Frozen Memory Is Not Enough: Rethinking External Memory as Extraction

Mingyuan Li
ELLIS Institute Finland
University of Turku
mingyl@utu.fi

Guangsheng Yu
University of Technology Sydney
guangsheng.yu@uts.edu.au

Xu Wang
University of Technology Sydney
xu.wang@uts.edu.au

Shaoxiong Ji*
ELLIS Institute Finland
University of Turku
shaoxiong.ji@utu.fi

Abstract

Methods for improving knowledge use in large language models typically fall into two regimes. Non-parametric retrieval offers flexible access to external knowledge, but adds retrieval latency, context overhead, and only shallow integration with the backbone. Parametric adaptation is efficient at inference time, but entangles knowledge with model weights and can be hard to update, audit, or transfer. Engram-style hashed memory occupies a middle regime: it stores learned information in an external, addressable table, yet consumes that table through a small learned reader. This raises a basic question: when such a memory is moved across backbones, what matters more, the frozen memory itself or the target-side reader? We study this question through cross-model frozen-memory extraction, in which a memory trained on a source model is frozen and attached to a different target model, with only a lightweight reader trained. Ablations show that learned memory content and correct addressing both matter, while the amount of usable signal depends on how well the reader is aligned to the target model. In downstream question answering tasks, a dual-layer, four-branch reader nearly closes the gap between same-model and cross-model reuse, achieving an average score of 38.8 under our controlled evaluation protocol. Moreover, when the provider reader is directly compatible with the target interface, the frozen artifact can provide substantial utility without target-side training, while optional reader adaptation yields further improvement. These results suggest that Engram can serve as a reusable external knowledge artifact, provided that the target has access to a compatible reader interface; target-side adaptation can further improve alignment when direct reader reuse is insufficient.

1 Introduction

Language models can use knowledge in two familiar ways. *Non-parametric* methods, such as retrieval-augmented generation (RAG) [Lewis et al., 2020] and nearest-neighbor retrieval [Khandelwal et al., 2020], keep knowledge outside the backbone and make updates relatively direct, but they pay for that flexibility with retrieval latency, longer contexts, and loose integration between retrieved evidence and generation. *Parametric* methods, such as continued pretraining, fine-tuning, or learned memory layers tied to a single model [Wei et al., 2026], avoid retrieval at inference time, but they entangle knowledge with model weights. This makes updates, audits, and transfers across backbones difficult.

*Corresponding author.

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

Engram-style hashed memory [Cheng et al., 2026] occupies an interesting middle ground. It stores learned information in an explicit, addressable external table, yet consumes that table through a small neural interface rather than by inserting raw documents into the context. This hybrid design is attractive because it combines the modularity of external memory with the efficiency of learned representations. But it also exposes a sharper question: **is the memory itself reusable or transferable, or is it merely a co-adapted extension of the model that trained it?**

The distinction matters: a memory table that is not portable is just another form of model-specific parameterization, not a reusable knowledge substrate. Thus, the portability requires an operational test: *remove the source backbone and ask whether another model can still extract a useful signal from the frozen memory table.*

We study this test through *cross-model frozen-memory transfer*. Given a memory table trained with a source model A, we freeze the table, attach it to a different target model B, and train only a lightweight target-side reader for memory extraction. If the frozen memory still improves the target after the original host is removed, then it contains usable information beyond source-specific co-adaptation. If it fails, then the apparent memory benefit was likely tied to the original backbone.

Figure 1 illustrates the life cycle of memory reuse and transfer: source-side memory training, memory export and freezing, and target-side extraction through a new reader. To make this test precise, we separate three roles: (1) *addressing* decides which entries are read, (2) *memory* denotes the stored vectors, and (3) *reader* maps them into the model. We standardize addressing, freeze the memory, and adapt only the reader. This turns portability into a measurable question: can a target model learn to read a fixed memory artifact? We use a tokenizer-agnostic canonicalization pipeline to keep memory addresses stable across tokenizers, freeze the exported source memory, and train only a small target-side reader that projects memory values into the target residual stream.

Our central claim is: **for portable hashed external memory, memory presence alone is not enough; successful reuse depends on the interface through which a target backbone addresses and consumes the stored representations.** Nevertheless, memory content still matters. Transferred memory clearly outperforms permuted-key controls and reaches substantially better downstream Question Answering (QA) performance than random memory under the final evaluation protocol. But the amount of usable signal depends on how well the target model can align and integrate what it retrieves. This view explains why imperfect memory can still help with a capable reader, and why reader design can matter as much as retraining or enlarging the stored table in this regime.

We support this claim with a concise chain of evidence, spanning diverse model families and settings. First, a source–target transfer study shows that frozen memory consistently improves target models across different model families and scales, including Pythia [Biderman et al., 2023], Qwen Qwen Team [2025], TinyLlama [Zhang et al., 2024], Phi Abouelenin et al. [2025], LLaMA [Touvron et al., 2023], and Mistral [Jiang et al., 2023], with relative perplexity reductions of up to 15.7%. A supplementary peer-to-peer setting further confirms that transfer remains beneficial in both directions between models of similar scale. Second, ablation experiments show that transferred memory clearly outperforms the permuted-key and no-memory controls, while scratch-trained target memory remains competitive under the same target-side adaptation budget. This suggests that memory reuse is a joint property of the stored content and the target side reader that makes it usable. Third, on downstream tasks, upgrading the target-side reader to a dual-layer, multi-branch architecture nearly closes the gap between same-model and cross-model reuse, with the best configuration reaching an average score of 38.8 across five QA tasks under our controlled evaluation protocol, while broader knowledge-intensive benchmarks show selective gains.

In summary, our contributions are:

- We formulate cross-model frozen-memory reuse as an evaluation problem for external memory, separating portable memory from backbone-specific co-adaptation and measuring whether the target model can extract usable signal from a frozen artifact.
- We instantiate this problem with a transfer protocol for Engram-style memory that standardizes addressing across tokenizers, and we propose lightweight target-side readers for memory extraction, including stronger multi-branch, multi-layer variants.
- We provide an evidence chain showing that reader design is first-order: consistent gains across diverse model families and settings, ablations that isolate memory content and address

integrity, and controlled QA results showing that stronger target-side readers, rather than source-backbone identity alone, drive the best performance.

This paper is organized as follows. Section 2 reviews Engram architecture, related memory mechanisms, and cross-model alignment. Section 3 describes the cross-model frozen-memory transfer protocol, including tokenizer-agnostic addressing, target-side reader design, and training regime. Section 4 presents the experimental setup, research questions, results, and takeaways. Supplementary material in Appendix A provides additional details on canonicalization, memory export, and the reader implementation. Appendix B gives training and evaluation details, including hyperparameters, datasets, and evaluation metrics. Appendix C extends RQ1 in Section 4.1 with exact matrix values and controls for tokenizer mismatch, peer-scale transfer, target scaling, and representational similarity. Appendix D extends RQ3 in Section 4.3 by diagnosing why downstream gains are task-dependent, separating source-corpus specialization, Phase-2 reader-fitting alignment, mixed-corpus behavior, and out-of-domain effects. The supplementary material concludes with a computational cost analysis (Appendix F), broader ethics and impact considerations (Appendix G), and limitations (Appendix H).

2 Background and Related Work

2.1 Engram Architecture

Engram [Cheng et al., 2026] augments a Transformer-MoE backbone with an external conditional memory. It retrieves static n-gram embeddings through deterministic hashing and injects the retrieved vector into the backbone through a learned gate. The key property for our setting is that Engram separates *addressing*, *storage*, and *read*: the input text determines which memory entry is read, the memory table stores the external vectors, and a small learned reader maps the retrieved vector into the backbone hidden space. This separation makes transfer possible: as long as another model can reproduce the same memory addresses through a shared addressing scheme, the frozen Engram table can be reused through a target-side reader.

For each n-gram order $n \in [2, N]$ and hash head $k \in [1, K]$, a deterministic hash $\varphi_{n,k}$ maps the canonicalized n-gram $g_{t,n}$ to an entry in memory table $\mathbf{E}_{n,k}$. The retrieved embeddings are concatenated as

$$\mathbf{e}_t \triangleq \bigoplus_{n=2}^{N} \bigoplus_{k=1}^{K} \mathbf{E}_{n,k}[\varphi_{n,k}(g_{t,n})]. \tag{1}$$

Crucially, this lookup depends on the canonicalized input sequence rather than the model hidden state, so the memory table can be treated as an external artifact once the address space is standardized.

The retrieved vector is consumed through learned projections and a gate:

$$\begin{aligned} s_t &= \frac{\mathrm{RMSNorm}(\mathbf{h}_t)^\top \mathrm{RMSNorm}(\mathbf{k}_t)}{\sqrt{d}}, \quad \alpha_t = \sigma\Big(\mathrm{sign}(s_t)\sqrt{|s_t|}\Big), \\ \mathbf{k}_t &= \mathbf{W}_K \mathbf{e}_t, \qquad\qquad\qquad\qquad \mathbf{v}_t = \mathbf{W}_V \mathbf{e}_t. \end{aligned} \tag{2}$$

Here, $\mathbf{W}_K$, $\mathbf{W}_V$, and the gate define the backbone-specific reader. This structural separation makes frozen-memory transfer possible: the source memory table can be frozen and moved to a target model, while only a lightweight target-side reader is trained to extract and align it.

2.2 Memory Mechanisms

MoE [Shazeer et al., 2017, Fedus et al., 2022, Dai et al., 2024] expands capacity through conditional computation, but its useful signal remains embedded in the host backbone rather than exposed as a reusable memory artifact. Other memory-augmented models also miss the extraction question for one of two reasons: they either retrieve from large dynamic memory banks at inference time [Graves et al., 2014, Weston et al., 2015], or they store information in model-specific hidden spaces, as in KNN-LM [Khandelwal et al., 2020]. Retrieval-augmented generation and RETRO-style systems [Lewis et al., 2020, Borgeaud et al., 2022, Izacard et al., 2023] are portable at the document level, but the reusable object is a text datastore rather than a trained memory representation. Larger learned memory layers [Lample et al., 2019, Berges et al., 2025, Behrouz and Zhong, 2025, Wei et al., 2026] improve capacity or efficient knowledge access, but their stored representations are usually trained

and consumed within the same model. Engram differs in the regime relevant here: deterministic hashing provides an explicit address space, the memory table is external and addressable, and a small learned reader mediates read by the backbone. This separation makes a frozen memory artifact well defined and allows portability to be tested as extraction rather than as additional model capacity. A broader qualitative comparison appears in Table 4.

2.3 Cross-Model Alignment

Cross-model transfer is often studied as representation or adapter alignment. Embedding-space work suggests that independently trained models can sometimes be connected by simple linear maps between hidden representations [Mikolov et al., 2013, Kornblith et al., 2019, Huh et al., 2024, Chen et al., 2025]. PEFT transfer methods such as Cross-LoRA [Xia et al., 2025], LoRA-X [Farhadzadeh et al., 2025], and Trans-LoRA [Wang et al., 2024] show that compact adapters or low-rank updates can sometimes be reused across heterogeneous models. These methods are useful baselines for parameter transfer, but they do not directly address frozen external-memory reuse: there is no separate memory table to address, freeze, and extract from after the source backbone is removed. In frozen-memory extraction, alignment starts earlier. The target model must first read the same memory entry as the source-intended lookup, even when their tokenizers differ. Otherwise, the same surface text may be routed to unrelated memory slots, and any downstream hidden-space mapping becomes ill-defined. Our method, therefore, treats alignment as a two-stage problem: tokenizer-agnostic canonicalization aligns the key space, and a lightweight target-side reader aligns the retrieved value space to the target residual stream.

3 Methods

Our method recasts Engram as a transferable artifact. We freeze the source memory and fit only a small target-side reader. This requires a shared address space, frozen memory, and a lightweight reader. Relative to native Engram, we make three changes: we (1) freeze the exported source memory, (2) replace model-specific token-ID lookup with tokenizer-agnostic canonicalization, and (3) use a lightweight target-side reader attached by residual injection to read the knowledge from Engram memory. Figure 1 illustrates the overall architecture and training and adaptation procedures, and Algorithm 1 summarizes the training procedure.

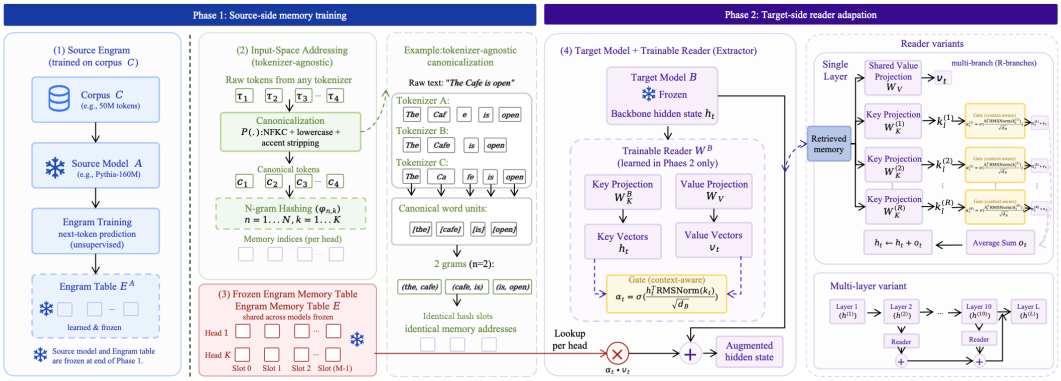


Figure 1: Overview of the cross-model memory transfer. A memory table trained with source model A is frozen and attached to target model B. Canonicalization keeps the address space fixed across tokenizers, and only the target-side reader (including multi-head and multi-layer variants) is trained. This setup operationalizes whether external memory remains reusable outside its source backbone.

