Memory Grafting: Scaling Language Model Pre-training via Offline Conditional Memory

Runxi Cheng[1] **Yuchen Guan**[1] **Yongxian Wei**[1] **Qianpu Sun**[1] **Qixiu Li**[1]
Sinan Du[1] **Feng Xiong**[1] **Chun Yuan**[1*] **Yan Lu**[2*] **Yeyun Gong**[2*]
[1]Tsinghua University, [2]Microsoft Research Asia
{crx23@mails.tsinghua.edu.cn; yegong@microsoft.com}

Abstract

Scaling conditional memory offers a promising way to increase language-model capacity, but existing methods such as Engram learn large memory tables from scratch during pre-training, making memory scaling expensive and sometimes ineffective. We propose *Memory Grafting*, a conditional memory scaling method that utilize frozen hidden states from a grafting model as conditional n-gram memory. Given frequent local n-grams, we run the grafting model offline, store final-token hidden representations as memory values, and let the recipient model retrieve them through exact longest-match suffix lookup. Retrieved memories are adapted by lightweight projections and gates, while a hash-based Engram fallback preserves coverage for unmatched contexts. Since the grafting model is only run offline and exact lookup has expected $\mathcal{O}(1)$ complexity with respect to memory-bank size, Memory Grafting expands external latent capacity with limited training and inference overhead. Experiments under matched recipient architectures and pre-training budgets show that Memory Grafting improves over both MoE and vanilla Engram baselines. In the 2.8B-scale setting, it improves the average benchmark score from 51.95 for MoE and 52.43 for vanilla Engram to 53.86. In the 0.92B–scale setting, all grafting-model variants improve over the baselines, with Qwen3.5-35B-A3B giving the strongest gains. These results suggest that pretrained models can serve as reusable constructors of external latent memory, providing a practical step toward scaling future language models beyond trainable parameters alone.

1 Introduction

The rapid progress of large language models [1, 13, 28, 41, 42] over the past few years has been driven in large part by joint scaling of model parameters and pre-training tokens [17, 22]. As frontier models have grown to hundreds of billions to trillions of parameters trained on trillions of tokens, the engineering cost of each pre-training run has risen substantially. In response, recent work has increasingly focused on architectural sparsity, where only a fraction of a model's total capacity participates in any individual forward pass. Mixture-of-Experts (MoE) architectures [6, 10, 37] are a representative instance and underlie a number of recent open-weight systems. Another direction [3, 18, 25] scales capacity through sparse memory lookup rather than sparse computation, attaching a large external table that is queried at each token. We refer to this line of work as conditional memory.

Conditional memory offers a scalable way to expand model capacity while adding only limited inference overhead. In particular, Engram [3] uses deterministic hash-based addressing to support $\mathcal{O}(1)$ retrieval per token, and reports that offloading a 100B-parameter memory only costs around 2% end-to-end inference throughput on a 4B-dense backbone. However, the training side is less

*Corresponding authors.

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

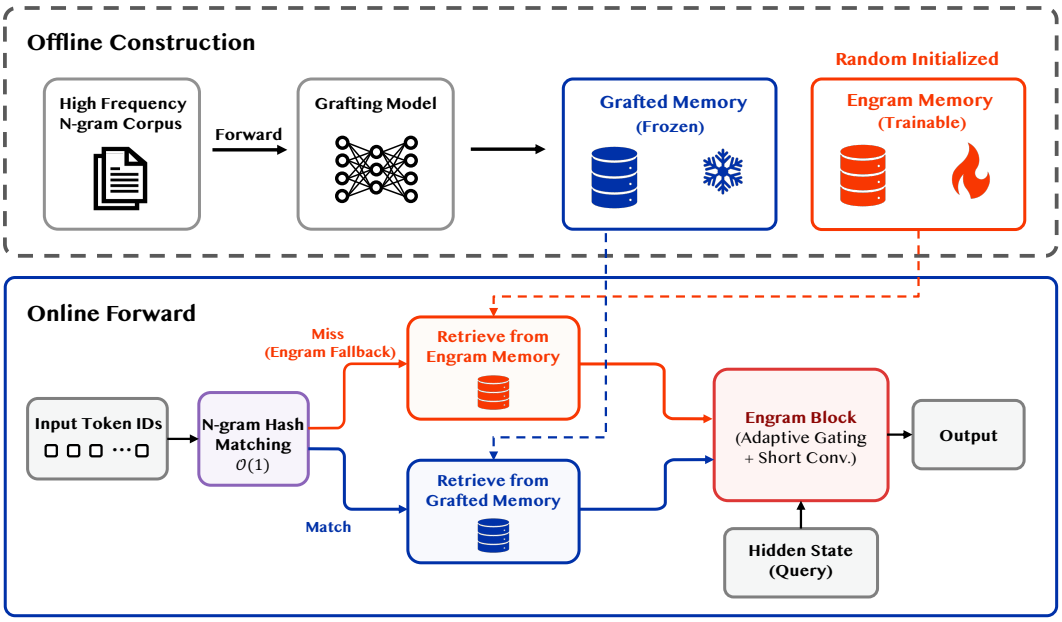

Figure 1: Computation pipeline for Memory Grafting

favorable. The memory table is learned from scratch during pre-training, so scaling it increases the number of parameters to optimize. Empirically, training efficiency declines once the memory module accounts for more than 70% of total parameters. Thus, scaling Engram still demands substantial training-time computation.

To solve this problem, we revisit what Engram memory stores [3, 23, 36]: an n-gram-indexed table of static embeddings. Such entries naturally correspond to recurring multi-token constructions – named entities, formulaic phrases, library calls, and mathematical notation [29] – whose meaning is largely fixed by surface form. A sufficiently pre-trained model has already learned strong representations for these patterns. If these encodings can be used as conditional memory, the memory capacity can scale without increasing the recipient's training cost, while the $\mathcal{O}(1)$ lookup that makes conditional memory efficient at inference remains intact.

Motivated by this observation, we propose Memory Grafting, which constructs conditional memory from a grafting model rather than learning the memory table entirely from scratch. We run the grafting model once offline over frequent local n-grams and store the resulting hidden states as frozen memory values. Keeping these entries frozen separates memory construction from recipient optimization and lets the recipient focus on adapting reusable features to its own hidden space. During recipient training and inference, the model retrieves these values through longest-match exact lookup, falls back to hash-based Engram memory on misses, and injects the selected memory through lightweight projection and gating. The context-dependent gate allows the same retrieved memory to be weighted differently across occurrences, according to the recipient's current hidden state. The grafting model runs once offline; input sequence length is unchanged, and per-token lookup remains $\mathcal{O}(1)$. This keeps training and inference efficient as memory scales. Our contributions are summarized as follows:

- We introduce **Memory Grafting**, a representation-level knowledge-transfer paradigm for scaling conditional memory. It stores frozen grafting-model hidden states in an exact n-gram-indexed table, retains a hash-based Engram fallback for unmatched contexts, and grafts retrieved memories through a gated residual update. The grafting-model cost is paid once offline, and per-token lookup remains $\mathcal{O}(1)$.
- We evaluate Memory Grafting under matched recipient architectures and pre-training budgets. The results show gains over MoE and vanilla Engram baselines, increasing exact-memory coverage as the table grows, practical inference efficiency, and probing evidence that the recipient actually uses the grafted memory.

2 Related Work

2.1 Conditional Memory and Modular Sparse Scaling

Efficiently scaling model capacity is crucial for advancing the performance of large language models. Sparse scaling and conditional memory represent two prominent directions in this pursuit. Mixture-of-Experts architectures [6, 10, 37], which activate only a fraction of the total capacity during any single forward pass, have become a widely adopted sparse paradigm in recent open-weight systems. Parallel to sparse computation, another line of research focuses on scaling capacity through sparse memory lookups rather than computation, a mechanism collectively termed conditional memory [3, 39, 40]. For instance, Product-Key Memory [25] replaces standard feed-forward layers with a massive key-value store, enabling sparse nearest-neighbor search via Cartesian product sets. Building on this, UltraMem [18] resolves sparse retrieval bottlenecks and scaling inefficiencies through tucker decomposed query-key retrieval and implicit value expansion. STEM [34] replaces the FFN up-projection in Transformer layers with token-indexed embedding modules, reducing per-token computation while increasing parameter capacity through input-dependent embedding lookup. MoLE [21] improves MoE inference efficiency by training experts as FFNs and re-parameterizing them into input-id-based lookup tables before inference, thereby reducing expert computation and offloading overhead while preserving MoE-like capacity. More recently, Engram [3] learns an n-gram vocabulary as its memory module, employing hash-based addressing to achieve $\mathcal{O}(1)$ sparse retrieval during the forward pass. Such conditional memory offers a clear inference-side advantage, but scaling it still requires substantial computation when the memory itself is learned from scratch during pre-training.

