected on the same examples without correction across the 144 sites. Confirmatory testing would require separate data with repeated target types and a site-wise label-permutation test with max-T or false-discovery-rate correction.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes, the claims in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: Yes see discussion session.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: No major theoretical results are included in the paper.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The architecture, routing/aggregation, and injection mechanisms are fully specified in Section 3; experiment protocol and per-family configurations are in Section 4.2 and Appendices A.2, A.3, A.4, A.5, and A.6.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [No]

   Justification: The code is undergoing an internal review process and will be released once that process is complete; it is not available at submission time.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.2 gives the data, tokenizer, optimizer, batch, and baseline settings; full per-family details are in Appendices A.3, A.4, and A.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: error bars are reported for main experiment.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: In the appendix

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: Research on fundamental properties of neural networks is not expected to have direct societal impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: Our research does not have such risks.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Training data, evaluation suite, and baseline implementations follow the nanochat-collected versions [29]; we credit nanochat and inherit its licensing terms for the assets it bundles.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [N/A]

Justification: Our paper does not introduce new assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: Our paper does not include crowdsourcing experiments or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: Our paper does not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: Our paper does not use LLM beyond writing, editing, or formatting purposes.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.