3.1 Tokenizer-Agnostic Addressing

The first obstacle to transfer is the address mismatch. Native Engram hashes model-specific token IDs inside the host model, which is sufficient when storage and read happen under one tokenizer, but does not define a stable cross-model address space. If source and target tokenizers segment the same

text differently, then identical surface content may be routed to different memory entries. To avoid this failure mode, we build the lookup space on canonicalized decoded text rather than raw token IDs. Let $\mathcal{T}_B$ be the tokenizer of target model B, and let $s_{1:t} = \mathrm{decode}(\mathcal{T}_B(x_{1:t}))$ denote the decoded target prefix up to position t. We normalize that prefix with a shared canonicalization function $\mathcal{P}$ and extract the completed word sequence

$$(w_1, \ldots, w_{m(t)}) = \mathrm{words}(\mathcal{P}(s_{1:t}))\,, \tag{3}$$

where $\mathrm{words}(\cdot)$ returns the canonicalized word-boundary units implied by the current decoded prefix. Word-boundary addressing is the default for the English-like settings above, but it degenerates for scripts without reliable whitespace segmentation. For such inputs, we replace word units with NFKC-normalized, case-folded non-whitespace Unicode character events and hash character spans using the same deterministic addressing interface. Appendix E evaluates this variant on Chinese and Japanese. In practice, $\mathcal{P}$ applies NFKC normalization, lowercasing, accent stripping, and whitespace cleanup. Memory lookup is then performed over canonical word-boundary n-grams rather than model-specific token IDs:

$$\mathbf{e}_t = \bigoplus_{n=2}^{N} \bigoplus_{j=1}^{K} \mathbf{E}_{n,j}\left[\varphi_{n,j}(w_{m(t)-n+1}, \ldots, w_{m(t)})\right]. \tag{4}$$

At subword positions inside the same decoded word, the current token reuses the same right-edge word context until a new word boundary appears. The retrieved vector $\mathbf{e}_t$ is therefore determined by canonicalized text and the frozen memory artifact, not by target-specific token IDs. Two target models with different tokenizers can still read the same transferred entry as long as they share the same canonicalization and hash specification.

Algorithm 1: Cross-Model Frozen-Memory Extraction

Input: Source memory $\mathcal{E}_A$, target model B, target corpus $\mathcal{D}$, selected injection layers $\mathcal{L}$, branch count R, canonicalization $\mathcal{P}$

Output: Target model B augmented with frozen transferred memory

`// // Phase 1 artifact is already trained; prepare transfer`

Extract Engram tables $\{\mathbf{E}_{n,j}\}$ and hash specification from $\mathcal{E}_A$;

Freeze all transferred tables $\{\mathbf{E}_{n,j}\}$ *[portable memory artifact]*;

`// // Initialize target-side reader and hooks`

foreach $\ell \in \mathcal{L}$ **do**

  Initialize reader parameters $\mathcal{W}^{(B,\ell)}$;

  Set $\mathcal{W}^{(B,\ell)} \leftarrow \{\mathbf{W}^{(B)}_{K,\ell,r}, \beta_{\ell,r}\}_{r=1}^{R}, \mathbf{W}^{(B)}_{V,\ell}$;

  if $R = 1$ **then**

    Set $\beta_{\ell,1} \leftarrow 0$;

  Register a forward hook at layer ℓ that computes $\mathbf{o}_{t,\ell}$ from the current $\mathbf{h}_{t,\ell}$ and looked-up $\mathbf{e}_t$, then injects $\mathbf{h}_{t,\ell} \leftarrow \mathbf{h}_{t,\ell} + \mathbf{o}_{t,\ell}$;

`// // Phase 2: Reader-only adaptation`

for $x_{1:T} \in \mathcal{D}$ **do**

  Canonicalize each decoded prefix into shared word-boundary n-grams with $\mathcal{P}$;

  Look up frozen memory vectors $\mathbf{e}_{1:T}$ from $\{\mathbf{E}_{n,j}\}$;

  Run the frozen backbone on $x_{1:T}$ with the registered hooks active;

  Update only reader parameters $\{\mathcal{W}^{(B,\ell)}\}_{\ell\in\mathcal{L}}$ using the LM loss *[backbone and memory frozen]*;

3.2 Target-Side Reader

Once the address space is shared, the remaining problem is extraction: how should the target model consume a memory vector whose geometry was shaped by a different backbone? We use a shared-value reader family parameterized by branch count R, where a larger R increases target-side alignment capacity without increasing memory size. For branch $r \in \{1, \ldots, R\}$ at layer ℓ:

$$\mathbf{k}^{(r)}_{t,\ell} = \mathbf{W}^{(B)}_{K,\ell,r}\mathbf{e}_t, \qquad \mathbf{v}_{t,\ell} = \mathbf{W}^{(B)}_{V,\ell}\mathbf{e}_t, \tag{5}$$

$$\alpha^{(r)}_{t,\ell} = \sigma\left(\frac{\mathrm{RMSNorm}(\mathbf{h}_{t,\ell})^\top \mathrm{RMSNorm}(\mathbf{k}^{(r)}_{t,\ell})}{\sqrt{d_B}} + \beta_{\ell,r}\right), \tag{6}$$

$$\mathbf{o}_{t,\ell} = \frac{1}{R} \sum_{r=1}^{R} \alpha_{t,\ell}^{(r)} \cdot \mathbf{v}_{t,\ell}. \tag{7}$$

The resulting reader output is injected through a residual connection:

$$\mathbf{h}_{t,\ell} \leftarrow \mathbf{h}_{t,\ell} + \mathbf{o}_{t,\ell}. \tag{8}$$

The shared value projection forces all branches to operate on the same retrieved content and differ only in *how* they gate and align it. The added capacity is therefore an extraction mechanism rather than extra storage. The main 3×3 transfer matrix uses $R = 1$; the stronger QA reader uses $R = 4$.

Multi-layer injection. The same reader family can be instantiated at one or more target layers $\mathcal{L}$. Single-layer injection is sufficient for the cross-architecture transfer matrix, whereas the best QA configuration uses dual-layer injection at layers 2 and 10 with $R = 4$ branches.

3.3 Training Regime

The training objective is the standard next-token language modeling loss on the target corpus. During target-side adaptation, both the exported source memory tables $\{\mathbf{E}_{n,j}\}$ and the target backbone parameters θ_B remain frozen; only the reader parameters $\{\mathcal{W}^{(B,\ell)}\}_{\ell \in \mathcal{L}}$ are updated. Storage is fixed, and only extraction is learned. Algorithm 1 summarizes the procedure concisely. The key operations are to freeze the exported source memory, canonicalize target inputs into the shared address space, and update only the target-side reader.

Deployment modes. The two-phase protocol above describes the general case in which the target requires a model-specific reader. We additionally distinguish between two deployment modes. In *direct artifact reuse*, a provider exports the frozen memory together with a reader whose residual interface is compatible with the consumer model; the consumer performs no target-side optimization. In *reader-adapted reuse*, the same frozen memory is retained, but a lightweight target-specific reader is fitted on an available adaptation stream. The first mode minimizes consumer-side cost, while the second provides additional alignment capacity when direct reader reuse is insufficient. Both modes preserve the defining property of the protocol: the transferred memory table itself is never rewritten.

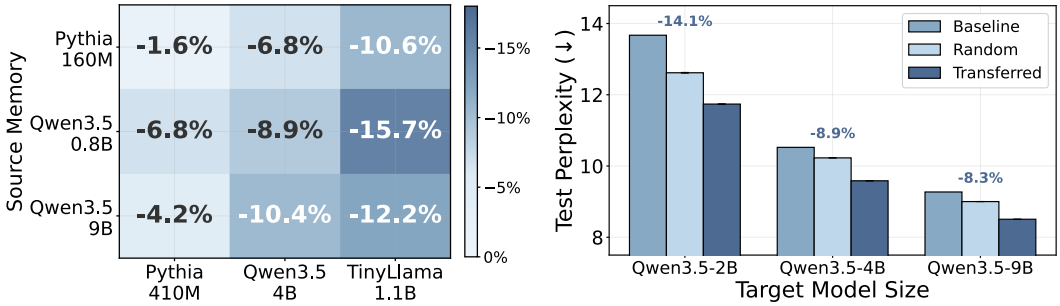
4 Experiments and Results

In this section, we address the effect of cross-model memory transfer through six research questions. All experiments follow the same protocol: a source memory is trained (Phase 1), then frozen and attached to a target model where only a reader is trained (Phase 2) as illustrated in Figure 1. We evaluate on intrinsic language modeling and downstream QA tasks. Intrinsic evaluations use the memory transfer architecture with a minimal reader, while QA evaluation uses a stronger multi-layer, multi-branch reader. Implementation details are in Appendix B.

4.1 RQ1: Does frozen memory transfer across backbones, tokenizers, and scales?

Our first question is whether the exported memory remains useful once detached from the source backbone. We start with a 3×3 source–target matrix spanning three source memories (Pythia-160M [Biderman et al., 2023], Qwen3.5-0.8B, Qwen3.5-9B [Qwen Team, 2025]) and three target models (Pythia-410M, Qwen3.5-4B, TinyLlama-1.1B [Zhang et al., 2024]), all trained with the same 20M-token target-side protocol. Appendix C therefore stress-tests the same conclusion against several alternative explanations. It reports the exact matrix values, introduces parameter-matched and retrieval-based comparisons, removes the small-to-large transfer asymmetry through peer-to-peer transfer, tests whether gains persist as the target scales, and examines whether representational similarity predicts transfer success.

Figure 2 summarizes this result together with the scaling trend. All nine cells in the main matrix in Figure 2a show PPL reduction in models with transferred memory over the no-memory baseline, and the same transferred Qwen3.5-0.8B memory continues to help as the target grows from 2B to 9B (Figure 2b), compared with the baseline without memory and the random memory control with a randomly initialized, untrained memory of the same architecture and size as the transferred memory.



(a) Full 3×3 source–target transfer matrix, reported as relative PPL reduction over the no-memory baseline.

(b) Target-scale intrinsic transfer: the same transferred Qwen3.5-0.8B memory improves Qwen3.5 targets from 2B to 9B.

Figure 2: The full 3x3 transfer matrix and target-scale transfer performance across different model types trained on Wikitext-103.

The full matrix in Figure 2a gives a positive answer in every one of the nine cells. Transferred memory improves over the no-memory baseline in all source–target pairs, with relative gains from 1.6% to 15.7%. The strongest gain appears in the Qwen3.5-0.8B $\rightarrow$ TinyLlama-1.1B case, showing that large improvements do not require the source and target to share an architecture or a tokenizer. The same cell also gives a 13.9% reduction with random memory, so the 15.7% headline should not be interpreted as a 15.7% content-specific gain; the transferred table adds a smaller but positive margin over the random control.

This is also not only an intrinsic-perplexity effect: the downstream evaluation later shows selective downstream gains for Qwen3.5 targets from 0.8B to 9B, and Table 13 shows that the same target family retains that pattern under two different source-corpus families. On Qwen3.5-2B specifically, Figure 5 shows the same asymmetry in a more focused form: transferred memory stays ahead of scratch on intrinsic PPL across all three budgets (Figure 5b), while the downstream gains are selective rather than universal, with the clearest best-of-budget improvements appearing on BoolQ, RTE, and SciQ (Figure 5a).

One subtle pattern is that source scaling helps, but not monotonically, for every target. Qwen3.5-0.8B improves over Pythia-160M on all three targets, yet Qwen3.5-9B does not always beat Qwen3.5-0.8B on the most dissimilar target (Pythia-410M), suggesting that transfer depends not only on what the source memory stores, but also on what the target reader can successfully extract. This underscores the importance of the reader's extraction capability, as supplementary results in Appendix C confirm that transfer is effective in peer-to-peer settings (Appendix C.3) and even between backbones with low representational similarity (Appendix C.5).

Takeaway. Frozen memory transfer remains effective across backbone families, tokenizer boundaries, and target scales, with gains in all nine main source–target pairs; the same Qwen3.5 target family also retains selective downstream gains beyond perplexity, and successful transfer ultimately depends on whether the target reader can extract the stored structure.

4.2 RQ2: How much does reader design matter?

The previous experiments established that memory transfer remains effective even across model families. However, they do not explain where the transfer gains actually come from. One possibility is that most of the benefit is already contained in the transferred memory itself. An alternative explanation is that transfer quality is primarily limited by the target-side reader, i.e., the mechanism responsible for extracting and integrating information from the frozen memory.

To distinguish between these possibilities, we keep the transferred memory fixed and vary only the reader architecture. Specifically, we study two aspects of reader design: (i) memory placement, namely the layers at which memory is injected into the target backbone, and (ii) retrieval capacity, controlled through the number of reader branches. Our central finding is that reader design is a

first-order factor: most of the gain comes from placing the reader at the right layers, while multiple branches further improve performance by accessing the same memory from different perspectives.

Table 1: QA accuracy (%) on Mistral-7B-v0.3 for different reader variants and baselines. Unless noted otherwise, Engram and baseline rows report F1; the MLP Memory row reproduces the source paper's reported Exact Match and is included only as a non-comparable reference. Budget notation follows Table 2 trained on Wikipedia-2021 [Izacard et al., 2023].

Variant	Budget	NQ	WebQA	TriviaQA	TruthQA	HotpotQA	Average
Base	–	20.6	29.3	57.7	32.1	21.0	32.1
Non-parametric Methods							
RAG [Lewis et al., 2020]	–	22.6 1.9	24.9 -4.4	54.2 -3.4	**35.5** 3.4	**29.8** 8.8	33.4(+3.9%)
kNN [Khandelwal et al., 2020]	–	21.1 0.4	30.5 1.2	57.8 0.1	32.3 0.2	21.2 0.2	32.6(+1.4%)
Parametric Methods							
CPT	–	12.2 -8.5	34.1 4.8	61.2 3.6	29.2 -2.9	16.0 -4.9	30.5(-5.0%)
LoRA [Hu et al., 2022]	–	18.2 -2.5	34.5 5.2	61.6 4.0	30.9 -1.2	16.2 -4.7	32.3(+0.5%)
MLP Memory [Wei et al., 2026]	–	25.2 4.6	**37.5** 8.2	61.0 3.3	32.5 0.5	24.1 3.2	36.1(+12.3%)
Engram Variants (sorted by Average)							
LLaMA Frozen $\{10\}$-R1	10/0	21.8 1.2	29.6 0.3	60.6 2.9	33.1 1.1	22.3 1.4	33.5(+4.4%)
Mistral Frozen $\{10\}$-R1	10/0	21.9 1.2	29.7 0.4	60.8 3.1	32.9 0.8	22.4 1.4	33.5(+4.4%)
Mistral $\{10\}$-R1	10/20	22.1 1.5	31.6 2.4	61.1 3.4	32.8 0.7	22.3 1.3	34.0(+5.8%)
LLaMA $\{10\}$-R1	10/20	22.2 1.5	32.6 3.3	61.2 3.6	32.9 0.8	22.3 1.3	34.2(+6.6%)
LLaMA $\{2, 10\}$-R1	10/20	30.3 9.7	28.7 -0.6	**70.1** 12.4	31.0 -1.1	27.4 6.4	37.5(+16.7%)
LLaMA $\{2, 10\}$-R4	10/20	**30.3** 9.7	33.7 4.4	69.9 12.3	30.9 -1.2	27.6 6.7	**38.5**(+19.9%)

The first pattern is that a weak or frozen reader improves only a small fraction of the performance. With a single injection layer at $\{10\}$, the frozen variants achieve only 33.5–33.5 average accuracy, a modest improvement over the 32.1 base model. Training the same single-layer reader raises performance to 34.0–34.2, indicating that target-side adaptation already contributes noticeably even before increasing reader capacity. This suggests that the transferred memory alone is insufficient; the target model must learn how to access it effectively.