2.2 Retrieval-Augmented Generation and Knowledge Distillation

Beyond conditional memory, another line of work improves language models by leveraging external memory sources in different ways. Retrieval-Augmented Generation [26] queries external non-parametric memories or documents during training or inference, with pioneering works such as REALM [16] casting both pre-training and fine-tuning as an end-to-end retrieve-then-predict process, and recent work [19, 27] further refining retrieval reasoning and extraction quality. In contrast, Knowledge Distillation transfers the parametric memory of a larger teacher into a recipient model, improving its capability without explicit retrieval at inference time [14, 32].

Although effective, RAG and KD introduce different efficiency bottlenecks. RAG relies on retrieval and longer augmented contexts at inference, shifting substantial overhead to deployment and increasing latency [12, 20, 47]. KD instead internalizes external knowledge into the student's parameters, but logits-based distillation during from-scratch pre-training requires running the teacher across the entire corpus and consuming large soft-label distributions, incurring substantial compute and memory cost. In contrast, Memory Grafting needs only a lightweight one-time offline pass to build the grafted memory table, preserving high efficiency in both training and inference and showing strong potential for model scaling.

3 Method

3.1 Preliminaries on Engram

Engram [3] is a conditional memory module for language models. Unlike MoE, which sparsely activates computation experts, Engram sparsely retrieves memory indexed by local token patterns. Given a token sequence $\mathbf{X} = (x_1, \ldots, x_T)$ and hidden states $\mathbf{H}^{(\ell)} \in \mathbb{R}^{T \times D}$ at layer ℓ, Engram performs two steps: hashed N-gram retrieval and context-aware fusion.

For each position t, Engram forms suffix n-grams from the local history. Following the original design, token ids may first be mapped through a vocabulary compression function $P : \mathcal{V} \to \mathcal{V}'$, so that textually equivalent tokens share a canonical id. The compressed suffix n-gram is:

$$\mathbf{g}_{t,n} = \big(P(x_{t-n+1}), \ldots, P(x_t)\big), \qquad n \in \{2, \ldots, N\}. \tag{1}$$

Because enumerating all possible N-grams is infeasible, Engram uses multi-head hashing. For each order n and hash head k, a deterministic hash function $\phi_{n,k}$ maps $\mathbf{g}_{t,n}$ to an embedding-table row:

$$z_{t,n,k} = \phi_{n,k}(\mathbf{g}_{t,n}), \qquad \mathbf{e}_{t,n,k} = \mathbf{E}_{n,k}[z_{t,n,k}]. \tag{2}$$

Algorithm 1: Memory Grafting with Engram Fallback

Input: Hidden states $\mathbf{H} \in \mathbb{R}^{B\times T\times C\times D}$, tokens $\mathbf{X}$, frozen memory $(\mathcal{M}, \mathbf{W}_{\text{mem}})$, trainable Engram table $\mathbf{W}_{\text{hash}}$, projections $\mathbf{W}_{k,v}^{\text{mem}}$, $\mathbf{W}_{k,v}^{\text{eng}}$
Output: Grafted hidden states $\widetilde{\mathbf{H}}$
$\mathbf{M}_{\text{mem}} \leftarrow \mathbf{0}$, $\mathbf{m} \leftarrow \mathbf{0}$ // Initialize memory features and hit mask
for each token position (b, t) **do**
  $j \leftarrow \texttt{ExactLookup}(\mathbf{X}_{b,\le t}, \mathcal{M})$ // Select the largest matched n-gram
  if j exists **then**
    $\mathbf{M}_{\text{mem}}[b,t] \leftarrow \mathbf{W}_{\text{mem}}[j]$, $\mathbf{m}[b,t] \leftarrow 1$
  end if
end for
for each token position (b, t) **do**
  if $\mathbf{m}[b,t] = 1$ **then**
    $\mathbf{e}_{b,t} \leftarrow \mathbf{M}_{\text{mem}}[b,t]$, $(\mathbf{K}_{b,t}, \mathbf{V}_{b,t}) \leftarrow (\mathbf{W}_k^{\text{mem}}\mathbf{e}_{b,t}, \mathbf{W}_v^{\text{mem}}\mathbf{e}_{b,t})$ // Memory grafting path
  else
    $\mathbf{z}_{b,t} \leftarrow \texttt{NgramHashMapping}(\mathbf{X}_{b,\le t})$ // Generate deterministic hash ids
    $\mathbf{E}_{b,t} \leftarrow \texttt{MultiHeadEmbedding}(\mathbf{z}_{b,t})$ // Lookup trainable Engram table
    $\mathbf{e}_{b,t} \leftarrow \texttt{Concat}(\mathbf{E}_{b,t})$
    $(\mathbf{K}_{b,t}, \mathbf{V}_{b,t}) \leftarrow (\mathbf{W}_k^{\text{eng}}\mathbf{e}_{b,t}, \mathbf{W}_v^{\text{eng}}\mathbf{e}_{b,t})$ // Engram fallback path
  end if
end for
$\mathbf{Q} \leftarrow \mathrm{RMSNorm}_q(\mathbf{H})$, $\bar{\mathbf{K}} \leftarrow \mathrm{RMSNorm}_k(\mathbf{K})$
$\boldsymbol{\alpha} \leftarrow \sigma(\langle \bar{\mathbf{K}}, \mathbf{Q}\rangle/\sqrt{D})$ // Query-key gate
$\mathbf{U} \leftarrow \boldsymbol{\alpha} \odot \mathbf{V}$, $\Delta\mathbf{H} \leftarrow \mathbf{U} + \texttt{ShortConv}(\mathbf{U})$
$\widetilde{\mathbf{H}} \leftarrow \mathbf{H} + \Delta\mathbf{H}$
return $\widetilde{\mathbf{H}}$

The retrieved embeddings across all n-gram orders and hash heads are concatenated into a static memory vector:

$$\mathbf{e}_t = \Big\Vert_{n=2}^{N} \Big\Vert_{k=1}^{K} \mathbf{e}_{t,n,k}. \tag{3}$$

Since $\mathbf{e}_t$ is retrieved only from the local token pattern, it is not by itself aware of the global context. Engram therefore uses the current hidden state as a query to gate the retrieved memory. For a branch c of the residual stream, let $\mathbf{h}_t^{(c)} \in \mathbb{R}^D$ be the hidden state. The memory vector is projected into key and value vectors:

$$\mathbf{k}_t^{(c)} = \mathbf{W}_K^{(c)}\mathbf{e}_t, \qquad \mathbf{v}_t = \mathbf{W}_V\mathbf{e}_t. \tag{4}$$

The gate is computed by comparing the normalized key with the normalized query [3, 9, 46]:

$$\alpha_t^{(c)} = \sigma\left(\frac{\left\langle \mathrm{RMSNorm}(\mathbf{k}_t^{(c)}), \mathrm{RMSNorm}(\mathbf{h}_t^{(c)})\right\rangle}{\sqrt{D}}\right) \tag{5}$$

The gated value is then processed by a lightweight depthwise causal convolution:

$$\mathbf{u}_t^{(c)} = \alpha_t^{(c)}\mathbf{v}_t, \qquad \Delta\mathbf{h}_t^{(c)} = \mathbf{u}_t^{(c)} + \mathrm{ShortConv}(\mathbf{U}^{(c)})_t. \tag{6}$$

3.2 Memory Grafting

Original Engram learns a hash-indexed memory table from scratch. Memory Grafting instead constructs an external memory bank from latent representations of a grafting model and grafts the retrieved features into a recipient model. This changes the role of conditional memory from purely trainable storage to reusable representation-level knowledge transfer.

Frozen Latent Memory Construction. We first build a set of frequent n-gram keys $\mathcal{M} = \{\mathbf{m}_i\}_{i=1}^{M}$, where each $\mathbf{m}_i$ is a token-id tuple such as a 2-, 3-, or 4-gram. For each key $\mathbf{m}_i$, we run a grafting model F_G offline and extract a hidden representation of its original text from a selected grafting

layer. In our implementation, we use the hidden state at the final token of the n-gram as the memory embedding. This produces a frozen table

$$\mathbf{W}_{\mathrm{mem}}[i] = F_G(\texttt{text}(\mathbf{m}_i))^{(r)}_{\mathrm{last}} \in \mathbb{R}^{D_{\mathrm{mem}}}, \tag{7}$$

where r is the grafting layer used for extraction. The mapping from $\mathbf{m}_i$ to row index i is stored as an exact integer lookup structure. The table $\mathbf{W}_{\mathrm{mem}}$ is not updated during recipient-model training; only the recipient-side projection and gating parameters are learned.

Exact Retrieval with Longest-Match Priority. At a grafting layer of the recipient model, let $\mathbf{H} \in \mathbb{R}^{B\times T\times C\times D}$ denote the hidden states, where B is the batch size, T is the sequence length, C is the number of residual branches, and D is the branch dimension. For each position (b, t), Memory Grafting queries the frozen memory using suffix n-grams ending at $x_{b,t}$. If multiple keys match the same position, we keep the longest matched n-gram, e.g., a 4-gram match overrides a 3-gram match. This yields a memory feature $\mathbf{E}_{\mathrm{mem},b,t}$ and a binary hit mask $m_{b,t} \in \{0, 1\}$:

$$(j_{b,t}, m_{b,t}) = \texttt{ExactLookup}(\mathbf{X}_{b,\le t}, \mathcal{M}), \qquad \mathbf{E}_{\mathrm{mem},b,t} = \mathbf{W}_{\mathrm{mem}}[j_{b,t}] \quad \text{if } m_{b,t} = 1. \tag{8}$$

The exact lookup is independent of the total memory size in expectation under hash-map or trie-style indexing, and costs only a constant number of suffix queries per token because the maximum n-gram order is fixed.

Memory Grafting with Engram Fallback. The frozen memory covers only the selected frequent n-grams. To preserve full coverage over arbitrary inputs, we retain the original hash-based Engram path as a fallback. For every position, we compute a trainable hash Engram feature:

$$\mathbf{E}_{\mathrm{hash},b,t} = \texttt{Concat}\big(\texttt{MultiHeadEmbedding}(\texttt{NgramHashMapping}(\mathbf{X}_{b,\le t}))\big). \tag{9}$$

Memory-hit positions use the frozen grafted memory, whereas memory-miss positions use the trainable Engram fallback. We use separate projections for the two sources:

$$\mathbf{K}^c_{\mathrm{mem},b,t} = \mathbf{W}^{\mathrm{mem},c}_k \mathbf{E}_{\mathrm{mem},b,t}, \qquad \mathbf{V}_{\mathrm{mem},b,t} = \mathbf{W}^{\mathrm{mem}}_v \mathbf{E}_{\mathrm{mem},b,t}, \tag{10}$$

$$\mathbf{K}^c_{\mathrm{eng},b,t} = \mathbf{W}^{\mathrm{eng},c}_k \mathbf{E}_{\mathrm{hash},b,t}, \qquad \mathbf{V}_{\mathrm{eng},b,t} = \mathbf{W}^{\mathrm{eng}}_v \mathbf{E}_{\mathrm{hash},b,t}. \tag{11}$$

The branch-specific key and shared value used by the gate are selected by the hit mask:

$$\mathbf{K}^c_{b,t} = m_{b,t}\mathbf{K}^c_{\mathrm{mem},b,t} + (1 - m_{b,t})\mathbf{K}^c_{\mathrm{eng},b,t}, \qquad \mathbf{V}_{b,t} = m_{b,t}\mathbf{V}_{\mathrm{mem},b,t} + (1 - m_{b,t})\mathbf{V}_{\mathrm{eng},b,t}. \tag{12}$$

Gated Grafting. The selected memory value is grafted into the decoder using the same context-aware gating principle as Engram. For each branch c, we compute

$$\alpha^c_{b,t} = \sigma\left(\frac{\left\langle \bar{\mathbf{K}}^c_{b,t}, \mathbf{Q}^c_{b,t} \right\rangle}{\sqrt{D}}\right), \text{ where } \mathbf{Q}^c_{b,t} = \mathrm{RMSNorm}_q(\mathbf{H}^c_{b,t}),\ \bar{\mathbf{K}}^c_{b,t} = \mathrm{RMSNorm}_k(\mathbf{K}^c_{b,t}) \tag{13}$$

The output can be calculated as:

$$\mathbf{U}^c_{b,t} = \alpha^c_{b,t}\mathbf{V}_{b,t}, \qquad \Delta\mathbf{H} = \mathbf{U} + \texttt{ShortConv}(\mathbf{U}), \qquad \widetilde{\mathbf{H}} = \mathbf{H} + \Delta\mathbf{H}. \tag{14}$$

4 Experiments

4.1 Experimental Setup

Model Architectures. We implement two recipient model scales [43], with total parameters around 0.9B and 2.8B. For both scales, the MoE layers contain 64 routed experts, and following DeepSeek-MoE [6], we use one shared expert in each MoE layer. For the Engram module, we follow the best parameter allocation reported in the Engram paper and set the parameter ratio between Engram memory and MoE computation to $1{:}3$. Since Memory Grafting introduces additional lightweight projection parameters for adapting the frozen memory features to the recipient hidden space, we reduce the corresponding number of experts to keep the fair comparison. Across all experiments, we ensure that the number of trainable parameters follows Memory Grafting $\le$ vanilla Engram $\le$ MoE baseline.

Table 1: Benchmark performance (%) in the 2.8B-trainable, 100B-token setting.

Benchmark *(Metric)*	**# Shots**	**MoE Baseline**	**Vanilla Engram**	**Memory Grafting**
# Total Params		2.8B	2.8B	2.8B
# Activated *(w/o token embed)*		0.55B	0.55B	0.55B
# Trained Tokens		100B	100B	100B
# Experts *(shared + routed, top-k)*		1 + 64 (top-4)	1 + 48 (top-4)	1 + 47 (top-4)
ARC-Challenge	0-shot	36.12	36.09	**37.03**
ARC-Easy	0-shot	72.39	71.80	**73.40**
BoolQ	0-shot	57.92	60.24	**62.54**
Social IQA	0-shot	41.61	41.76	**42.94**
RACE	0-shot	34.74	34.55	**35.98**
LAMBADA	5-shot	42.98	45.22	**48.19**
WinoGrande	5-shot	58.98	59.66	**60.93**
PIQA	5-shot	75.41	75.63	**76.17**
HellaSwag	10-shot	47.39	46.94	**47.54**
Average		51.95	52.43	**53.86**

Table 2: Result in the 0.92B-trainable, 50B-token setting with different grafting models. MG-X denotes Memory Grafting using model X as the grafting model: MG-GLM refers to GLM-4.7-Flash, MG-Deepseek refers to DeepSeek-V2-Lite, and MG-Qwen3.5 refers to Qwen3.5-35B-A3B.

Benchmark *(Metric)*	**# Shots**	**MoE Baseline**	**Vanilla Engram**	**MG-GLM**	**MG-Deepseek**	**MG-Qwen3.5**
# Trainable Params		0.92B	0.92B	0.92B	0.92B	0.92B
# Activated *(w/o token embed)*		0.29B	0.29B	0.29B	0.29B	0.29B
# Trained Tokens		50B	50B	50B	50B	50B
# Experts *(shared + routed, top-k)*		1 + 64 (top-4)	1 + 48 (top-4)	1 + 46 (top-4)	1 + 46 (top-4)	1 + 46 (top-4)
ARC-Challenge	0-shot	27.56	28.92	28.07	28.92	**29.35**
ARC-Easy	0-shot	64.31	64.31	62.96	**64.39**	64.06
BoolQ	0-shot	55.93	46.97	53.03	56.64	**58.10**
Social IQA	0-shot	39.36	40.28	40.12	40.48	**40.84**
RACE	0-shot	**31.77**	31.39	30.05	30.62	31.58
LAMBADA	5-shot	31.88	32.27	**34.37**	32.52	33.73
WinoGrande	5-shot	51.07	51.85	53.51	53.51	**55.01**
PIQA	5-shot	70.67	70.95	71.87	**71.93**	71.44
HellaSwag	10-shot	38.02	38.34	38.48	**38.80**	38.74
Average		45.62	45.03	45.83	46.42	**46.98**

Data & Tokenizer. We trained the 925M parameter model on a 50B-token subset of the Nemotron-CC dataset [38] and the 2.81B parameter model on a 100B-token subset of the Nemotron-CC dataset. All text was tokenized with the LLaMA-3-8B tokenizer [15].

Grafting Setting. We use Qwen3.5-35B-A3B [33], DeepSeek-V2-Lite [8], and GLM-4.7-Flash [45] as grafting models. For each n-gram order, we select the top 1M most frequent n-grams and compute their frozen memory representations offline, yielding 3M entries per grafted layer. Each entry stores a hidden-state vector of the grafting model in `bfloat16`; for Qwen3.5-35B-A3B , this corresponds to roughly 12 GB of frozen memory per layer. To cover different representation levels of the grafting model, we extract hidden states from an early-to-middle layer and a deeper layer, approximately corresponding to 10% and 60% of the model depth. In particular, we use Layers 8, 24 for Qwen3.5-35B-A3B, Layers 4, 18 for DeepSeek-V2-Lite, and Layers 4, 28 for GLM-4.7-Flash.

Hyper-Parameters. The hyper-parameters are selected based on the common practice for dense language models. We replace all FFN layers with MoE layer in the transformer. Please refer Section A for detailed training hyper-parameters.