The largest gain comes from reader placement. Moving from LLaMA $\{10\}$-R1 to LLaMA $\{2, 10\}$-R1 increases average accuracy from 34.2 to 37.5. The improvement is broad rather than dataset-specific: NQ rises from 22.20 to 30.3, TriviaQA from 61.2 to 70.1, and HotpotQA from 22.3 to 27.4. These results suggest that dual-layer injection provides the target model with multiple opportunities to integrate retrieved memory throughout the forward pass, instead of forcing all memory interaction through a single late-layer interface.

Reader capacity explains the remaining improvement. Keeping the injection layers fixed at $\{2, 10\}$ and increasing the number of branches from $R = 1$ to $R = 4$ raises average accuracy from 37.5 to 38.5. The most visible gain appears on WebQA (28.7 to 33.7), while the strong improvements on NQ, TriviaQA, and HotpotQA are preserved. This behavior is consistent with the intended role of the multi-branch design: different branches provide complementary key-gating pathways over the same shared value representation. The branches, therefore, increase the flexibility with which the target conditions the contribution of a retrieved memory vector, rather than introducing independent stored value directions.

Reader structure is not the only capacity factor. We additionally vary the frozen-memory interface width in a controlled Pythia-160M→Pythia-410M diagnostic. Increasing the memory width from 256 to 512 and 1024 reduces test perplexity from 22.368 to 22.157 and 21.808, respectively. However, a width-512 reader injected at two layers reaches 21.522, outperforming the wider width-1024 single-layer configuration. Thus, the default 512-dimensional interface is a design choice rather than a hard architectural bottleneck, and extraction capacity depends jointly on interface width and reader structure. Because changing the width also changes both memory and reader parameter counts, we treat this experiment as a capacity-sensitivity analysis rather than a causal estimate of memory width alone.

The same table also places reader improvements in context relative to existing approaches. The strongest transfer configuration, LLaMA $\{2, 10\}$-R4, achieves 38.5 average accuracy, outperform-

ing the included like-for-like baselines, including RAG, kNN-LM, CPT, and LoRA. Notably, this improvement is obtained without modifying the transferred memory itself, reinforcing the conclusion that extraction quality, rather than memory storage, is the dominant factor governing transfer performance.

Takeaway. Reader design is a first-order determinant of transfer quality. Dual-layer injection recovers most of the available gain, while multi-branch gating provides an additional improvement by allowing the same retrieved value representation to be conditioned through multiple target-dependent key-gate pathways. Together, these design choices raise average QA accuracy from 34.2 to 38.5 without changing the transferred memory itself.

4.3 RQ3: When does transferred memory help downstream?

Table 2: QA accuracy (%) on Mistral-7B-v0.3 for transfer verification and ablation studies. Deltas are measured against the baseline. Budget = P1/P2 denotes source-memory and target-reader training tokens (millions) trained on Wikipedia-2021.

Variant	Budget	NQ	WebQA	TriviaQA	TruthQA	HotpotQA	Average
Base	–	20.6	29.3	57.7	32.1	21.0	32.1
Transfer Verification							
Mistral $\{2, 10\}$-R4	10/20	30.2 9.5	**34.0** 4.7	70.0 12.3	30.8 -1.3	27.7 6.8	**38.5**(+20.0%)
LLaMA $\{2, 10\}$-R4	10/20	30.3 9.7	33.7 4.4	69.9 12.3	30.9 -1.2	27.6 6.7	38.5(+20.0%)
Frozen Mistral $\{2, 10\}$-R4	10/0	30.5 9.8	30.3 1.0	**70.8** 13.2	32.2 0.2	27.7 6.8	38.3(+19.2%)
Frozen LLaMA $\{2, 10\}$-R4	10/0	**30.6** 10.0	30.3 1.0	70.8 13.1	32.2 0.1	27.6 6.7	38.3(+19.2%)
Controls and Ablations							
Mem-only Mistral	10/0	0.2 -20.4	0.3 -29.0	1.4 -56.3	33.0 0.9	0.2 -20.7	7.0(-78.1%)
Mem-only LLaMA	10/0	0.0 -20.6	0.0 -29.3	0.0 -57.7	–	0.0 -21.0	–
Permuted	10/0	20.6 0.0	28.9 -0.4	63.1 5.4	32.4 0.3	23.1 2.2	33.6(+4.7%)
FFN R4	10/20	19.6 -1.0	27.3 -2.0	58.1 0.5	32.1 -0.0	17.6 -3.4	30.9(-3.7%)
Token Scaling (LLaMA2-7B source, $\{2, 10\}$-R4)							
Engram	10/10	30.2 9.5	29.9 0.6	70.4 12.8	31.2 -0.9	27.6 6.7	37.9(+17.8%)
Engram	15/15	30.2 9.6	32.7 3.4	70.1 12.4	30.6 -1.5	27.8 6.8	38.3(+19.2%)
Engram	20/20	30.3 9.7	33.9 4.6	69.9 12.3	30.7 -1.4	27.6 6.7	38.5(+19.8%)
Engram	25/25	30.5 9.9	33.1 3.8	70.0 12.3	30.9 -1.2	27.7 6.7	38.4(+19.6%)
Engram	30/30	30.6 10.0	33.9 4.6	70.4 12.7	31.1 -1.0	**27.9** 7.0	**38.8**(+20.7%)

RQ1–RQ2 established that memory can be transferred across model families and that transfer quality is largely determined by the target-side reader. The remaining question is not whether transfer works, but rather how far it can be pushed and where it begins to break down.

We study this question from two complementary perspectives. First, we use QA benchmarks to estimate the upper bound of transfer performance under increasingly strong reader configurations. Second, we evaluate whether these gains generalize beyond QA to broader knowledge-intensive tasks. Throughout this section, $\{a, b\}$-Rr denotes memory injection at layers $\{a, b\}$ with r retrieval branches, and "Frozen" indicates evaluation with the source reader without target-side adaptation.

Table 2 reveals a clear and surprisingly high transfer ceiling. Under the strongest configuration ($\{2, 10\}$-R4), same-model and cross-model transfer achieve nearly identical performance (38.5 vs. 38.5 average accuracy), achieving a ceiling once the reader is sufficiently expressive. Performance further saturates with scale: increasing training budgets from 10/10 to 30/30 improves accuracy from 37.9 to 38.8, suggesting diminishing returns in the high-38 regime.

Control results confirm that this ceiling is not driven by parameter addition alone. Memory-only variants collapse without a reader, and FFN-based substitutes fall below the base model, while permuting memory keys removes most gains. These results indicate that successful transfer critically depends on the correct retrieval and integration of frozen memory rather than additional capacity. The frozen-reader rows additionally expose a second deployment regime. When the provider reader is directly compatible with the target residual interface, the complete frozen artifact can be reused without target-side training: mean QA remains approximately 38.3 compared with 32.1 for the

no-memory target. Optional target-reader fitting further raises the score to approximately 38.5. Thus, target-side adaptation is useful but is not an inherent requirement for every compatible source–target pair. We next examine whether these gains extend beyond QA to a broader set of downstream tasks.

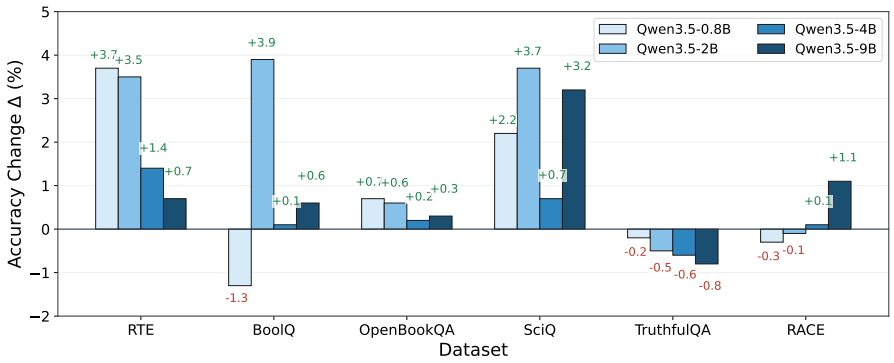


Figure 3: Downstream task evaluation. Bars report the change in accuracy (Δ) produced by transferred memory relative to the no-memory baseline across six downstream tasks. Results are averaged over three seeds (42, 137, and 2024). Positive values indicate that transferred memory improves the target model after target-side adaptation on FineWeb-Edu (FW-9B) [Lozhkov et al.].

Figure 3 shows that transfer helps selectively beyond QA. Gains are most consistent on factual or evidence-oriented tasks: RTE improves at every target scale, SciQ remains positive, OpenBookQA is smaller but positive, and BoolQ becomes positive for targets of 2B parameters and above. RACE stays near zero, suggesting limited benefit when factual retrieval is not the primary bottleneck. TruthfulQA is the negative boundary, with decreases at every scale, consistent with its emphasis on calibrated truthfulness rather than factual association recovery.

Overall, RQ3 shows a high but task-dependent transfer ceiling. The frozen memory helps when downstream decisions can exploit additional factual associations, but it is neutral or mildly harmful when success depends on resisting misleading associations. Appendix D reports the source-corpus and reader-fitting diagnostics.

Interpretation. This pattern identifies a practical boundary for frozen-memory extraction. The memory transfers factual associations that the target reader can route into a decision, but it does not replace task-specific calibration or long-context reasoning. The negative TruthfulQA result is therefore informative: portability of the artifact does not imply portability of every behavior supported by the source model. Reader design and evaluation alignment remain necessary for reliable downstream use.

Takeaway. Transfer improves downstream performance selectively: it is most useful for factual and evidence-oriented tasks, while broader reading comprehension and truthfulness remain challenging.

5 Conclusion

We study whether a learned external memory remains useful after it is detached from the backbone that trained it and attached to a different one. For Engram-style hashed memory, the answer is yes: frozen memory improves the target across tokenizers, hidden sizes, and architectures, including every cell of the main 3×3 transfer matrix and both directions of peer transfer. The main lesson is that portability depends not only on the frozen table but also on the target-side reader: stronger readers nearly close the same-model/cross-model gap on QA, and transferred memory clearly beats permuted-key and FFN-only controls while substantially outperforming random memory on the final QA evaluation. More broadly, cross-model frozen-memory extraction turns external-memory portability into an evaluation framework: a memory artifact should be judged by whether a new backbone can address it, align to it, and extract useful signal without retraining the stored table.

Acknowledgments and Disclosure of Funding

The authors wish to acknowledge CSC – IT Center for Science, Finland, for computational resources. Computational facilities were provided by the UTS eResearch High Performance Computer Cluster. Shaoxiong Ji gratefully acknowledges the support of Foundation PS through the PS Fellowship.

References

A. Abouelenin, A. Ashfaq, A. Atkinson, H. Awadalla, N. Bach, J. Bao, A. Benhaim, M. Cai, V. Chaudhary, C. Chen, et al. Phi-4-mini technical report: Compact yet powerful multimodal language models via mixture-of-loras. *arXiv preprint arXiv:2503.01743*, 2025. URL https://arxiv.org/pdf/2503.01743.

A. Behrouz and P. Zhong. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2025. URL https://arxiv.org/abs/2501.00663.

J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1533–1544, 2013.

V.-P. Berges, B. Oguz, D. HAZIZA, W. tau Yih, L. Zettlemoyer, and G. Ghosh. Memory layers at scale. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ATqGm1WyDj.

S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff, A. Skowron, L. Sutawika, and O. van der Wal. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2304.01373.

S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. van den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, D. de Las Casas, A. Guy, J. Menick, R. Ring, T. Hennigan, S. Huang, L. Maggiore, C. Jones, A. Cassirer, A. Brock, M. Paganini, G. Irving, O. Vinyals, S. Osindero, K. Simonyan, J. W. Rae, E. Elsen, and L. Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, pages 2206–2240, 2022. URL https://arxiv.org/abs/2112.04426.

A. Chen, J. Merullo, A. Stolfo, and E. Pavlick. Transferring linear features across language models with model stitching, 2025. URL https://arxiv.org/abs/2506.06609.

X. Cheng, W. Zeng, D. Dai, Q. Chen, B. Wang, Z. Xie, K. Huang, X. Yu, Z. Hao, Y. Li, H. Zhang, H. Zhang, D. Zhao, and W. Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*, 2026. URL https://arxiv.org/abs/2601.07372.

C. Clark, K. Lee, M.-W. Chang, T. Kwiatkowski, M. Collins, and K. Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings of NAACL-HLT*, pages 2924–2936, 2019.

I. Dagan, O. Glickman, and B. Magnini. The PASCAL recognising textual entailment challenge. In J. Quiñonero-Candela, I. Dagan, B. Magnini, and F. d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33428-6.

D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. K. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024. URL https://arxiv.org/abs/2401.06066.

F. Farhadzadeh, D. Das, S. Borse, and F. Porikli. Lora-x: Bridging foundation models with training-free cross-model adaptation, 2025. URL https://arxiv.org/abs/2501.16559.

W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2106.09685.

M. Huh, B. Cheung, T. Wang, and P. Isola. The platonic representation hypothesis. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2405.07987.

G. Izacard, P. Lewis, M. Lomeli, L. Hosseini, F. Petroni, T. Schick, J. Dwivedi-Yu, A. Joulin, S. Riedel, and E. Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251):1–43, 2023. URL https://jmlr.org/papers/v24/23-0037.html.

A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

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

U. Khandelwal, O. Levy, D. Jurafsky, L. Zettlemoyer, and M. Lewis. Generalization through memorization: Nearest neighbor language models. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1911.00172.

S. Kornblith, M. Norouzi, H. Lee, and G. E. Hinton. Similarity of neural network representations revisited. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019. URL https://arxiv.org/abs/1905.00414.

T. Kwiatkowski, J. Palomaki, O. Redfield, M. Collins, A. Parikh, C. Alberti, D. Epstein, I. Polosukhin, J. Devlin, K. Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

G. Lai, Q. Xie, H. Liu, Y. Yang, and E. Hovy. RACE: Large-scale reading comprehension dataset from examinations. In *Proceedings of EMNLP*, pages 785–794, 2017.

G. Lample, A. Sablayrolles, M. Ranzato, L. Denoyer, and H. Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019. URL https://arxiv.org/abs/1907.05242.

P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020. URL https://arxiv.org/abs/2005.11401.

S. Lin, J. Hilton, and O. Evans. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of ACL*, pages 3214–3252, 2022.

A. Lozhkov, L. B. Allal, L. von Werra, and T. Wolf. Fineweb-edu: the finest collection of educational content, 2024. *URL https://huggingface. co/datasets/HuggingFaceFW/fineweb-edu*.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1609.07843.

T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of EMNLP*, pages 2381–2391, 2018.

T. Mikolov, Q. V. Le, and I. Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013. URL https://arxiv.org/abs/1309.4168.

D. Paperno, G. Kruszewski, A. Lazaridou, N. Q. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1525–1534, 2016. doi: 10.18653/v1/P16-1144.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

S. Sanjeev and A. Troynikov. Embedding adapters. Technical report, Chroma, May 2024. URL https://trychroma.com/research/embedding-adapters. Accessed on July 8, 2024.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1701.06538.

D. Su, K. Kong, Y. Lin, J. Jennings, B. Norick, M. Kliegl, M. Patwary, M. Shoeybi, and B. Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset, 2025. URL https://arxiv.org/abs/2412.02595.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

R. Wang, S. Ghosh, D. Cox, D. Antognini, A. Oliva, R. Feris, and L. Karlinsky. *Trans-LoRA*: towards data-free transferable parameter efficient finetuning, 2024. URL https://arxiv.org/abs/2405.17258.

R. Wei, J. Cao, J. Wang, J. Kai, Q. Guo, B. Zhou, and Z. Lin. Mlp memory: A retriever-pretrained memory for large language models, 2026. URL https://arxiv.org/abs/2508.01832.

J. Welbl, N. F. Liu, and M. Gardner. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106, 2017.

J. Weston, S. Chopra, and A. Bordes. Memory networks, 2015. URL https://arxiv.org/abs/1410.3916.

F. Xia, M. Liao, Y. Fang, D. Li, Y. Xie, W. Li, Y. Li, D. Xia, and J. Huang. Cross-lora: A data-free lora transfer framework across heterogeneous llms, 2025. URL https://arxiv.org/abs/2508.05232.

W. Xie, H. Zhao, W. Liu, Y. Zhu, L. Chen, M. Ye, Z. Chen, Y. Xu, S. Dong, Z. Wang, et al. Step-wise rubric rewards for llm reasoning. *arXiv preprint arXiv:2605.17291*, 2026.

Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

P. Zhang, G. Zeng, T. Wang, and W. Lu. TinyLlama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024. URL https://arxiv.org/abs/2401.02385.

A Supplementary Formalization and Method Details

This section supplies notation and formal details for Section 3.

A.1 Notations

Table 3 summarizes the notation used throughout this paper.

Table 3: Notation used throughout this paper.

Symbol	Meaning
A, B	Source model and target model, respectively.
d_A, d_B	Hidden dimensions of source model A and target model B, respectively.
$\mathcal{E}_A$	Frozen Engram memory artifact learned with source model A, comprising the tables $\{E_{n,k}\}$.
$E_{n,k}$	Memory table associated with N-gram order n and hash head k, where $E_{n,k} \in \mathbb{R}^{M \times d_{\text{head}}}$.
$N_{\max}$	Maximum N-gram order used for memory addressing, with $n \in \{2, \ldots, N_{\max}\}$.
K	Number of independent hash heads for each N-gram order.
H	Total number of hash heads, $H = (N_{\max} - 1)K$.
M	Number of rows in each hash-head memory table.
d_{head}	Dimensionality of the embedding row retrieved from one hash-head table.
$\mathbf{e}_t$	Concatenated memory vector retrieved at token position t.
d_{mem}	Dimensionality of $\mathbf{e}_t$, where $d_{\text{mem}} = H d_{\text{head}}$.
$\mathcal{P}$	Canonicalization function mapping a raw decoded string to its canonical form.
V, V'	Raw vocabulary and canonical vocabulary, respectively.
$\varphi_{n,k}$	Deterministic hash function for N-gram order n and hash head k.
$\mathbf{h}_{t,\ell}$	Target-backbone hidden state at token position t and layer ℓ.
$\mathcal{L}$	Set of target-backbone layers at which memory readers are injected.
S	Number of memory injection sites, $S = \lvert\mathcal{L}\rvert$.
R	Number of reader branches at each injection site.
$\mathbf{W}^{(B)}_{K,\ell,r}$	Branch-specific key-projection matrix for branch r at target layer ℓ.
$\mathbf{W}^{(B)}_{V,\ell}$	Value-projection matrix shared across the R branches at target layer ℓ.
$\alpha^{(r)}_{t,\ell}$	Context-aware scalar gate for token t, layer ℓ, and branch r.
$\beta_{\ell,r}$	Learnable scalar gate bias for branch r at injection layer ℓ.
$\mathcal{W}^{(B)}$	Complete target-side reader parameter set, including all key and value projections, normalization parameters, and gate biases.
b, T	Batch size and sequence length, respectively.

A.2 Memory Transfer Conditions

Let model A be a source model equipped with a trained Engram memory table $\mathcal{E}_A$, and let model B be a target model with hidden dimension d_B, tokenizer $\mathcal{T}_B$, and potentially a different architecture. We ask whether the frozen memory $\mathcal{E}_A$ can provide extractable external knowledge to B through a lightweight reader $\mathcal{W}^{(B)}$, even when $d_B \neq d_A$ and $\mathcal{T}_B \neq \mathcal{T}_A$.

We allow lightweight target-side reader training, and we do not assume zero-data or universal transfer across all memory architectures. Under this scope, successful extraction is meaningful only if three conditions can be met simultaneously:

- **Deterministic addressing**: memory lookup indices depend only on the input token sequence through a fixed canonicalization-and-hashing pipeline. They do not depend on the backbone's internal hidden states. The *same text* can therefore map to the *same memory entries* across backbones, provided the key pipeline is standardized.
- **Architectural decoupling**: the memory table $\mathcal{E}$ is extracted through learned projection matrices $\mathbf{W}_K$ and $\mathbf{W}_V$. These projections are natural reader interfaces because they already mediate between the memory space and the backbone hidden space.
- **Gate-based robustness**: even if some transferred memory entries are imperfectly aligned, the sigmoid gate α_t can softly suppress their contribution. In practice, this requires the

reader to learn a sufficiently negative match score and/or bias for unhelpful memory, thereby reducing interference with the hidden state.

The remainder of this section turns these conditions into a concrete decomposition. The *key space* determines which memory slot is selected for a piece of text (Appendix A.3); the *value space* determines how the vector stored in that slot is interpreted by the target backbone (Appendix A.4). Transfer, therefore, requires both address agreement and a reader that maps the retrieved vector into the target residual stream, with gating available to limit interference when alignment is incomplete (Appendix A.5).

A.3 Key-Space Unification

The first obstacle is address compatibility. In Engram's original design, the hash key is computed over model-specific canonical token IDs. For example, if model A assigns token ID 1823 to "San" and model B assigns ID 47, the resulting hash indices differ even when the underlying text is the same, so the two models would read unrelated memory entries.

To remove this mismatch, we define a *tokenizer-agnostic canonical key pipeline*. Let $\text{decode}(\cdot)$ denote the mapping from a tokenizer's token sequence to its surface string. For a shared decoded text prefix, both models first normalize the same surface form before hashing:

$$\mathcal{P}\big(\text{decode}(\mathcal{T}_A(x^{\text{text}}_{1:t_A}))\big) = \mathcal{P}\big(\text{decode}(\mathcal{T}_B(x^{\text{text}}_{1:t_B}))\big)\,, \tag{9}$$

where $x^{\text{text}}_{1:t_A}$ and $x^{\text{text}}_{1:t_B}$ denote source and target token prefixes that decode to the same text span, and $\mathcal{P}$ applies NFKC normalization, lowercasing, and accent stripping. In this construction, both models map the same decoded text to the same canonical form before hashing, so they can compute identical lookup indices into the shared memory table.

Subword boundary misalignment. The remaining difficulty is that tokenizers may segment the same word into different subword sequences. Three mitigations are available, in increasing order of robustness: (i) the original NFKC-based canonicalization when vocabularies overlap heavily; (ii) word-boundary N-grams computed over decoded strings; and (iii) character- or byte-level N-gram hashing. We adopt strategy (ii) as a practical middle ground: it substantially reduces tokenizer dependence without moving all the way to byte-level modeling. Our current experiments cover GPT-NeoX (Pythia), LLaMA, Qwen, and Phi tokenizers; languages or tokenizers without clean word boundaries may require the byte-level alternative.

A.4 Value-Space Alignment via Reader

Key agreement ensures that the target model reads the *same* memory entry as the source model; it does not ensure that the retrieved vector is meaningful in the target residual stream. Model A's memory vectors $\mathbf{e}_t \in \mathbb{R}^{d_{\text{mem}}}$ were shaped by gradients flowing through A's hidden geometry, which model B does not share.

Research on embedding space alignment [Mikolov et al., 2013, Kornblith et al., 2019, Sanjeev and Troynikov, 2024] suggests that representational spaces of models trained on similar data are often related by an approximately linear transformation. The Platonic Representation Hypothesis [Huh et al., 2024] offers one interpretation of this phenomenon, and Chen et al. [2025] showed empirically that affine maps can transfer linear features across LLMs. This motivates the smallest plausible bridge between source memory and target backbone:

$$\mathbf{W}_V^{(B)} : \mathbb{R}^{d_{\text{mem}}} \to \mathbb{R}^{d_B}, \qquad \mathbf{W}_K^{(B)} : \mathbb{R}^{d_{\text{mem}}} \to \mathbb{R}^{d_B}. \tag{10}$$

The memory table $\mathcal{E}_A$ remains *frozen*; only the reader parameters $\mathcal{W}^{(B)}$ are trained. For typical configurations ($d_{\text{mem}} = 512$, $d_B = 4096$), the reader size is

$$|\mathcal{W}^{(B)}| = 2 \cdot d_{\text{mem}} \cdot d_B + 2 \cdot d_B \approx 4.2\text{M}. \tag{11}$$

This is well within the LoRA-scale budget and small relative to multi-billion-parameter backbones. The linear choice is deliberate: it isolates whether cross-model reuse is already feasible under a minimal alignment assumption before introducing richer readers.

A.5 Gating as a Robustness Mechanism

The transfer mechanism also needs a fallback when the transferred memory is unhelpful. The context-aware gate provides exactly such a mechanism. If a memory entry is poorly aligned, for example because of a hash collision or because the source geometry does not translate cleanly to the target, the key vector $\mathbf{k}_t = \mathbf{W}_K^{(B)}\mathbf{e}_t$ will be geometrically dissimilar to the backbone query $\mathbf{h}_t$. By learning a sufficiently negative match score or bias in those cases, the reader can push α_t toward zero and suppress the memory contribution. The model can therefore tolerate a non-trivial amount of noisy or only partially aligned transferred memory without forcing that signal into the residual stream. This self-suppression is analogous to a soft mask in attention, but it operates on the memory channel itself and requires no explicit confidence model.

A.6 Comparison with Existing Knowledge-Augmentation Methods and Native Engram

Our source memory and target reader are trained with next-token prediction rather than task-specific labels. Consequently, a single fitted reader can be evaluated across multiple downstream tasks without being retrained for each task. This does not, however, imply universal downstream improvements. As shown in RQ5 (Appendix I.2), transferred memory provides stable intrinsic perplexity gains, whereas downstream improvements remain task-dependent and depend on the alignment among the memory corpus, target model, reader interface, and evaluation objective.

Table 4 situates frozen-memory extraction relative to retrieval-based, native-memory, and model-editing approaches. The main design distinction is that our method retains an explicit external memory while making the stored artifact portable across backbones through reader-only fitting.

Table 4: Qualitative comparison of knowledge augmentation approaches. ✓ = fully supported, ○ = partially supported, ✗ = not supported.

Property	KNN-LM	RAG	RETRO	Mem. Layers	LLM Mod.	Engram	Ours
$O(1)$ retrieval	✗	✗	✗	✗	✗	✓	✓
Parametric (trained)	✗	✗	✓	✓	✓	✓	✓
Cross-model portable	✗	✓	✗	✗	○	✗	✓
Surgical deletion	✗	✓	✗	✗	✗	✓	✓
No context overhead	✗	✗	○	✓	✓	✓	✓
Reader-only integration	N/A	N/A	✗	✗	✗	✗	✓

Differences from Native Engram Table 5 makes the architectural changes explicit at the formula level. Relative to native Engram, the transfer-oriented version uses shared canonical hashing, freezes the exported table, and replaces within-block integration with a target-side residual reader.

B Implementation Details

B.1 Models

We evaluate transfer across six independently developed model families with different architectures, tokenizers, and parameter scales: Pythia [Biderman et al., 2023], TinyLlama [Zhang et al., 2024], Qwen3.5 [Qwen Team, 2025], Phi-4-mini [Abouelenin et al., 2025], LLaMA 2 [Touvron et al., 2023], and Mistral [Jiang et al., 2023]. These choices are designed to separate several sources of transfer difficulty, including model scale, architectural mismatch, tokenizer mismatch, and differences in pre-training or post-training.

The main transfer matrix uses Pythia, Qwen3.5, and TinyLlama because together they provide both same-family and cross-family transfer settings across a wide range of model sizes. The Pythia pair offers a controlled same-tokenizer comparison, while transfers to TinyLlama and Qwen3.5 introduce a tokenizer and architectural mismatch. The peer-transfer study adds Phi-4-mini to test whether memory remains portable between similarly scaled but independently trained model families, rather than only from smaller to larger models.

The corpus-matched downstream suite uses Qwen3.5 source and target models, so that source scale, target scale, and corpus alignment can be varied within a single model family while keeping the

Table 5: Formula-level comparison between original Engram and our transfer-oriented reader. We replace the native within-block module with a target-side reader over frozen hashed memory.

Aspect	Original Engram	Transfer-oriented Engram Reader
Memory indexing	Layer-specific compressed-token n-gram hashing: $r_t^{(l,n,j)} = \left(\bigoplus_{k=0}^{n-1} c(x_{t-k}) \cdot a_k^{(l)}\right) \bmod p_{l,n,j}$	Shared canonical hashing: $r_t^{(n,j)} = \left(H(\text{canon_ngram}_t^{(n)}) \oplus s_j\right) \bmod M$
Memory representation	Layer-specific Engram embedding: $e_t^{(l)} = \text{Concat}_{n,j} \mathbf{E}_{l,n,j}[r_t^{(l,n,j)}]$	Shared memory vector: $m_t = \text{Concat}_j \mathbf{T}_j[r_t^{(j)}] \quad (d_{\text{mem}} = 512)$
Key / gate	$s_t = \frac{\langle \text{RMSNorm}(W_k e_t), \text{RMSNorm}(h_t) \rangle}{\sqrt{d}}$ $\alpha_t = \sigma\left(\text{sign}(s_t)\sqrt{\lvert s_t \rvert}\right)$	$k_t^{(r)} = W_k^{(r)} m_t, \quad v_t = W_v m_t$ $\alpha_t^{(r)} = \sigma\left(\frac{\langle \text{RMSNorm}(h_t), \text{RMSNorm}(k_t^{(r)}) \rangle}{\sqrt{d}} + \beta_r\right)$
Output update	$u_t = \alpha_t W_v e_t, \quad o_t = u_t + \text{ShortConv}(u_t)$	$o_t = \frac{1}{R}\sum_{r=1}^{R} \alpha_t^{(r)} v_t, \quad h_t \leftarrow h_t + o_t$
Branch structure	Native multi-branch inside backbone	Explicit reader branches (R controls capacity)
Injection	Internal transformer block component	Post-layer hook injection (e.g., layers 2 and 10)
Training	Joint training with backbone	Two-stage: source training + frozen memory + target reader fitting

broader architecture and tokenizer family controlled. This makes it easier to attribute downstream differences to memory content, target capacity, and reader fitting, rather than to unrelated model-family-level changes.