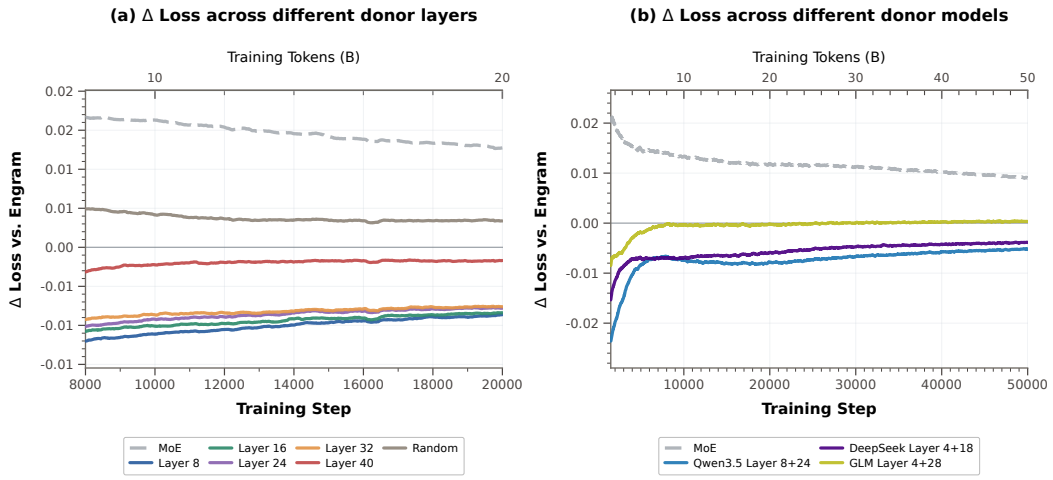

Figure 2: Loss differences relative to vanilla Engram when varying the grafted memory source. (a) Comparison across different Qwen3.5-35B-A3B source layers (b) Comparison across different grafting models.

Benchmarks. We use the lm-evaluation-harness [11] for evaluation. The benchmarks used include ARC-C [5],ARC-E[5], BoolQ [4], HellaSwag [44], LAMBADA [31], PIQA [2], RACE [24], WinoGrande [35], .

4.2 Main Results

We report two main sets of experimental results. Table 1 uses a 2.8B total / 0.55B activated MoE backbone trained for 100B tokens and compares the MoE baseline, vanilla Engram, and Memory Grafting with a Qwen3.5-35B-A3B grafting model. Memory Grafting is best on all nine benchmarks: the average improves from 51.95 for MoE and 52.43 for vanilla Engram to 53.86. The largest gains appear on LAMBADA, from 42.98 to 48.19; BoolQ, from 57.92 to 62.54; and WinoGrande, from 58.98 to 60.93. Since the grafting memory is frozen and produced offline, and the recipient keeps matched trainable parameters and activated parameters, the result demonstrate the effectiveness of memory grafting.

Table 3 fixes a 0.92B trainable / 0.29B activated recipient trained for 50B tokens and varies only the grafting model. All grafting variants improve average accuracy over the baselines, but the gains are uneven: MG-GLM gives only a small improvement, from 45.62 for MoE to 45.83, whereas MG-Deepseek and MG-Qwen3.5 reach 46.42 and 46.98, respectively. We also observe that vanilla Engram underperforms the MoE baseline in this 1B-scale setting, with 45.03 versus 45.62. These may caused by the limited Engram embedding table in 0.9B model, which lead to more hash collisions. This suggests that Memory Grafting benefits from external latent memory, but the quality of the grafting model's n-gram representations matters. Appendix B.1 analyzes the memory geometry and shows that GLM-4.7-Flash memory is much more anisotropic and less discriminative, which helps explain why its downstream gain is limited.

4.3 Further Analysis

4.3.1 Impact of the Grafting-Model Source Layer

Beyond the choice of grafting model, we find that the grafting-model layer used to construct the frozen memory is also important. In this analysis, we keep the recipient model, training data, memory keys, and recipient-side grafting module fixed, and vary only the source layer from which the grafting-model n-gram embeddings are extracted. The loss trend shows that memory built from an intermediate grafting-model layer, especially around Layer 6 in the compared setting, gives the strongest benefit. When the memory embeddings are extracted from deeper grafting-model layers, the loss can instead increase.

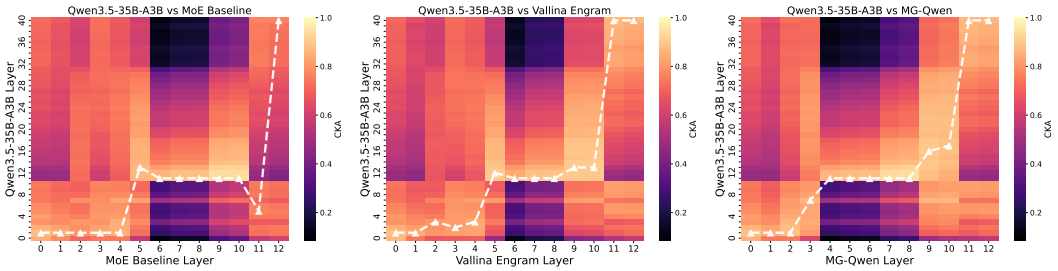

Figure 3: CKA comparison between Qwen3.5-35B-A3B and the MoE baseline, vanilla Engram, and Memory Grafting model. The white trajectory marks the best-matching recipient layer for each Qwen3.5-35B-A3B layer.

This result suggests that Memory Grafting requires a compatible match between the representation level of the external memory and the recipient model's own hidden states. Earlier or middle-layer grafting-model embeddings are more likely to encode local lexical and phrase-level n-gram information that a small recipient model can align with. Deeper grafting-model embeddings may be more abstract and more tightly coupled to the internal computation of the large grafting model. For a smaller recipient, these high-level n-gram representations can be harder to interpret through a lightweight projection and gate, making the injected signal less useful or even harmful. Therefore, the benefit of Memory Grafting depends not only on whether external memory is available, but also on choosing a grafting-model source layer whose representations the recipient can effectively understand.

4.4 CKA Analysis of Memory Grafting

We use centered kernel alignment (CKA) [7] to analyze how Memory Grafting changes the recipient representation space, as shown in Figure 3. The heatmaps compare Qwen3.5-35B-A3B layers with the layers of the MoE baseline, vanilla Engram, and Memory Grafting model, and the white trajectory marks the best-matching recipient layer for each grafting-model layer. Compared with the two baselines, Memory Grafting shows a clearer alignment with deeper Qwen3.5-35B-A3B layers at shallower recipient layers. In other words, after memory grafting, the recipient begins to exhibit functions similar to the deep Qwen representations earlier in its own layer stack, whereas the MoE baseline and vanilla Engram show weaker or later alignment. This indicates that the grafted memory leaves a measurable signature in the recipient hidden states, which supports the intended mechanism of Memory Grafting. Specifically, the recipient-side projection and gate allow the recipient model to effectively incorporate feature patterns injected from the grafting model, thereby aligning its internal representations with the grafted memory.

4.5 Ablation of Grafting Mechanism

We compare four memory grafting mechanisms in Figure 4: directly grafting multiple matched n-gram embeddings through attention, adding a gate after this attention aggregation, grafting only the longest matched n-gram through a gate, and further adding the Engram fallback to the gated longest-match mechanism. The two attention-based variants improve over vanilla Engram, but their gains are limited. Selecting only the longest matched n-gram and grafting it through a gate is more effective because the selected memory is more specific and easier to regulate.

The best result is obtained by further adding the Engram fallback to this gated longest-match grafting mechanism. This variant consistently achieves the largest loss reduction relative to vanilla Engram. Since the frozen grafting memory is built only for high-frequency n-grams, it cannot cover all tokens through exact matching. Using only this memory may therefore introduce a hit-token bias during training: tokens with matched high-frequency contexts receive grafting signals, while unmatched or low-frequency contexts receive no external memory. The Engram fallback mitigates this coverage bias by providing a trainable memory path for missed tokens and by retaining additional low-frequency n-gram memories. Therefore, the final Memory Grafting mechanism combines adaptive gating, efficient longest-match grafting, and engram fallback mechanism.

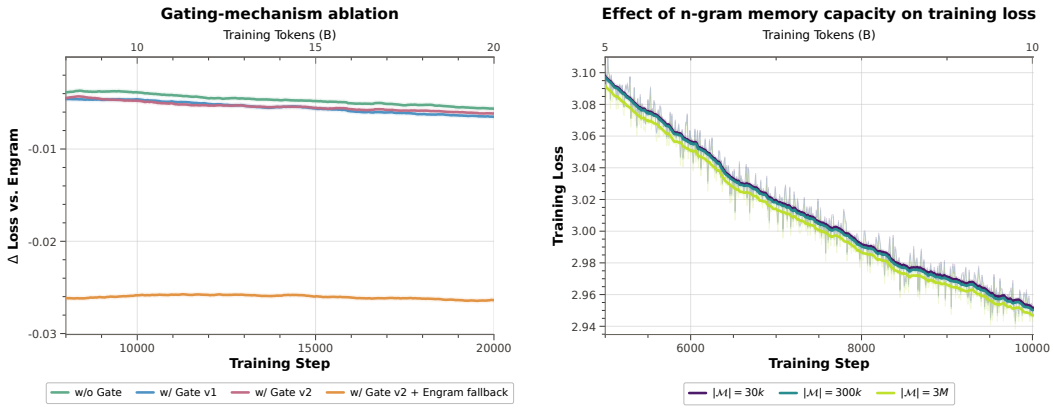

Figure 4: Ablation of the Memory Grafting design and the grafted n-gram memory capacity.

4.6 Impact of Grafted Memory Capacity

For exploring the scaling potential of memory grafting, We compare the training loss across different grafted memory size. As shown in the right panel of Figure 4, larger memory capacity consistently leads to lower training loss. Increasing the memory size from 30k to 300k and further to 3M improves the loss curve throughout training, indicating a clear scaling trend: grafting more memory provides more useful exact-match contexts and improves the recipient model more strongly. This result demonstrate that the benefit of Memory Grafting can be scaled with the size of the grafted memory.

4.7 Efficiency Analysis

We report the efficiency comparison in Figure 5. Memory Grafting keeps the runtime cost close to the MoE baseline, reaching $0.92\times$ training throughput and $0.95\times$ inference throughput, while using $0.97\times$ inference memory. Although Memory Grafting introduces an external memory bank that is several times larger than the recipient model, the resulting slowdown is limited. The main reason is that exact n-gram retrieval is implemented as a hash lookup, whose expected complexity is $\mathcal{O}(1)$ with respect to the memory-bank size. The expensive grafting-model computation is also performed offline, so the online path only adds lookup, projection, gating, and the memory update. The offline construction cost is also modest in our setting: extracting a 3M n-gram memory from Qwen3.5-35B-A3B requires only about 3 A100 GPU-hours. Therefore, the result suggests that Memory Grafting can scale to substantially larger memory banks while adding only modest offline construction cost and runtime overhead.

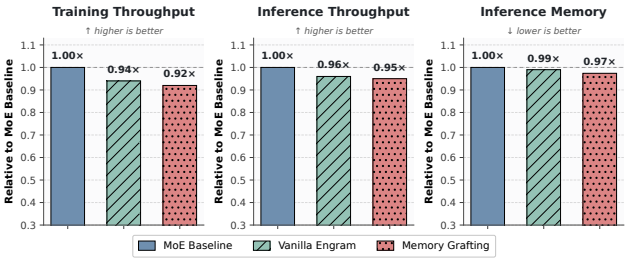

Figure 5: Throughput and memory usage comparison. The model of memory grafting is 0.92B trainable parameters with approximate 12B frozen grafting memory offload.

5 Conclusion

In this paper, we introduced *Memory Grafting*, a conditional memory scaling method that turns frozen grafting-model hidden states into conditional n-gram memory for a smaller recipient model. The key idea is to move the expensive grafting-model computation offline: the recipient performs exact longest-match retrieval over the frozen memory bank, projects and gates the retrieved latent representation, and falls back to trainable Engram memory when no exact match is available. Under matched pre-training settings, this design improves over both MoE and vanilla Engram baselines, and the gains are stronger when the grafted memory comes from more discriminative grafting models

such as Qwen3.5-35B-A3B and DeepSeek-V2-Lite. The analysis further shows that the retrieved memories are actively used by the recipient and that increasing memory capacity improves training loss. Since exact memory retrieval has expected $\mathcal{O}(1)$ complexity with respect to memory-bank size, Memory Grafting mainly expands external latent capacity rather than per-token activated computation, enabling efficient training and inference. We therefore view it as a practical step toward scaling future language models through reusable external latent memory beyond trainable parameters alone.

Acknowledgments and Disclosure of Funding

Funding and computational resources. This work was supported by the SSTIC Grants (KJZD20230923115106012, KJZD20230923114916032, GJHZ20240218113604008). We thank Microsoft Research Asia for providing the computational resources.

Competing interests. The authors declare no other competing interests.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

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

[3] 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.

[4] 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, 2019.

[5] 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.

[6] Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Yu Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1280–1297, 2024.

[7] MohammadReza Davari, Stefan Horoi, Amine Natik, Guillaume Lajoie, Guy Wolf, and Eugene Belilovsky. Reliability of cka as a similarity measure in deep learning. arXiv preprint arXiv:2210.16156, 2022.

[8] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

[9] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural networks, 107:3–11, 2018.

[10] 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):1–39, 2022.

[11] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024.

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, Haofen Wang, et al. Retrieval-augmented generation for large language models: A survey. arXiv preprint arXiv:2312.10997, 2(1):32, 2023.

[13] Team Glm, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. arXiv preprint arXiv:2406.12793, 2024.

[14] Sachin Goyal, David Lopez-Paz, and Kartik Ahuja. Distilled pretraining: A modern lens of data, in-context learning and test-time scaling. arXiv preprint arXiv:2509.01649, 2025.
[15] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.
[16] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In International conference on machine learning, pages 3929–3938. PMLR, 2020.
[17] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. arXiv preprint arXiv:2203.15556, 10, 2022.
[18] Zihao Huang, Qiyang Min, Hongzhi Huang, Defa Zhu, Yutao Zeng, Ran Guo, and Xun Zhou. Ultra-sparse memory network. arXiv preprint arXiv:2411.12364, 2024.
[19] Shayekh Bin Islam, Md Asib Rahman, KSM Tozammel Hossain, Enamul Hoque, Shafiq Joty, and Md Rizwan Parvez. Open-rag: Enhanced retrieval augmented reasoning with open-source large language models. In Findings of the Association for Computational Linguistics: EMNLP 2024, pages 14231–14244, 2024.
[20] Wenqi Jiang, Marco Zeller, Roger Waleffe, Torsten Hoefler, and Gustavo Alonso. Chameleon: a heterogeneous and disaggregated accelerator system for retrieval-augmented language models. arXiv preprint arXiv:2310.09949, 2023.
[21] Shibo Jie, Yehui Tang, Kai Han, Yitong Li, Duyu Tang, Zhi-Hong Deng, and Yunhe Wang. Mixture of lookup experts. arXiv preprint arXiv:2503.15798, 2025.
[22] 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.
[23] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. In 1995 international conference on acoustics, speech, and signal processing, volume 1, pages 181–184. IEEE, 1995.
[24] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. In Proceedings of the 2017 conference on empirical methods in natural language processing, pages 785–794, 2017.
[25] Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. Advances in Neural Information Processing Systems, 32, 2019.
[26] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in neural information processing systems, 33:9459–9474, 2020.
[27] Jianghao Lin, Rong Shan, Chenxu Zhu, Kounianhua Du, Bo Chen, Shigang Quan, Ruiming Tang, Yong Yu, and Weinan Zhang. Rella: Retrieval-enhanced large language models for lifelong sequential behavior comprehension in recommendation. In Proceedings of the ACM Web Conference 2024, pages 3497–3508, 2024.
[28] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437, 2024.
[29] Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. arXiv preprint arXiv:2401.17377, 2024.
[30] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101, 2017.
[31] 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, 2016.
[32] Hao Peng, Xin Lv, Yushi Bai, Zijun Yao, Jiajie Zhang, Lei Hou, and Juanzi Li. Pre-training distillation for large language models: A design space exploration. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3603–3618, 2025.
[33] Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026.

[34] 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.
[35] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. Communications of the ACM, 64(9):99–106, 2021.
[36] Claude Elwood Shannon. A mathematical theory of communication. The Bell system technical journal, 27(3):379–423, 1948.
[37] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538, 2017.
[38] Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 2459–2475, 2025.
[39] Gemma Team. Gemma 3n. 2025.
[40] Gemma Team. Gemma 4. 2026.
[41] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2023.
[42] Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. arXiv preprint arXiv:2507.20534, 2025.
[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.
[44] 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, 2019.
[45] Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. arXiv preprint arXiv:2508.06471, 2025.
[46] Biao Zhang and Rico Sennrich. Root mean square layer normalization. Advances in neural information processing systems, 32, 2019.
[47] Zhihao Zhang, Alan Zhu, Lijie Yang, Yihua Xu, Lanting Li, Phitchaya Mangpo Phothilimthana, and Zhihao Jia. Accelerating retrieval-augmented language model serving with speculation. arXiv preprint arXiv:2401.14021, 2024.

Appendix

Contents

A Experiment Details **14**

A.1 Model Architecture and Hyper Parameters 14

A.2 Construction of Grafting-Model Memory 16

A.3 Calculate resources and environment 16

A.4 Appendix: Details of the Grafting Gate 17

B Additional Experiment **18**

B.1 Additional Analysis of Grafting-Model Memory Geometry 18

B.2 Loss Curves for Memory Grafting 19

B.3 More CKA Analysis Results of Memory Grafting 20

B.4 Single-Layer Memory Grafting Results 21

B.5 Probe Analysis 22

C Limitations & Discussion **23**

D LLM Usage **23**

A Experiment Details

A.1 Model Architecture and Hyper Parameters

Table 3: Architecture and training hyper-parameters for the 0.92B-class models.