The QA experiments in Table 1 and associated ablations use LLaMA 2 as the memory source and Mistral as the target backbone. This pair provides a stronger cross-family setting at the 7B scale, with comparable model capacity but independently trained backbones and distinct tokenization and representation spaces. It therefore offers a demanding test of whether the reader can recover useful factual knowledge from a frozen source memory without relying on parameter scale or within-family compatibility.

Table 6 lists the checkpoints, experimental roles, hidden dimensions, checkpoint types, and tokenizer families used in these experiments.

Pythia uses the GPT-NeoX tokenizer, and the Qwen3.5 models share the Qwen3.5 tokenizer. TinyLlama and LLaMA-2 use LLaMA-family tokenizers, Phi-4-mini-instruct uses the tokenizer distributed with the Phi checkpoint, and Mistral-7B-v0.3 uses the Mistral tokenizer. These differences in the tokenizer and hidden size require the target reader to bridge representational and, in cross-tokenizer settings, tokenization boundaries rather than simply reusing a dimension-compatible memory interface.

B.2 Architecture Configuration

Memory Configuration The source Engram memory uses $N_{\max} = 3$ (bigram and trigram), $K = 4$ hash heads per order, table size $M = 65{,}536$, and head dimension $d_{\text{head}} = 64$, yielding $d_{\text{mem}} = 512$ and approximately 33.5M total memory parameters.

Reader Configuration We use the reader configuration appropriate to each experimental regime rather than fitting a separate reader for each evaluation benchmark. For the experiments in Table 1, the rows compare a single-layer one-branch reader, a dual-layer one-branch reader, and a dual-layer, four-branch reader. For the LLaMA-2-7B $\rightarrow$ Mistral-7B-v0.3 transfer ablation experiments in Table 24, memory is injected at layers 2 and 10, with four branch-specific key/gating paths at each site and a shared value projection. This reader is fitted with 2048-token Wikipedia-2021 sequences and is then applied zero-shot to the task prompts, without task-specific reader training. For the other experiments,

Table 6: Models and checkpoints used in the transfer experiments. "Source" denotes memory construction, and "target" denotes reader fitting for a frozen memory. Here, d is the residual hidden size used by the memory interface.

Family	Model	Experimental role	Params.	d	Checkpoint	Hugging Face identifier
Pythia	Pythia-160M	Source: main matrix	160M	768	Base	`EleutherAI/pythia-160m`
	Pythia-410M	Target: main matrix	410M	1024	Base	`EleutherAI/pythia-410m`
Qwen3.5	Qwen3.5-0.8B	Source: main matrix, scaling Target: downstream	0.8B	1024	Base	`Qwen/Qwen3.5-0.8B-Base`
	Qwen3.5-2B	Target: scaling, downstream Source: peer, downstream	2B	2048	Base	`Qwen/Qwen3.5-2B-Base`
	Qwen3.5-4B	Target: main matrix, peer, scaling, downstream	4B	2560	Base	`Qwen/Qwen3.5-4B-Base`
	Qwen3.5-9B	Source: main matrix, downstream Target: scaling, downstream	9B	4096	Base	`Qwen/Qwen3.5-9B-Base`
TinyLlama	TinyLlama-1.1B	Target: main matrix	1.1B	2048	Chat-/instruction-tuned	`TinyLlama/TinyLlama-1.1B-Chat-v1.0`
Phi	Phi-4-mini-instruct	Source and target: peer transfer	3.8B	3072	Instruction-tuned	`microsoft/Phi-4-mini-instruct`
LLaMA 2	LLaMA-2-7B	Source: open-domain QA, ablation	7B	4096	Base	`meta-llama/Llama-2-7b-hf`
Mistral	Mistral-7B-v0.3	Target: open-domain QA, ablation Source: self-transfer control	7B	4096	Base	`mistralai/Mistral-7B-v0.3`

we use the minimal single-layer, single-branch reader from Section 3.2. The reader is attached at the default relative depth $L/3$, where L is the number of transformer layers, and the target backbone and exported memory remain frozen while only the reader is optimized. The reader is fitted with the default 512-token training sequences and is then applied zero-shot to the task prompts, without task-specific reader training. These configuration choices reflect the purpose of each experiment. The QA reader-design study in Table 1 intentionally varies reader depth and branch capacity to measure how interface expressivity affects extraction from frozen memory, while the ablation study in Table 24 uses the strongest validated reader to isolate the roles of gating, addressing, and memory content.

B.3 Data, Training and Evaluation

We select the training corpora to match the purpose of each experimental regime rather than using a single corpus throughout. WikiText-103 provides a compact and reproducible language-modeling benchmark for the controlled transfer matrix, scaling, and representation analyses. FineWeb-Edu and the Nemotron-CC subsets provide substantially broader and more domain-diverse text for studying downstream transfer and corpus alignment. The five QA (NQ, WebQA, TriviaQA, TruthQA, HotpotQA) experiments use Wikipedia-2021 because their evaluation tasks emphasize factual and encyclopedic knowledge. Finally, LAMBADA, WikiText-103, and C4 are used in the distribution-shift experiments so that each fitted reader can be evaluated both on its training distribution and on corpora with different genres, structures, and provenances.

For Figures 2b, 4 and 5b and Tables 8 and 23 experiments, we use WikiText-103 [Merity et al., 2017]. The loader removes empty records, tokenizes each remaining text with the active model tokenizer without special tokens, concatenates the resulting token stream, and partitions it into fixed, non-overlapping sequences. The standard sequence length is 512 tokens; the standard train and test splits are used for training and final evaluation, respectively. Because the loader discards the incomplete final sequence and the evaluation data are batched with "drop-last", the test loader contains 34 or 36 batches in the standard runs due to tokenizer-dependent token counts, rather than because the evaluation size is manually set to one of those values.

For experiments in the downstream evaluation and Tables 13, 14 and 17, we use FineWeb-Edu [Lozhkov et al.] (`HuggingFaceFW/fineweb-edu`, "sample-10BT"), a broad educational web corpus, for source-memory and reader fitting. The domain-alignment experiments additionally use Nemotron-CC HQ-DQA [Su et al., 2025] and Nemotron-CC HQ from `nvidia/Nemotron-CC-v2.1`; the former is a QA-heavy STEM subset and the latter is a broader organic-web subset used as a corpus-size and

structure control. The resulting reader is evaluated zero-shot on RTE [Dagan et al., 2006], BoolQ [Clark et al., 2019], openBookQA [Mihaylov et al., 2018], SciQ [Welbl et al., 2017], TruthfulQA [Lin et al., 2022], and RACE [Lai et al., 2017].

For QA tasks in Table 1 to Table 24, it contains Natural Questions (NQ) [Kwiatkowski et al., 2019], WebQA [Berant et al., 2013], TriviaQA [Joshi et al., 2017], TruthfulQA [Lin et al., 2022], and HotpotQA [Yang et al., 2018]. The default corpus for the LLaMA-2-7B $\rightarrow$ Mistral-7B-v0.3 experiments is the pre-tokenized December-2021 English Wikipedia [Izacard et al., 2023] release (`Rubin-Wei/enwiki-dec2021-preprocessed-mistral`), distributed with the MLP Memory data pipeline [Wei et al., 2026]. It provides train and test splits; the implementation maps a requested validation split to test and, when the active tokenizer differs from Mistral's, decodes and re-tokenizes the windows before chunking them.

The out-of-domain language-modeling evaluation in Appendix D.5 uses LAMBADA [Paperno et al., 2016], WikiText-103 [Merity et al., 2017], and the English C4 validation set [Raffel et al., 2020]. LAMBADA and WikiText-103 are considered out-of-domain only when the reader is fitted to a different corpus. LAMBADA contains narrative text requiring long-range contextual prediction, whereas WikiText-103 consists of curated encyclopedic articles; both differ substantially in genre and provenance from broad web corpora such as C4 and FineWeb-Edu.

Training protocol. The principal settings are summarized in Table 7. All phases optimize the next-token language-modeling objective; no task labels are used to fit a reader.

For the standard Pythia source configurations, Phase 1 is end-to-end; the large Qwen and LLaMA source runs freeze the source backbone when required by the memory budget. After Phase 1, the exported memory table is frozen and attached to the target model. For Phase 2, only the target-side reader is optimized.

Conditions and metrics. Across transfer experiments, we compare a *Baseline* target with no memory, a *Transferred* condition with a trained frozen source memory, and a *Random memory* condition with an untrained table of the same architecture. Unless noted otherwise, all conditions use three seeds (42, 137, 2024). For the performance in Tables 1 and 2, we use F1 score as the main result, and multiple-choice average and follow the official code and checkpoint to evaluate for MLP memory. For the other downstream tasks, we report the accuracy. We report test perplexity (PPL, $\downarrow$) with 95% bootstrap confidence intervals and use paired t-tests on per-batch perplexities. The experiments in the appendix run on a single NVIDIA H100 NVL GPU. The experiments in the main text are conducted on the LUMI supercomputer operated by CSC – IT Center for Science, Finland, using the LUMI-G partition equipped with AMD Instinct MI250X GPUs. **Evaluation-set overlap audit.** We compare all 31,807 downstream evaluation questions against the exact Phase-1 and Phase-2 training streams used in the QA experiments. Natural Questions, WebQuestions, and HotpotQA contain no exact question matches. The remaining exact matches consist of four duplicated TriviaQA rows (0.013%) and one generic TruthfulQA question without its gold answer. Partial 8-gram overlaps predominantly correspond to shared phrasing rather than direct question-answer leakage. Thus, the reader training streams use no benchmark supervision, although they are not strictly overlap-free. Since the pretraining corpora of the underlying foundation models are unavailable, we interpret the QA results as controlled comparisons rather than as a contamination-free state-of-the-art claim.

C Supplementary Evidence for RQ1

The main 3×3 matrix in Section 4.1 answers the primary question of whether a frozen memory remains useful after being moved to a different backbone. The experiments below test whether that conclusion survives several plausible alternative explanations. Specifically, we examine whether the gains depend on shared tokenization, arise only when transferring from a smaller to a larger model, disappear for stronger targets, or can be explained by similarity between source and target representations.

C.1 Full Cross-Architecture Matrix

Table 8 reports the complete cross-architecture results for all nine source–target combinations used in the main transfer study in Figure 2a. For each source memory, we compare three conditions

on each target model: the target backbone without memory augmentation (*Baseline*), the frozen source memory integrated through a fitted target reader (*Transferred*), and a size-matched randomly initialized memory evaluated under the same reader fitting protocol (*Random*). The transferred-memory condition improves test perplexity in every source–target pair, with relative reductions ranging from 1.6% to 15.7%. The strongest improvement occurs for Qwen3.5-0.8B $\rightarrow$ TinyLlama-1.1B, while the random-memory control generally improves less than the corresponding transferred memory. In that same cell, random memory already reduces PPL by 13.9%; therefore, the 15.7% reduction is not evidence that all of the gain comes from transferred content, but reflects a smaller content-specific margin over the random control.

C.2 Supplement Transfer Comparisons Experiments

This section supplements the transfer comparison experiments in the same-tokenizer and cross-tokenizer transfer settings and compares against parameter-matched baselines and an external retrieval reference.

Table 9 isolates the simplest transfer setting, where the source and target share the Pythia family and tokenizer, and asks whether frozen memory still helps when comparing against parameter-matched alternatives. The result is still favorable to the transferred memory. It outperforms both random memory and the iso-parameter LoRA baselines, while kNN-LM remains a stronger but much more expensive non-parametric reference because it requires a retrieval datastore and nearest-neighbor search at inference time.

Table 10 moves to the harder Pythia $\rightarrow$ TinyLlama case, where tokenizer mismatch is unavoidable, and transfer depends on the shared canonicalization pipeline. Even in this setting, transferred memory reduces PPL from 10.63 to 9.50 and clearly beats random memory. This supports the claim that the exported memory is not tied to the source tokenizer as long as the target reader is given a compatible canonical interface.

C.3 Peer-to-Peer Transfer

This section tests whether transfer remains useful once the source and target are both strong mid-scale models rather than a small source feeding a much larger target. Table 11 shows the results of peer-to-peer transfer between Phi-4-mini [Abouelenin et al., 2025] (3.8B) and Qwen3.5-4B.

The gains remain bidirectional: Phi $\rightarrow$ Qwen improves by 10.1%, and Qwen $\rightarrow$ Phi improves by 9.2%. Random memory is substantially weaker in both directions, even though it uses the same target-side reader architecture and optimization budget. Thus, the observed gains cannot be attributed solely to the additional parameter capacity introduced by the reader.

Table 7: Principal training and evaluation settings. "LR / warm-up" reports the learning rate and number of warm-up steps. Unless otherwise noted, training uses cosine decay, AdamW with weight decay 0.01, and gradient-norm clipping at 1.0. Alternative-corpus experiments follow the intrinsic reader-fitting protocol with LAMBADA or English C4 replacing WikiText-103. OOD rows are evaluation-only.

Regime / phase	Corpus	Budget / cap	Seq. / batch	LR / warm-up	Trainable components and exceptions
			Primary training settings		
RQ1 4.1 Phase 1	WikiText-103	50M tokens	512 / 16	3×10^{-5} 1,000	Source backbone, memory, and source reader are trainable. For Qwen3.5 sources, the memory learning rate is 10^{-3}.
RQ1 4.1 Phase 2	WikiText-103	20M tokens	512 / 16	3×10^{-5} 500	Only the target reader is trainable; the target backbone and transferred memory remain frozen.
Downstream evaluation, Phase 1	FineWeb-Edu	50M tokens	512 / 2	10^{-3} / 1,000	Source backbone, memory, and source reader trained end-to-end; gradient checkpointing
Downstream evaluation, Phase 2	FineWeb-Edu	20M tokens	512 / 4	3×10^{-5} / 500	Target reader only; target backbone and transferred memory frozen; gradient checkpointing; early stopping patience 5
Table 1 to Table 24 Phase 1	Wikipedia-2021	10–30M tokens	2048 / 1	3×10^{-5} 1,000	The source memory and reader are trainable; the source backbone is frozen. Early stopping uses patience 3.
Table 1 to Table 24 Phase 2	Wikipedia-2021	10–30M tokens	2048 / 1	3×10^{-5} 500	Only the target reader is trainable; the target backbone and transferred memory remain frozen. Early stopping uses patience 3.
			Alternative-corpus training		
LAMBADA Phase 1	LAMBADA train split	50M tokens	512 / 16	3×10^{-5} 1,000	Source backbone, memory, and source reader are trainable. The finite training split is cycled as needed.
LAMBADA Phase 2	LAMBADA train split	20M tokens	512 / 16	3×10^{-5} 500	Only the target reader is trainable; the target backbone and transferred memory remain frozen.
C4 Phase 1	English C4 train stream	50M tokens	512 / 16	3×10^{-5} 1,000	Source backbone, memory, and source reader are trainable.
C4 Phase 2	English C4 train stream	20M tokens	512 / 16	3×10^{-5} 500	Only the target reader is trainable; the target backbone and transferred memory remain frozen.
			Out-of-domain evaluation without additional training		
LAMBADA	`lambada`, test	2M eval tokens	512 / 16	—	Evaluation only; source memory, reader, and target backbone remain frozen.
WikiText-103	`wikitext-103`, test	2M eval tokens	512 / 16	—	Evaluation-only distribution-shift probe; no trainable components.
C4	`allenai/c4`, English validation	5M eval tokens	512 / 16	—	Evaluation-only streaming web-text probe; no trainable components.

Table 8: Full cross-architecture transfer matrix: test PPL (↓) for 3 source memories × 3 target models. *Random* replaces the learned source memory with a size-matched randomly initialized memory under the same fitting protocol. Mean ± std over 3 seeds. Deltas shown next to the transferred result are relative to the no-memory baseline; negative values indicate better PPL. A displayed standard deviation of 0.0 indicates a value below 0.05 after rounding to one decimal place.

Target	Baseline	Transferred	Random
Pythia-160M source			
Pythia-410M	21.9	**21.6 ± 0.1** -1.6%	21.9 ± 0.0
Qwen3.5-4B	10.8	10.1 ± 0.0 -6.8%	10.3 ± 0.0
TinyLlama-1.1B	10.6	9.5 ± 0.0 -10.6%	10.0 ± 0.0
Qwen3.5-0.8B source			
Pythia-410M	22.9	21.4 ± 0.0 -6.8%	22.1 ± 0.0
Qwen3.5-4B	10.5	9.6 ± 0.0 -8.9%	10.2 ± 0.0
TinyLlama-1.1B	10.8	**9.1 ± 0.0** -15.7%	9.3 ± 0.0
Qwen3.5-9B source			
Pythia-410M	22.6	21.6 ± 0.0 -4.3%	22.3 ± 0.0
Qwen3.5-4B	10.5	9.4 ± 0.0 -10.3%	10.2 ± 0.0
TinyLlama-1.1B	10.4	9.1 ± 0.0 -12.2%	9.3 ± 0.0

Table 9: Same-tokenizer transfer (Pythia-160M → Pythia-410M). Mean ± std over 3 seeds. Deltas in the PPL column are relative to the no-memory baseline; negative values indicate better PPL. LoRA and Cross-LoRA are iso-parameter baselines (1.03M parameters, same 20M token budget). KNN-LM uses a 5M-token datastore with k=1024 retrieval at inference time.

Condition	Test PPL ↓	95% CI	Params / Cost
Baseline (no memory)	21.9	[20.6, 23.4]	—
Transfer Conditions			
Random memory	21.9 ± 0.0 -0.2%	[20.6, 23.3]	1.05M
Transferred memory	**21.6 ± 0.1** -1.6%	**[20.2, 23.0]**	**1.05M**
Parameter-Matched Baselines			
LoRA (rank 7) [Hu et al., 2022]	23.3 ± 0.8 +6.5%	[21.9, 24.9]	1.03M
Cross-LoRA [Xia et al., 2025]	23.9 ± 0.6 +9.3%	[22.5, 25.5]	1.03M
External Retrieval Reference			
KNN-LM [Khandelwal et al., 2020]†	20.4 -7.1%	[19.3, 21.5]	5M-tok store

†Non-parametric; requires k=1024 nearest-neighbor search per token at inference.

Table 10: Cross-tokenizer transfer (Pythia-160M → TinyLlama-1.1B). Mean ± std over 3 seeds. Deltas in the PPL column are relative to the no-memory baseline; negative values indicate better PPL. Reader parameters: 2.10M.

Condition	Test PPL ↓	95% CI	Params
Baseline (no memory)	10.6	[10.0, 11.3]	—
Transfer Conditions			
Random memory	10.0 ± 0.0 -5.9%	[9.4, 10.7]	2.10M
Transferred memory	**9.5 ± 0.0** -10.6%	**[8.9, 10.2]**	**2.10M**

Table 11: Peer-to-peer transfer between Phi-4-mini (3.8B) and Qwen3.5-4B. Source memories trained on WikiText-103 (50M tokens). Mean ± std over 3 seeds. Deltas shown next to the transferred result are relative to the no-memory baseline; negative values indicate better PPL.

Direction	Baseline	Transferred	Random
Phi-4-mini → Qwen3.5-4B	10.5	**9.5 ± 0.0** -10.1%	10.2 ± 0.0
Qwen3.5-4B → Phi-4-mini	12.6	11.5 ± 0.0 -9.2%	12.3 ± 0.0

Table 12: Target model scaling: Qwen3.5-0.8B source → Qwen3.5-{2B, 4B, 9B}. Mean ± std over 3 seeds. Deltas shown next to the transferred result are relative to the no-memory baseline; negative values indicate better PPL. Baselines are eval-only (no reader training).

Target	Baseline	Transferred	Random
Qwen3.5-2B	13.7	**11.7 ± 0.0** -14.1%	12.6 ± 0.0
Qwen3.5-4B	10.5	9.6 ± 0.0 -8.9%	10.2 ± 0.0
Qwen3.5-9B	9.3	8.5 ± 0.0 -8.3%	9.0 ± 0.0

C.4 Target Model Scaling

Table 12 keeps the source fixed and scales the target from 2B to 9B parameters to check whether the transfer disappears once the target becomes stronger.

The gains persist across all three target sizes. The largest relative improvement appears at 2B, but even the 9B target still benefits, which suggests that stronger backbones reduce but do not eliminate the value of the transferred memory.

C.5 Analysis of Backbone Representations

We use centered kernel alignment (CKA) [Kornblith et al., 2019] to measure the similarity between hidden representations produced by different target backbones. Linear CKA compares the geometry of two representation matrices and is invariant to isotropic scaling and orthogonal transformations, making it suitable for comparing models with different hidden dimensions. We use WikiText-103 for training with sequence length 512.

For each backbone, we extract hidden states from the Transformer block located approximately one third of the way through the network, $\lfloor L/3 \rfloor$, where L is the total number of Transformer blocks. This corresponds to block 4 of 12 for Pythia-160M, block 8 of 24 for Pythia-410M, and block 7 of 22 for TinyLlama-1.1B. We process the same WikiText-103 input sequences with each model, mean-pool the hidden states over non-padding tokens to obtain one representation per sequence, and compute pairwise linear CKA between the resulting representation matrices. We select the one-third-depth layer to compare intermediate representations after substantial contextual processing, while avoiding the stronger model-specific specialization that may occur near the output layer.

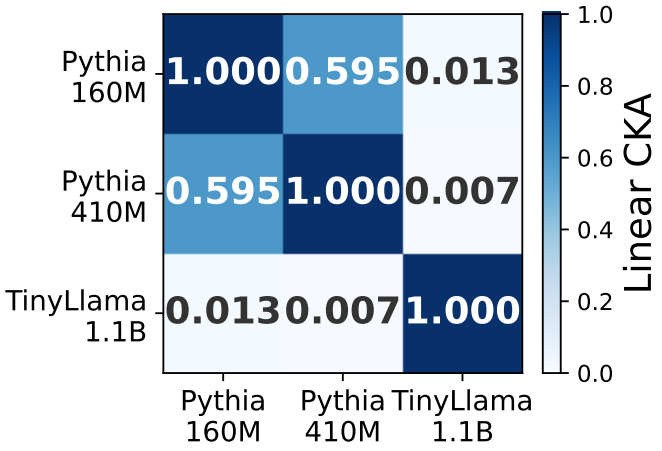


Figure 4: Linear CKA similarity between mean-pooled backbone representations at layer $\lfloor L/3 \rfloor$ for the three models shown in the matrix.

Figure 4 provides a representation-level view of why transfer need not track backbone similarity monotonically. The two Pythia models exhibit substantially higher representational similarity, with a CKA score of 0.595, than either model does with TinyLlama, for which the scores are only 0.013 and 0.007. Nevertheless, Pythia-160M → TinyLlama-1.1B is among the strongest cross-tokenizer transfer

settings in Table 8, reducing perplexity by 10.6%. By comparison, the more representationally similar Pythia-160M $\rightarrow$ Pythia-410M pair improves perplexity by only 1.6%.

Representational similarity alone does not predict the magnitude of transfer gains in these experiments. Nevertheless, this result is consistent with the reader analysis in Section 4.2: successful transfer depends on whether the target reader can recover and route useful information from the frozen memory, rather than requiring the source and target backbones to have closely aligned intermediate representation spaces. This observation complements the main-text finding that reader design, rather than storage capacity alone, determines the utility of a frozen artifact.

D Supplementary Analysis of Corpus-Dependent Downstream Transfer for RQ3

These analyses extend RQ3 in Section 4.3 by separating corpus specialization, reader alignment, mixed-corpus behavior, and out-of-domain effects.

D.1 Reference Downstream Pattern

The downstream evaluation reports the full six-task evaluation under FineWeb-Edu [Lozhkov et al.] with aligned Phase 2 training, so the table is not duplicated here. RTE and SciQ improve most consistently, BoolQ becomes positive for targets of 2B parameters and above, and TruthfulQA remains slightly negative at every scale. This pattern motivates the following diagnostics: corpus specialization in Appendix D.2, Phase 2 reader alignment in Appendix D.3, mixed-corpus training in Appendix D.4, and unrelated-corpus evaluation in Appendix D.5.

D.2 Source-Corpus Specialization

Table 13 compares two source-corpus families under matched Phase 2 adaptation: a broad educational web corpus (FW) and a narrower QA-heavy corpus (Nemo HQ-DQA).

Table 13: Domain alignment ablation with corpus-matched Phase 2 readers. These rows use a separate corpus-alignment control protocol and are not numerically identical to the main FineWeb-Edu values in Figure 3. Positive gains are shown in green and negative changes in red.

Target	Source	RTE ↑	BoolQ ↑	OpenBookQA ↑	SciQ ↑	TruthfulQA ↑	RACE ↑
Qwen3.5-0.8B	FW-avg	**+6.1**	-0.6	**+0.7**	**+1.8**	-0.1	-0.2
	Nemo-avg	**+7.9**	**+4.1***	**+1.1**	**+8.4***	-0.8	**+1.1***
Qwen3.5-2B	FW-avg	**+2.7**	**+3.2**	+0.3	**+3.5**	-0.4	-0.0
	Nemo-avg	**+8.7***	**+14.6***	**+0.5**	-0.7	+0.0	**+1.1***
Qwen3.5-4B	FW-avg	**+1.3**	-0.2	**+0.6**	**+0.9**	-0.5	+0.1
	Nemo-avg	+0.4	**+0.6**	+0.2	-6.3	-2.9	-0.4
Qwen3.5-9B	FW-avg	**+0.5**	**+0.6**	+0.2	**+2.7**	-0.7	**+1.0**
	Nemo-avg	-4.6	**+2.1***	-0.3	-3.7	+0.2	**+1.4**

*Nemo-avg $>$ FW-avg by $\geq$ 2 percentage points.

The main pattern is that domain-specialized memory helps most on smaller targets and QA-style tasks. Nemo-avg is especially strong on Qwen3.5-0.8B and Qwen3.5-2B for BoolQ, RTE, and SciQ, but the advantage is less uniform on 4B and 9B targets, showing that specialization helps most when the target still needs strong external support.

Table 14 controls for corpus size on Qwen3.5-2B by comparing an 8B-token STEM question-answering corpus (HQ-DQA) with a broader 26B-token general web corpus (HQ). Here, the smaller STEM question-answering corpus substantially outperforms the larger general web corpus on BoolQ and RTE and also improves RACE.

Table 14: Corpus structure vs. size on Qwen3.5-2B: accuracy change (Δ%) under corpus-matched Phase 2 readers. Positive gains are shown in green and negative changes in red.

Source corpus	BoolQ ↑	RTE ↑	OBQA ↑	SciQ ↑	TQA ↑	RACE ↑
HQ-DQA (8B, STEM Q&A)	**+15.3 ± 0.3**	**+8.9 ± 4.7**	**+0.3 ± 0.2**	+0.9 ± 3.0	+0.3 ± 0.2	**+1.4 ± 0.4**
HQ (26B, organic web)	+3.0 ± 0.4	+1.2 ± 2.1	+0.5 ± 0.1	-0.5 ± 0.0	+0.2 ± 0.3	-0.4 ± 0.2
FW-avg (reference)	+3.2	+2.7	+0.3	+3.5	-0.4	-0.0

D.3 Phase-2 Reader-Fitting Alignment

Table 15 probes the failure mode suggested in the main text: stronger source memories can look worse downstream if the target-side reader is trained on a mismatched Phase 2 objective.

Table 15: Reader alignment: downstream accuracy change (Δ%) under mismatched vs. matched Phase 2 adaptation. Positive gains are shown in green and negative changes in red.

Source	Phase 2	BoolQ Δ% ↑	RTE Δ% ↑	SciQ Δ% ↑
Mismatched Phase 2 (WikiText-103)				
10M	WikiText-103	+8.3 ± 0.5	+7.7 ± 3.9	-4.5 ± 0.5
50M	WikiText-103	+6.6 ± 0.4	+6.1 ± 4.7	-1.8 ± 0.6
200M	WikiText-103	+0.0 ± 0.0	+2.9 ± 4.7	-1.4 ± 0.6
Matched Phase 2 (HQ-DQA)				
10M	HQ-DQA	**+14.1 ± 1.2**	**+10.7 ± 4.0**	-2.3 ± 3.0
50M	HQ-DQA	**+15.3 ± 0.3**	**+8.9 ± 6.2**	+0.9 ± 3.7
200M	HQ-DQA	**+14.0 ± 0.8**	**+6.6 ± 4.9**	**+2.9 ± 4.6**

Under WikiText-103 Phase 2 adaptation, increasing source training from 10M to 200M tokens steadily hurts BoolQ transfer. Once Phase 2 is moved to the matched HQ-DQA distribution, however, all three source budgets recover to roughly 14–15% BoolQ gain, which shows that the apparent degradation is primarily an alignment problem at the reader rather than a problem with the frozen memory itself.