	MoE-1B	**Engram-1B**	**MG-1B**
Total Params	0.92B	0.92B	0.92B
Active Params		0.29B	
Total Tokens		50B	
Layers		12	
Dimension		768	
MoE Inter. Size		384	
Routed Experts	64	48	46
Active Experts		4	
Shared Experts		1	
Heads		16	
Sequence Length		2048	
Vocab Size		128256	
Training Steps		50000	
Optimizer		AdamW	
Base LR		5e-4	
LR Scheduler		Cosine w/ warmup	
Weight Decay		0.1	
Engram Dim d_{mem}	-	384	384
Engram N-gram	-	[2,3]	[2,3] †
Engram Num Head	-	8	8
Engram Vocab Size	-	112865	118265
Engram Layer (target)	-	[1, 6]	[1, 6]
Engram ShortConv Kernel	-	4	4
Engram Weight Decay	-	0.1	0.1
Engram Optimizer	-	AdamW [30]	AdamW
Memory Grafting Model	-	-	Qwen3.5-35B-A3B
Memory Grafting Source Layers	-	-	[8, 24]
Memory Grafting Recipient Layers	-	-	[1, 12]
Memory Grafting Entries per Layer	-	-	3,000,000
Memory Grafting Hidden Dim	-	-	2048
Memory Grafting Dtype	-	-	bfloat16
Memory Grafting Size per Layer	-	-	$\approx$12 GB
Memory Grafting Total Frozen Memory	-	-	$\approx$24 GB
Memory Grafting Trainable	-	-	False (frozen)
Memory Grafting Retrieval	-	-	Exact longest-match (2/3/4-gram)
Memory Grafting Fallback	-	-	Hash Engram (2/3-gram)

Table 4: Architecture and training hyper-parameters for the 2.8B-class models.

	MoE-2.8B	**Engram-2.8B**	**MG-2.8B**
Total Params	2.8B	2.8B	2.8B
Active Params		0.55B	
Total Tokens		100B	
Layers		24	
Dimension		1024	
MoE Inter. Size		512	
Routed Experts	64	48	47
Active Experts		4	
Shared Experts		1	
Heads		16	
RoPE θ		10000	
Sequence Length		2048	
Vocab Size		128256	
Training Steps		50000	
Optimizer		AdamW	
Base LR		5e-4	
LR Scheduler		Cosine w/ warmup	
Weight Decay		0.1	
Engram Dim d_{mem}	-	512	512
Engram N-gram	-	[2,3]	[2,3]
Engram Num Head	-	8	8
Engram Vocab Size	-	292680	292680
Engram Layer (target)	-	[1, 12]	[1, 12]
Engram ShortConv Kernel	-	4	4
Engram Weight Decay	-	0.1	0.1
Engram Optimizer	-	AdamW	AdamW
Memory Grafting Model	-	-	Qwen3.5-35B-A3B
Memory Grafting Source Layers	-	-	[8, 24]
Memory Grafting Recipient Layers	-	-	[1, 12]
Memory Grafting Entries per Layer	-	-	3,000,000
Memory Grafting Hidden Dim	-	-	2048
Memory Grafting Dtype	-	-	bfloat16
Memory Grafting Size per Layer	-	-	$\approx$12 GB ($\approx$6.1B params)
Memory Grafting Total Frozen Memory	-	-	$\approx$24 GB ($\approx$12.3B params)
Memory Grafting Trainable	-	-	False (frozen)
Memory Grafting Retrieval	-	-	Exact longest-match (2/3/4-gram)
Memory Grafting Fallback	-	-	Hash Engram (2/3-gram)

A.2 Construction of Grafting-Model Memory

We construct the frozen grafting-model memory from frequent local n-grams before recipient training. The input table stores the surface text of each selected n-gram, together with the recipient-side lookup key used to index the memory table. Following the notation in Section 3.2, the selected memory keys are

$$\mathcal{M} = \{\mathbf{m}_i\}_{i=1}^{M}, \qquad \mathbf{m}_i = (x_{i,1}, \ldots, x_{i,n_i}), \qquad n_i \in \{2, 3, 4\}, \tag{15}$$

where each $\mathbf{m}_i$ is the exact lookup key used by the recipient model. The implementation stores a map from this key to the memory row index i.

To build the memory values, we run the grafting model offline on the surface text of each n-gram. In the Qwen3.5-35B-A3B implementation, we tokenize the n-gram text with the Qwen tokenizer, run the text backbone, and extract the hidden state at the last non-padding token from selected grafting-model layers. These vectors are written to disk and then packed into frozen `nn.Embedding` tables in bfloat16 format. The same construction also saves a map from the recipient-side n-gram id to the row index of the stored vector. At runtime, the recipient only performs exact lookup over $\mathcal{M}$, obtains the corresponding frozen vector, and uses learned projections and gates to adapt it to the recipient hidden space. Equivalently, for a selected grafting-model layer r, the stored value is

$$\mathbf{W}_{\mathrm{mem}}[i] = F_G(\mathrm{text}(\mathbf{m}_i))_{\mathrm{last}}^{(r)} \in \mathbb{R}^{D_{\mathrm{mem}}}, \tag{16}$$

which is the implementation-level form of $F_G(\mathbf{m}_i)_{\mathrm{last}}^{(r)}$ in Section 3.2. During recipient training and inference, the recipient queries this frozen table with the current suffix context:

$$(j_{b,t}, m_{b,t}) = \texttt{ExactLookup}(\mathbf{X}_{b,\le t}, \mathcal{M}), \qquad \mathbf{e}_{\mathrm{mem},b,t} = \mathbf{W}_{\mathrm{mem}}[j_{b,t}] \quad \text{if } m_{b,t} = 1. \tag{17}$$

This design avoids conflicts when the recipient model and the grafting model use different tokenizers. Online retrieval is always performed over the recipient-side key set $\mathcal{M}$, while the grafting-model tokenizer is used only offline to encode the same surface text and produce the stored latent value. Even if the grafting model splits the text differently, this only changes the latent vector stored in $\mathbf{W}_{\mathrm{mem}}$; it does not change the online lookup key. To make this explicit, let τ_G denote the grafting-model tokenizer and let $s_i = \mathrm{text}(\mathbf{m}_i)$. Then

$$\mathbf{W}_{\mathrm{mem}}[i] = F_G(\tau_G(s_i))_{\mathrm{last}}^{(r)}, \tag{18}$$

but online retrieval only queries $\mathcal{M}$ through $\mathbf{m}_i$. Thus, $\tau_G(s_i)$ affects only the stored latent value, not the exact lookup key.

A.3 Calculate resources and environment

We use deepspeed as the training framework. For the 0.9B model, We conduct training on a cluster with 4 nodes and 32 A100 GPUs. For the 2.8B model, We conduct training on a cluster with 8 nodes and 64 A100 GPUs.

A.4 Appendix: Details of the Grafting Gate

We provide the detailed formulation of the grafting mechanisms used in the ablation using the notation in Section 3. At recipient position (b, t) and branch c, let $\mathbf{H}^c_{b,t}$ denote the recipient hidden state. For an exact matched n-gram order n, let $\mathbf{e}^{(n)}_{\mathrm{mem},b,t}$ denote the corresponding frozen grafting-model memory feature, and let $\mathcal{N}_{b,t}$ denote the set of matched orders. The hash-based Engram fallback feature is $\mathbf{e}_{\mathrm{hash},b,t}$. Given any candidate key-value pair $(\mathbf{K}^c_{b,t}, \mathbf{V}_{b,t})$, the Method gate is

$$\mathbf{Q}^c_{b,t} = \mathrm{RMSNorm}_q(\mathbf{H}^c_{b,t}), \qquad \bar{\mathbf{K}}^c_{b,t} = \mathrm{RMSNorm}_k(\mathbf{K}^c_{b,t}),$$

$$\alpha^c_{b,t} = \sigma\left(\frac{\left\langle \bar{\mathbf{K}}^c_{b,t}, \mathbf{Q}^c_{b,t} \right\rangle}{\sqrt{D}}\right), \qquad \mathbf{U}^c_{b,t} = \alpha^c_{b,t}\mathbf{V}_{b,t},$$

$$\Delta\mathbf{H}^c_{b,t} = \mathbf{U}^c_{b,t} + \texttt{ShortConv}(\mathbf{U})^c_{b,t}, \qquad \widetilde{\mathbf{H}}^c_{b,t} = \mathbf{H}^c_{b,t} + \Delta\mathbf{H}^c_{b,t}.$$

① The attention-only grafting variant first aggregates all matched n-gram embeddings and adds the resulting value without the query-key gate:

$$\mathbf{U}^c_{\mathrm{attn},b,t} = \mathbf{e}_{\mathrm{attn},b,t} = \texttt{Attn}\left(\mathbf{H}^c_{b,t}, \{\mathbf{U}^c_{\mathrm{attn},b,t} = \mathbf{e}^{(n)}_{\mathrm{mem},b,t} : n \in \mathcal{N}_{b,t}\}\right)$$

$$\Delta\mathbf{H}^c_{b,t} = \mathbf{U}^c_{\mathrm{attn},b,t} + \texttt{ShortConv}(\mathbf{U}_{\mathrm{attn}})^c_{b,t}.$$

② The attention-gated variant uses the same attention-based memory but modulates it with the gate:

$$\mathbf{K}^c_{\mathrm{attn},b,t} = \mathbf{W}^{\mathrm{mem},c}_k \mathbf{e}_{\mathrm{attn},b,t}, \qquad \mathbf{V}_{\mathrm{attn},b,t} = \mathbf{W}^{\mathrm{mem}}_v \mathbf{e}_{\mathrm{attn},b,t},$$

followed by the query-key gate above with $(\mathbf{K}^c_{b,t}, \mathbf{V}_{b,t}) = (\mathbf{K}^c_{\mathrm{attn},b,t}, \mathbf{V}_{\mathrm{attn},b,t})$.