Table 16 gives a mechanistic view of the same effect through gate statistics.

Table 16: Gate activation statistics during BoolQ evaluation versus Phase 2 WikiText-103 training. Longer source training suppresses the gate globally; BoolQ shows a lower mean gate, while the near-closed-gate fraction rises on both distributions under mismatch.

Source	Gate mean		Frac. < 0.1	
tokens	WikiText-103	BoolQ	WikiText-103	BoolQ
10M	0.6	0.5	5.6%	3.6%
50M	0.6	0.5	9.1%	2.9%
200M	0.5	0.4	17.5%	15.3%

Longer source training suppresses the gate more strongly overall, and this is visible on BoolQ prompts both in the lower mean gate and in the rise of near-closed gates. At 200M source tokens, the mean gate on BoolQ drops to 0.352 and the fraction of near-closed gates rises to 15.3%, while WikiText-103 has a slightly higher near-closed fraction at 17.5% on the same threshold metric, consistent with broad suppression under reader mismatch rather than a BoolQ-only effect.

D.4 Broadening Specialist Memory through Corpus Mixing

Table 17 tests whether a strong specialist memory can be broadened without destroying the downstream gains that make it useful in the first place. The answer is mostly yes. A 50/50 HQ-DQA + FineWeb-Edu mix preserves the DQA aggregate and slightly exceeds the specialist reference, while sequential mixing and the orthogonal code mixture also retain most of the specialist advantage. The

main tradeoff is that broader mixtures help coverage more than they help the QA-focused aggregate itself.

Table 17: Mixed-corpus memory study on Qwen3.5-2B. Positive gains are shown in green and negative changes in red. The two rightmost columns report aggregate scores: DQA-agg = mean(BoolQ, RTE, SciQ); Br-agg = mean(OBQA, TruthfulQA, RACE).

Design	BoolQ ↑	RTE ↑	OBQA ↑	SciQ ↑	TQA ↑	RACE ↑	DQA-agg ↑	Br-agg ↑
Single-corpus references								
HQ-DQA ref.	**+15.3 ± 0.3**	**+8.9 ± 4.7**	+0.3 ± 0.2	+0.9 ± 3.0	+0.3 ± 0.2	+1.4 ± 0.4	**+8.4**	+0.7
FW-Edu ref.	+2.5 ± 1.7	+1.9 ± 0.2	-0.0 ± 0.2	**+3.2 ± 0.6**	-0.2 ± 0.2	+0.1 ± 0.1	+2.5	-0.1
Mixed or broadened memories								
50/50 mix	**+14.3 ± 1.1**	**+11.7 ± 3.5**	+0.4 ± 0.4	-0.5 ± 3.8	+0.3 ± 0.3	+0.5 ± 0.3	**+8.5**	+0.4
Sequential HQ→FW	**+13.6 ± 1.0**	**+11.9 ± 2.8**	-0.0 ± 0.2	-1.7 ± 2.7	+0.1 ± 0.4	+0.2 ± 0.3	**+7.9**	+0.1
Orthogonal + Code	**+14.4 ± 0.9**	**+9.8 ± 2.6**	+0.4 ± 0.3	-0.4 ± 3.4	-0.2 ± 0.7	+1.3 ± 0.4	**+7.9**	+0.5

D.5 Out-of-Domain Side Effects

Table 18 checks whether a reader trained in the WikiText-103 transfer setting systematically distorts language-modeling behavior on unrelated corpora, without additional adaptation. LAMBADA emphasizes long-range narrative completion, WikiText-103 provides a separately curated language-modeling test set, and C4 represents broad web text; together, they probe distribution shift away from the Phase 2 training corpus. The OOD effect is mostly neutral. Both targets get a modest improvement

Table 18: Out-of-domain evaluation: test PPL (↓) for baseline vs. transferred memory. Negative Δ indicates improvement.

Target Model	Dataset	Baseline	Transferred	Δ (%)
Pythia-410M	LAMBADA	41.0	41.0	+0.0%
	WikiText-103	22.6	**22.1**	-2.4%
	C4	24.9	**24.8**	-0.3%
TinyLlama-1.1B	LAMBADA	23.6	**23.5**	-0.7%
	WikiText-103	10.6	**10.1**	-5.0%
	C4	11.7	**11.6**	-0.2%

on WikiText-103 [Merity et al., 2017], while LAMBADA [Paperno et al., 2016] and C4 [Raffel et al., 2020] stay very close to the baseline in either direction. This supports the paper's boundary claim that transferred memory is not universally helpful, but it is also usually not catastrophically harmful outside the aligned regime.

D.6 Reader–Evaluation Distribution Mismatch

To isolate whether transfer requires identical reader-training and evaluation distributions, we fix a WikiText-trained Pythia-160M source memory and fit Pythia-410M readers on either WikiText-103 or C4. We then evaluate each reader on both corpora.

Results in Table 19 show that transfer improves over the no-memory target in all four conditions, so reader-training and evaluation corpora need not be identical. However, the substantially larger matched WikiText gain shows that the amount of recoverable signal remains distribution-sensitive. We therefore distinguish *portability*, which persists under moderate corpus mismatch, from *transfer magnitude*, which depends on alignment among the stored memory, reader-fitting distribution, and evaluation distribution.

E Multilingual Transfer

Word-boundary units are the default addressing granularity in our English-like experiments, but they are not appropriate for writing systems without reliable whitespace segmentation. For such

Table 19: Cross-corpus reader fitting and evaluation. Lower PPL is better.

Reader corpus	Eval. corpus	No memory	Transfer	Δ
WikiText	WikiText	22.616	22.113	-0.502
WikiText	C4	24.726	24.663	-0.062
C4	WikiText	22.616	22.538	-0.078
C4	C4	24.726	24.604	-0.122

Table 20: Character-span addressing for non-segmented languages. Results are averaged over three seeds.

Language	No-memory PPL	Transfer PPL	Rel. improvement
Chinese	20.1248	19.6718 ± 0.0033	2.251%
Japanese	15.3555	15.1985 ± 0.0251	1.023%

inputs, we replace word units with NFKC-normalized, case-folded non-whitespace Unicode character events and apply the same deterministic hashing procedure to character spans. This preserves the model-independent addressing principle while changing only the canonical unit used to form n-grams. This character-span variant restores non-degenerate cross-tokenizer addressing for Chinese and Japanese and yields positive transfer gains in both languages, as shown in Table 20. We then trained Qwen2.5-0.5B source memories and Llama-3.2-1B target readers separately on Chinese and Japanese Wikipedia, using character-span hashing, four branches at layers 2/10, fixed held-out 262K-token test splits, and seeds 42/137/2024.

F Computational Cost Analysis

We count multiply-adds (MAdds) for dense projections and report memory addressing separately, since a table lookup is a memory-traffic operation rather than a dense matrix multiplication. Let $N_{\max}$ denote the maximum n-gram order and let K denote the number of independent hash heads used for each order $n \in \{2, \ldots, N_{\max}\}$. The total number of hash heads is therefore

$$H = (N_{\max} - 1)K. \tag{12}$$

R be the number of reader branches at one injection site, and S be the number of injection sites. For the default memory configuration, $N_{\max} = 3$, $K = 4$, $H = 8$, and the memory dimension is $d_{\text{mem}} = Hd_{\text{head}} = 512$, where d_{head} is the dimension of the embedding row retrieved by one hash head. Each token, therefore, requires H deterministic hash indices and H embedding-row reads, together supplying $Hd_{\text{head}} = 512$ memory values. This cost is $O(N_{\max} + H)$ hash operations and $O(Hd_{\text{head}})$ memory traffic per site and does not grow with the table size M; in particular, it does not perform a nearest-neighbor search over the $M = 65{,}536$ rows.

At one injection site of the target-side reader, the shared-value, R-branch reader performs one $d_{\text{mem}} \times d_B$ value projection and R key projections of the same shape, where d_B is the hidden dimension of target model B. The remaining reader operations include RMS normalization, R key-query dot products, sigmoid gates, scalar–vector products, and branch aggregation. These operations require $O(Rd_B)$ work. We refer to this as the non-projection, lower-order term because it scales linearly with d_B, whereas the dense projections scale as $d_{\text{mem}}d_B$. With $d_{\text{mem}} = 512$, the projection term is substantially larger than the non-projection terms in the evaluated configurations. Thus, the dominant reader cost for a sequence of length T and batch size B is

$$C_{\text{reader}} = BTS\left[(R+1)d_{\text{mem}}d_B + O(Rd_B)\right] \quad \text{MAdds}, \tag{13}$$

or, per token and ignoring the lower-order gate terms, $S(R+1)d_{\text{mem}}d_B$. The single-layer, single-branch reader has $R = S = 1$, giving $2 \cdot 512 \cdot d_B$ MAdds per token: 1.05M at $d_B = 1024$ (Pythia-410M) and 4.19M at $d_B = 4096$ (the 7B configurations). The dual-layer, four-branch QA reader has $S = 2$ and $R = 4$, so its dominant projection cost is $2 \cdot 5 \cdot 512 \cdot 4096 = 20.97$M MAdds per token. The shared value projection is counted once per injection site, not once per branch.

For completeness, the corresponding trainable reader parameter count is

$$P_{\text{reader}} = S\left[(R+1)d_{\text{mem}}d_B + (R+1)d_B + R\right], \tag{14}$$

where the second term accounts for the RMSNorm weights and the final term for the branch-specific gate biases. This gives approximately 4.20M parameters for the one-site, one-branch reader at $d_B = 4096$, and 21.01M parameters for the two-site, four-branch reader used in the QA configuration. These reader parameters are the only additional trainable parameters in Phase 2; the transferred 33.6M-parameter memory table and the target backbone remain frozen.

The comparison with attention should be read as a projection-only lower bound. A standard self-attention layer has four dense projections, giving $4d_B^2$ MAdds per token before counting the sequence-dependent attention score and value products. This is 4.19M at $d_B = 1024$ and 67.11M at $d_B = 4096$, compared with 1.05M and 4.19M for the single-branch reader. The additional attention products contribute $O(Td_B)$ work per token for a length-T sequence, whereas the reader contribution remains constant per token. Consequently, the reader adds a small fixed overhead to each injected layer and does not change the transformer's quadratic dependence on sequence length. During Phase 2, freezing the backbone and memory removes their optimizer-state and parameter-update costs, although their forward computation and the backward path needed to train the reader are still executed.

Amortization across consumers. The practical advantage of a transferable artifact emerges when the same provider memory is reused across multiple targets. Using the measured payload times above, the cumulative cost of transfer to N adapted targets is

$$C_{\text{transfer}}(N) = 0.653 + 3.798N,$$

whereas training a fresh target-specific memory for each consumer costs

$$C_{\text{fresh}}(N) = 3.871N.$$

Under these measurements, the one-time provider cost is amortized after approximately nine adapted consumers by payload time. Using allocated GPU-hours gives a corresponding break-even point of approximately eleven consumers. These numbers are system- and configuration-specific rather than universal thresholds, but they illustrate the lifecycle regime in which reusable memory artifacts become advantageous. When the provider reader is directly compatible with the consumer, target-side adaptation is unnecessary: the measured consumer cost is zero training tokens and zero parameter updates, while mean QA rises from 32.12 to 38.31. The optional reader fitting raises it further to 38.50.

F.1 Measured System and Lifecycle Cost

Table 21: Measured cost for the LLaMA-2-7B→Mistral-7B-v0.3 using QA experiment configuration on LUMI supercomputer.

Condition	Tokens	Train time	GPU-hours	Peak inf. memory	Prefill
No memory	0	0	0	13.658 ± 0.020 GiB	55.10 ± 0.06 ms
Phase-1 source artifact	4.096M	0.653 h	3.154	–	–
Fresh Mistral memory	19.968M	3.871 h	15.760	–	–
Matched FFN	19.968M	3.680 h	14.929	13.736 ± 0.020 GiB	55.32 ± 0.49 ms
Transfer, incl. Phase 1	4.096M+19.968M	4.451 h	18.623	13.881 ± 0.020 GiB	57.69 ± 0.40 ms

Appendix F.1 shows the cost for transfer experiments from LLaMA-2-7B to Mistral-7B-v0.3. Relative to the no-memory target, the transferred configuration adds 0.223 GiB of peak inference memory and 2.60 ms of prefill latency in this measured setup. The Phase-1 source artifact is a one-time provider cost: the recorded run processes 4.096M tokens in 0.653 hours. Each adapted consumer then trains only the 21.01M-parameter reader, while the memory and backbone remain frozen.

G Ethics and Broader Impact

Transferred memory raises both opportunities and risks. On the positive side, a frozen external memory can reduce repeated backbone retraining and make factual updates more modular. On the negative side, the same mechanism can propagate memorized or provenance-sensitive content

across multiple target models if the source memory is not audited. Engram's advantage is that its stored knowledge is explicit and addressable: entries can be inspected, removed, or replaced without retraining the backbone.

H Limitations

Scale and availability. Our experiments cover source and target models up to 9B parameters, but do not systematically characterize how target-side reader capacity should scale with backbone size, memory size, training data, or source–target heterogeneity. In particular, our two-phase protocol first trains the memory with the source model and then freezes both the transferred memory and the target backbone while adapting only the reader. We therefore do not explore joint training of the backbone, memory, and reader, nor establish a scaling law relating reader depth, branch count, injection placement, and parameter count to model scale. Although the present results show that stronger readers substantially improve frozen-memory extraction, it remains unclear whether the same reader configurations remain sufficient for much larger models or whether reader capacity must grow with the backbone or memory.

Addressing and language diversity. Our default word-boundary addressing is effective for the evaluated English-like settings, but it is not universally tokenizer- or language-independent. In particular, whitespace-based word units degenerate for non-segmented Chinese and Japanese. We show that replacing word units with normalized character spans repairs this specific failure mode and produces positive transfer on Chinese and Japanese Wikipedia. These experiments demonstrate that the model-independent addressing principle can extend beyond whitespace-delimited text, but they do not establish a universally optimal canonical unit. Code, noisy Unicode, mixed-script text, byte-level tokenizers, and additional writing systems may require character-, byte-, or task-specific span definitions. More generally, portability requires agreement on both the canonicalization procedure and the granularity from which memory addresses are formed. More aggressive tokenizer mismatch may require a stronger key pipeline than the word-boundary canonicalization used here.