③ The longest-match gated variant removes the attention aggregation and uses only the longest exact match. For this ablation, missed positions do not use Engram fallback; if no exact match is found, the lookup returns zero memory:

$$(j_{b,t}, m_{b,t}) = \texttt{ExactLookup}(\mathbf{X}_{b,\le t}, \mathcal{M}), \qquad m_{b,t} = 0 \quad \text{if no exact memory hit is found.}$$

When $m_{b,t} = 1$, let $n^*_{b,t} = \max \mathcal{N}_{b,t}$. The candidate memory feature is then

$$\mathbf{e}_{\mathrm{long},b,t} = \begin{cases} \mathbf{e}^{(n^*_{b,t})}_{\mathrm{mem},b,t}, & m_{b,t} = 1, \\ \mathbf{0}, & m_{b,t} = 0, \end{cases}$$

and the projected key and value are

$$\mathbf{K}^c_{\mathrm{long},b,t} = \mathbf{W}^{\mathrm{mem},c}_k \mathbf{e}_{\mathrm{long},b,t}, \qquad \mathbf{V}_{\mathrm{long},b,t} = \mathbf{W}^{\mathrm{mem}}_v \mathbf{e}_{\mathrm{long},b,t},$$

followed by the same query-key gate with $(\mathbf{K}^c_{b,t}, \mathbf{V}_{b,t}) = (\mathbf{K}^c_{\mathrm{long},b,t}, \mathbf{V}_{\mathrm{long},b,t})$.

④ The final mechanism adds Engram fallback to the gated longest-match mechanism:

$$\mathbf{K}^c_{\mathrm{eng},b,t} = \mathbf{W}^{\mathrm{eng},c}_k \mathbf{e}_{\mathrm{hash},b,t}, \qquad \mathbf{V}_{\mathrm{eng},b,t} = \mathbf{W}^{\mathrm{eng}}_v \mathbf{e}_{\mathrm{hash},b,t},$$

$$\mathbf{K}^c_{b,t} = m_{b,t}\mathbf{K}^c_{\mathrm{mem},b,t} + (1 - m_{b,t})\mathbf{K}^c_{\mathrm{eng},b,t}, \qquad \mathbf{V}_{b,t} = m_{b,t}\mathbf{V}_{\mathrm{mem},b,t} + (1 - m_{b,t})\mathbf{V}_{\mathrm{eng},b,t},$$

where $m_{b,t}$ is the exact-hit mask defined by $\texttt{ExactLookup}(\mathbf{X}_{b,\le t}, \mathcal{M})$. The selected $(\mathbf{K}^c_{b,t}, \mathbf{V}_{b,t})$ is then passed to the Method gate. This formulation makes the role of the gate explicit: it does not decide which memory is retrieved, but controls how strongly the selected grafting signal is written into the recipient hidden state through $\Delta\mathbf{H}$.

B Additional Experiment

B.1 Additional Analysis of Grafting-Model Memory Geometry

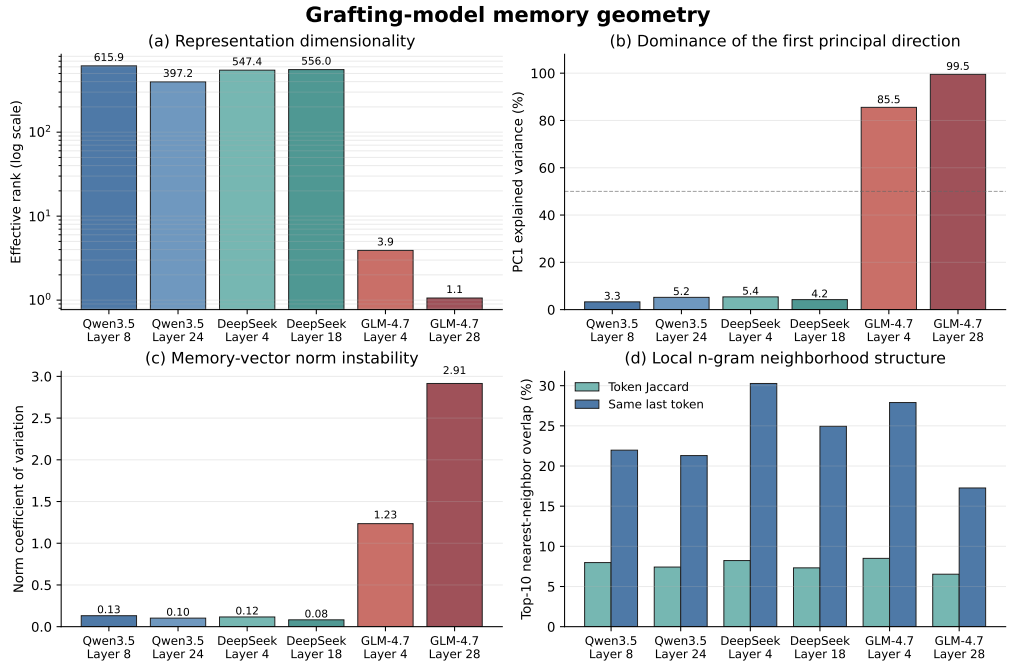

Figure 6: Geometry of frozen grafting-model memory tables. We compare effective rank, first-principal-component dominance, norm instability, and local nearest-neighbor structure across memory tables constructed from Qwen3.5-35B-A3B, DeepSeek-V2-Lite, and GLM-4.7-Flash.

Figure 6 compares frozen memory tables constructed from the same n-gram key set but different grafting models: Qwen3.5-35B-A3B Layers 8/24, DeepSeek-V2-Lite Layers 4/18, and GLM-4.7-Flash Layers 4/28. For each table, we sample 10,000 memory rows for norm statistics, 2,048 rows for effective rank and first-principal-component variance, and 1,024 rows for cosine nearest-neighbor structure. All diagnostics are computed directly on the stored memory vectors before recipient training.

Qwen3.5-35B-A3B and DeepSeek-V2-Lite produce high-dimensional and stable memory geometry: their effective ranks remain in the hundreds, PC1 explains only a small fraction of the variance, and norm variation is low. In contrast, GLM-4.7-Flash is strongly anisotropic: Layer 4 already has low effective rank, and Layer 28 is nearly dominated by a single direction with unstable norms. Thus, many GLM n-grams retrieve nearly collinear vectors, making the memory less discriminative. This suggests that Memory Grafting requires not only a capable grafting model, but also structured local n-gram representations; when this structure is weak, the improvement from grafting is limited.

B.2 Loss Curves for Memory Grafting

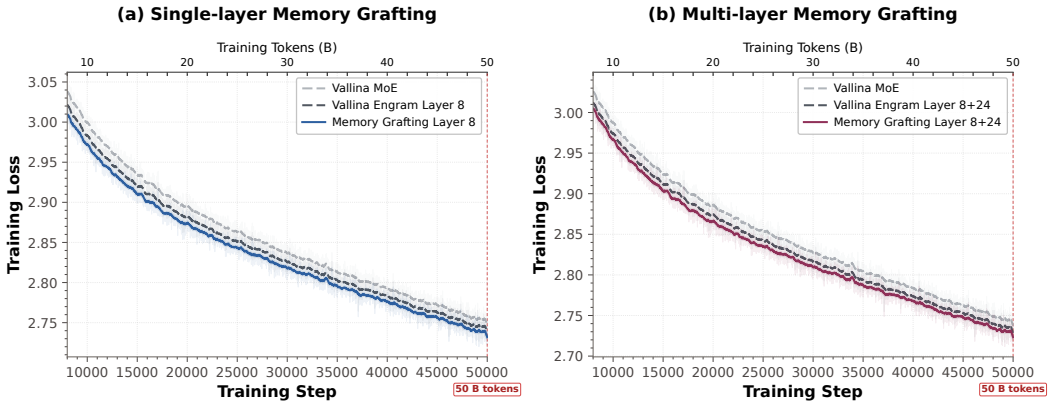

Figure 7: Training loss curves for memory graftings.

We further report the training loss curves of single-layer and multi-layer grafting settings in Figure 7. The results show that Memory Grafting effectively reduces the training loss in both cases.

B.3 More CKA Analysis Results of Memory Grafting

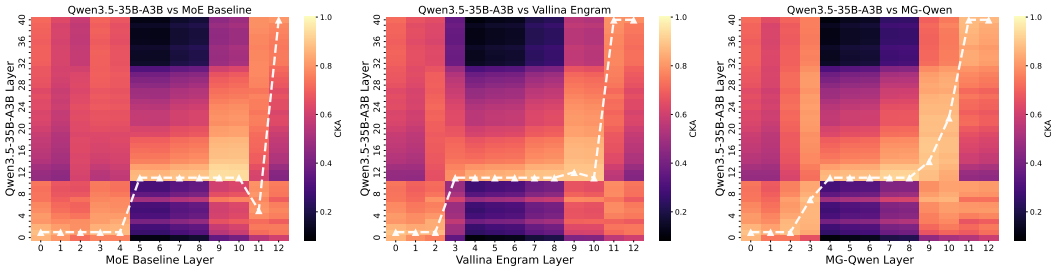

Figure 8: CKA comparison for single-layer Memory Grafting.

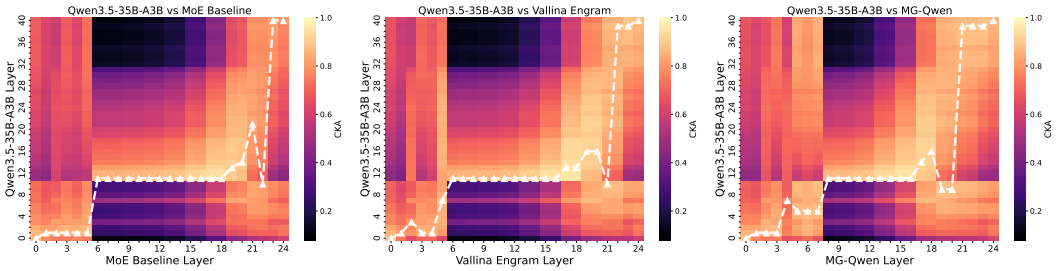

Figure 9: CKA comparison for the 2.8B model.

We provide additional CKA results in Figures 8 and 9. Both the single-layer setting and the 2.8B Memory Grafting model show stronger alignment with deeper Qwen3.5-35B-A3B representations than the corresponding baselines. This further indicates that the grafted memory changes the recipient hidden states rather than remaining an unused external table.

B.4 Single-Layer Memory Grafting Results

Table 5: Benchmark performance (%) in the 0.8B-trainable, 50B-token setting. Only the first layer was grafted by the layer 6 of Qwen3.5-35B-A3B

Benchmark *(Metric)*	**# Shots**	**MoE Baseline**	**Vanilla Engram**	**Memory Grafting**
# Total Params		0.92B	0.92B	0.92B
# Activated *(w/o token embed)*		0.29B	0.29B	0.29B
# Trained Tokens		50B	50B	50B
# Experts *(shared + routed, top-k)*		1 + 56 (top-4)	1 + 48 (top-4)	1 + 46 (top-4)
ARC-Challenge	0-shot	28.24	28.50	**29.27**
ARC-Easy	0-shot	63.30	63.30	**63.51**
BoolQ	0-shot	56.21	48.50	**60.73**
Social IQA	0-shot	40.53	39.92	**40.69**
RACE	0-shot	30.53	29.95	**32.06**
LAMBADA	5-shot	29.50	29.11	**30.37**
WinoGrande	5-shot	51.78	52.09	**52.72**
PIQA	5-shot	70.24	71.22	**71.49**
HellaSwag	10-shot	38.03	38.25	**38.40**
Average		45.37	44.54	**46.58**

We also evaluate Memory Grafting under the single-layer grafting setting. With the same model size and activated computation, Memory Grafting achieves the best average performance among the compared models, improving over both the MoE baseline and vanilla Engram. The gains are consistent across the evaluated benchmarks, showing that even a single grafting layer can effectively transfer useful n-gram memory into the recipient model.

B.5 Probe Analysis

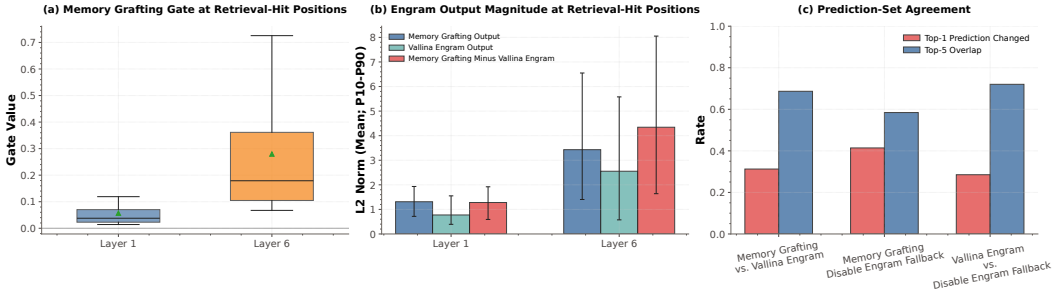

Figure 10: Overview of gate values, output magnitudes, and prediction-set agreement at retrieval-hit positions.

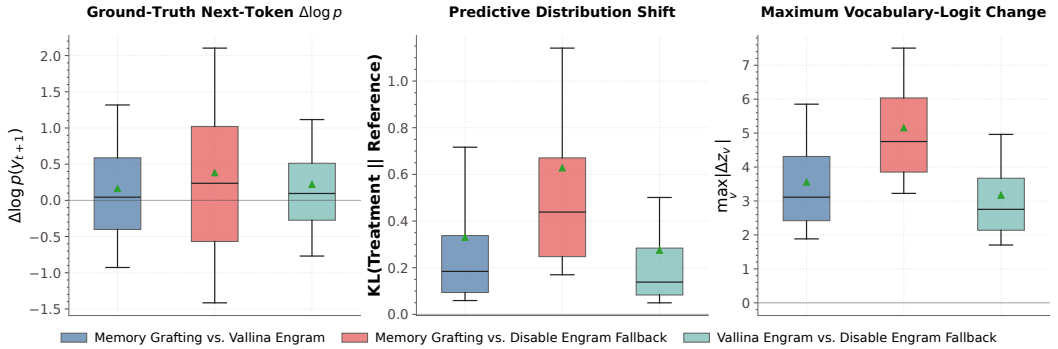

Figure 11: Logit-level impact of grafted memory at retrieval-hit positions.

We separately probe the behavior of the grafted memory at retrieval-hit positions, where an exact n-gram match is found in the external memory. Figure 10 shows that the gate is non-zero and the memory output has a clear magnitude at these positions, meaning that the retrieved memory is actually written into the recipient hidden states. Figure 11 further shows that this hidden-state change propagates to the output distribution: the next-token logits, target token log-probability, and prediction set all change after grafting. Therefore, the grafted memory is not only retrieved, but also participates in the model's computation and directly affects its predictions.

C Limitations & Discussion

We do not include a full comparison with knowledge distillation. A from-scratch distillation baseline at the same pre-training scale would require running a large teacher model over the full training corpus and repeating this process under comparable recipient settings, which would require substantial additional computation and is difficult to complete under our compute budget. Moreover, forcing distillation into the same compute budget would also be difficult to interpret, because the result may reflect insufficient distillation tuning or an under-resourced distillation setup rather than a clean comparison of the transfer mechanism. Such a comparison would therefore not provide a clean evaluation of the effectiveness of Memory Grafting.

To make this gap concrete, For example, distilling a 1B-A0.2B recipient from a 30B-A3B teacher requires one extra teacher forward pass for every training token. Using standard transformer compute accounting[17, 22], the recipient's own training cost is roughly 6×0.2B activated computation per token, while the teacher forward adds 2×3B. Thus, online distillation would add about $5\times$ the recipient's own training compute, making the total cost about $6\times$ solo recipient pre-training.

Therefore, our experiments focus on comparisons under nearly matched recipient architectures, token budgets, and activated computation. Under these controlled conditions, Memory Grafting serves as a knowledge-transfer method independent of knowledge distillation, showing that reusable latent memory can improve pre-training without running a teacher model online.

D LLM Usage

This study utilizes Large Language Models (LLMs) to refine content, adjust formatting, construct tables, and provide writing suggestions for specific chapters.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction state three claims that are directly supported by the paper.

   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: We discuss limitations in Appendix.

   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: The paper does not contain formal theorems requiring proofs.

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: We provide the full algorithmic specification and all training hyper-parameters in Appendix.

   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: Code and pretrained checkpoints are not released with this anonymous submission, but the training data and tokenizer are public and the appendix lists all architectural and hyper-parameter details; we will release a clean implementation upon acceptance.

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: The appendix specifies the training settings, all chosen following common practice for Transformer language models.

   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: [No]

   Justification: Pretraining Language models at this scale makes multi-seed runs prohibitively expensive, so we report train identical setting with one time.

   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: The appendix reports the computation resources.

   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: This is methodological research on memory grafting using only publicly released text data and a public tokenizer, with no human subjects or sensitive data, and the submission is anonymized in line with the NeurIPS Code of Ethics.

   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: [Yes]

    Justification: By providing a highly cost-effective and modular pathway to scale model capacity, Memory Grafting lowers the compute barrier and democratizes advanced AI research for broader communities.

    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: The paper releases no new dataset, and the model are trained by public dataset.

    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: All external assets used in the paper – the pretraining corpus, tokenizer, evaluation harness, and benchmark datasets – are credited to their original creators and used within their license terms.

    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: The submission introduces no new datasets.

    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: The paper involves no crowdsourcing or human subjects; all experiments use publicly available text corpora and benchmarks.

    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: The paper involves no human subjects, so IRB approval is not applicable.

    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: [Yes]

    Justification: We made the declaration, and we use LLM is used only for polishing the paper.

    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.