Reader scope. Our capacity diagnostics further show that both interface width and reader placement affect extraction quality; the default $d_{\mathrm{mem}} = 512$ setting should therefore be viewed as an evaluated operating point rather than a fixed architectural requirement. The target-side reader is a central component of our transfer framework, but the present study evaluates only a deliberately lightweight subset of the possible reader design space. The main transfer matrix uses a single-branch linear reader, while the QA experiments extend it with shared-value multi-branch reading and multi-layer injection. These controlled designs make it possible to isolate the role of target-side extraction while keeping the number of trainable parameters small. Nevertheless, more heterogeneous source–target pairs may benefit from nonlinear transformations, adaptive branch aggregation, layer-dependent value projections, or learned injection placement. Such extensions could improve extraction capacity, but would also increase computation and make it harder to distinguish reusable memory content from capacity introduced by the reader itself. In addition, our results indicate that the usefulness of context-dependent gating varies with the source and target distributions. A broader study across corpus shifts and model families is needed to characterize when gating is essential and when a simpler reader is sufficient Xie et al. [2026].

I Additional Results: RQ4–RQ6

I.1 RQ4: Does transferred memory improve answer-token support in question answering?

Our fourth question is whether transferred memory measurably improves support for the correct answer, rather than merely changing final task accuracy in a hard-to-interpret way. By *answer-token support*, we mean the log-probability that the target model assigns to a valid gold-answer continuation at the positions where that answer is generated. To test this, we compare four inference settings on the same LLaMA-2-7B $\rightarrow$ Mistral-7B dual-layer QA system used in Table 22: transferred memory, random memory, memory disabled, and an answer-position memory ablation. We compute the length-normalized gold-answer log-probability for an alias a as:

$$s_c(i,a) \triangleq \frac{1}{L_a} \sum_{t=1}^{L_a} \log p_c\left(y_{i,t}^a \mid x_i, y_{i,<t}^a\right), \tag{15}$$

where p_c is the target model distribution under condition $c \in \{\text{trans}, \text{rand}, \text{disabled}, \text{ablated}\}$, and L_a is the token length of alias a. The probability is computed with teacher forcing: the model is given the prompt and the gold-answer prefix, and we evaluate the log-probability assigned to the next gold-answer token. For questions with multiple valid aliases $\mathcal{A}_i$, the question-level score is:

$$s_c(i) \triangleq \max_{a \in \mathcal{A}_i} s_c(i, a). \tag{16}$$

For datasets with a single annotated answer, $\mathcal{A}_i$ contains only that answer. Finally, the mean score difference over the dataset $\mathcal{D}$ is:

$$\Delta \log p(c_1 - c_2) \triangleq \frac{1}{|\mathcal{D}|} \sum_{i \in \mathcal{D}} \left(s_{c_1}(i) - s_{c_2}(i) \right), \tag{17}$$

where $\mathcal{D}$ is the evaluation set. In Table 22, we report $\Delta \log p(\text{trans} - \text{rand})$, $\Delta \log p(\text{trans} - \text{disabled})$, and $\Delta \log p(\text{trans} - \text{ablated})$. The first two differences ask whether transferred memory helps more than either an untrained table or no memory at all. The third asks whether zeroing the memory channel only at answer-prediction positions reduces gold-answer likelihood. For NQ and TriviaQA, where many examples have multiple valid aliases, the analysis scores all annotated aliases and keeps the best-valid gold continuation rather than forcing the first alias in the dataset.

Table 22: QA memory-contribution trained on Wikipedia-2021 analysis for LLaMA-2-7B [Touvron et al., 2023] → Mistral-7B-v0.3 [Jiang et al., 2023] with dual-layer injection. The $\Delta \log p$ columns report mean per-example differences in *length-normalized* gold-answer log-probability. Positive values indicate that transferred memory makes the gold answer more likely.

Task	Transferred	Random	Disabled	Ablated	$\Delta \log p$ (Trans. - Random)	$\Delta \log p$ (Trans. - Disabled)	$\Delta \log p$ (Trans. - Ablated)
NQ	25.1	20.1	18.3	26.2	+0.035	+0.032	-0.013
WebQA	32.3	27.4	31.1	33.6	+0.147	+0.197	-0.001
TriviaQA	72.5	65.5	58.9	72.5	+0.047	+0.072	-0.000
TruthQA	30.8	32.6	31.7	30.9	-0.581	-0.258	-0.029
HotpotQA	27.1	22.9	18.6	27.2	+0.022	+0.019	-0.009

Transferred memory beats random memory and memory-disabled evaluation on four of the five QA subsets, both in task score and in gold-answer log-probability. The cleanest gains appear on WebQA and TriviaQA. On WebQA, transferred improves mean gold-answer log-probability by $+0.147$ over random memory and $+0.197$ over memory-disabled inference. On TriviaQA, after fixing alias-aware gold-answer scoring, transferred also improves gold-answer log-probability by $+0.047$ over random memory and $+0.072$ over memory-disabled inference, aligning the logit-level analysis with the task-level F1 gains. NQ and HotpotQA show the same direction with smaller positive margins.

TruthfulQA remains the main negative case. Transferred is worse than both random memory and memory-disabled inference there, both in MC-average and in gold-answer log-probability. This matches the broader boundary condition already visible in RQ3: transferred memory is most useful on factual retrieval-style QA, but not on truthfulness-style calibration.

The ablation column sharpens the interpretation. For every task, transferred minus answer-position ablation is near zero or slightly negative rather than strongly positive. That means the main benefit is not coming from a simple direct injection of the correct answer token at the final prediction positions. Instead, the evidence points to an indirect effect: transferred memory improves the model's overall answer-supporting state on factual QA tasks, but direct answer-token memory contribution is weak and sometimes slightly harmful.

Taken together, RQ4 supports a narrower claim than "memory use causes the gain". The gain is real on NQ, WebQA, TriviaQA, and HotpotQA, and the corrected alias-aware analysis strengthens that claim for TriviaQA in particular. The results are therefore more consistent with improved factual support after target-side integration than with raw answer-position copying.

Takeaway. Transferred memory improves answer-token support selectively but positively: transferred memory increases gold-answer support on NQ, WebQA, TriviaQA, and HotpotQA relative to both random memory and no memory, with the corrected alias-aware TriviaQA analysis removing the main contradictory case, while the near-zero answer-position ablation effects indicate that the gain comes more from useful factual support than from direct answer-token injection.

I.2 RQ5: How data-efficient is frozen-memory transfer?

We first ask whether frozen-memory transfer reduces the amount of target-side training required to reach competitive performance. Across both Pythia and Qwen model pairs, transferred memory reaches strong intrinsic performance substantially earlier than learning a new memory from scratch, indicating improved target-side data efficiency. The downstream gains are also positive, although their magnitude remains task-dependent.

Controlled scaling on Pythia. Table 23 provides a controlled comparison on Pythia-160M $\rightarrow$ Pythia-410M by tracking the same transfer setting across 5M, 20M, and 50M target-side training tokens.

Table 23: Scaling analysis for frozen-memory transfer on Pythia-160M $\rightarrow$ Pythia-410M. Results report test perplexity as a function of target-side training tokens, averaged over 3 seeds.

Condition	Trainable Params	5M	20M	50M
Transferred	**1.05M**	**21.8**	**21.5**	**21.5**†
From scratch	34.6M	21.9	21.8	21.6

†Early stopped at step 1,500 ($\approx$12M tokens); validation PPL diverges beyond this point.

Transferred memory is already competitive after only 5M target-side tokens and essentially saturates by 20M, reaching a test PPL of 21.5. In contrast, training a new target memory from scratch improves more gradually and approaches a comparable PPL only at the largest token budget. This difference is particularly notable because the transferred condition updates only 1.05M trainable parameters, compared with 34.6M for the from-scratch condition, corresponding to roughly 3% as many trainable parameters.

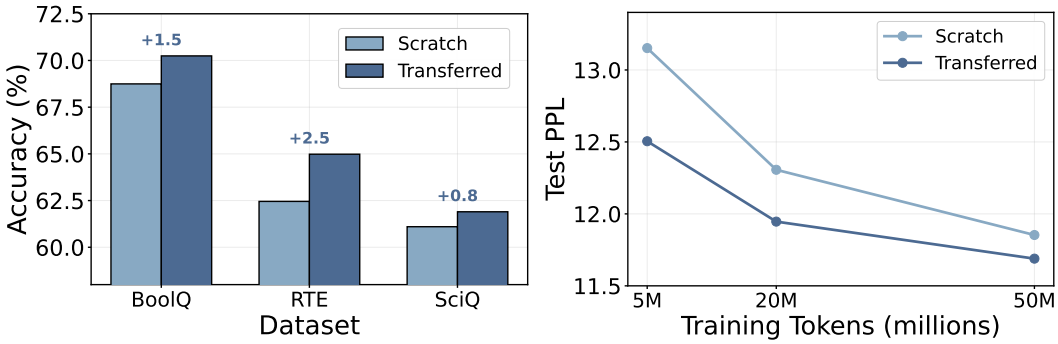
The 50M transferred entry is reported at its best early-stopped checkpoint. Although the nominal training budget is 50M tokens, validation perplexity begins to diverge after approximately 12M tokens, and the best checkpoint therefore matches rather than improves upon the 20M result. Taken together, these results suggest that Phase 2 is primarily adapting and reusing structure already learned in the source memory rather than relearning an equivalent memory representation from the target corpus alone.

Scaling to a larger target model. We next test whether the same trend persists at a larger scale using Qwen3.5-0.8B $\rightarrow$ Qwen3.5-2B. Figure 5 reports both intrinsic perplexity and downstream QA performance as the amount of target-side training data increases.

The Qwen results reinforce the Pythia scaling study. Transferred adaptation achieves lower test perplexity than scratch training across all evaluated target-side token budgets, showing that the intrinsic data-efficiency advantage is not restricted to the smaller Pythia setting. The downstream results show a similar but less uniform trend: transferred memory improves BoolQ, RTE, and SciQ by $+1.5$, $+2.5$, and $+0.8$ points, respectively.

These results also clarify the distinction between intrinsic adaptation and downstream utility. The perplexity advantage of transfer is stable across token budgets, whereas the magnitude of downstream improvement depends on the task. Thus, transferred memory provides a consistently better target-side initialization, but the extent to which this advantage translates into downstream accuracy still depends on whether the reader can extract and route the stored signal in a task-compatible manner.

Takeaway. Frozen-memory transfer is target-data efficient: it reaches strong perplexity with substantially fewer target-side updates than learning a new memory from scratch, while downstream gains remain positive but task-dependent.



(a) Qwen3.5-2B downstream QA scaling: average zero-shot accuracy across BoolQ, RTE, and SciQ as a function of target-side training tokens.

(b) Qwen3.5-2B intrinsic scaling: test perplexity versus target-side training tokens for scratch and transferred memory.

Figure 5: Target-side scaling for Qwen3.5-0.8B → Qwen3.5-2B with the source memory trained on WikiText-103. Transferred memory maintains a consistent intrinsic perplexity advantage, while downstream improvements remain positive but task-dependent.

I.3 RQ6: What drives the gains from memory transfer?

Having established the target-side data-efficiency advantage of transfer, we next ask what actually drives the downstream gains. In particular, we test whether the improvement can be explained simply by adding trainable target-side parameters, or whether successful transfer depends on the transferred memory content, its addressing structure, and the reader used to access it.

We study this question on LLaMA-2-7B [Touvron et al., 2023] → Mistral-7B-v0.3 [Jiang et al., 2023] using the same dual-layer QA setting that produces our strongest downstream results. Table 24 compares the full transferred reader against controls designed to isolate the main components of the transfer mechanism.

"Transferred (R=4)" denotes the full dual-layer four-branch reader. "No gate" removes context-dependent gating and always injects memory values. "Affine stitch" replaces the reader with a single affine memory-to-hidden-state mapping. "Permuted keys" preserves the memory table but destroys addressing integrity through index permutation. "Random memory" replaces the learned memory table with an untrained one. "Train from scratch" learns a fresh target memory under the same downstream training setup. Finally, "FFN only" replaces memory lookup with a parameter-matched residual feed-forward module, while "No memory baseline" corresponds to the underlying Mistral-7B model without memory augmentation.

Table 24: Ablation study of frozen-memory transfer on LLaMA-2-7B → Mistral-7B-v0.3, with the source memory trained on Wikipedia-2021. Results report held-out test perplexity and downstream QA performance.

Condition	Test PPL ↓	95% CI	NQ	WebQA	TriviaQA	TruthQA	HotpotQA	Avg ↑
No memory baseline	9.9	[9.8, 10.1]	20.6	29.3	57.7	32.1	21.0	32.1
Transferred (R=4)	8.7	[8.6, 8.9]	30.3	33.7	69.9	30.9	27.6	38.5
Interface Simplifications								
No gate	10.1	[9.9, 10.3]	24.2	27.7	61.6	31.0	24.1	33.7
Affine stitch	9.5	[9.4, 9.7]	25.0	29.2	62.8	30.4	23.9	34.3
Content and Training Controls								
Permuted keys	8.7	[8.5, 8.8]	18.3	34.5	57.7	30.6	21.4	32.5
Random memory	8.7	[8.6, 8.9]	23.0	27.1	65.1	33.6	24.1	34.6
Train from scratch	8.1	[8.0, 8.2]	29.9	33.0	70.4	30.9	27.9	38.4
No-memory Control								
FFN only (param-matched R=4)	7.4	[7.2, 7.5]	25.5	27.9	64.1	31.6	23.6	34.5

Parameter count alone does not explain the gains. The full transferred reader improves average QA from 32.1 for the no-memory baseline to 38.5. More importantly, it clearly outperforms the parameter-matched FFN control, which reaches only 34.5 average accuracy. The difference cannot be attributed to intrinsic language-model fit: the FFN-only control actually obtains the lowest perplexity in the table (7.4 versus 8.7 for transfer), yet remains 4.0 points behind on average QA.

The task-level results show the same pattern. The transferred reader outperforms the parameter-matched FFN on NQ, WebQA, TriviaQA, and HotpotQA, with particularly clear margins on NQ (30.3 versus 25.5), TriviaQA (69.9 versus 64.1), and HotpotQA (27.6 versus 23.6). The FFN control is slightly stronger only on TruthQA. Thus, simply attaching an equally parameterized target-side residual module does not reproduce the downstream behavior of transferred memory.

The reader interface is critical. Removing context-dependent gating reduces average QA from 38.5 to 33.7, a drop of 4.8 points. Replacing the reader with a single affine stitch similarly reduces the average to 34.3, a 4.2-point decrease. These results indicate that transfer quality depends not only on the presence of a memory table, but also on a sufficiently expressive interface for selecting, transforming, and routing retrieved information into the target model.

Addressing integrity and memory structure matter. Permuting the memory keys while retaining the same underlying table reduces average QA from 38.5 to 32.5. This is the largest degradation among the memory-content controls and shows that preserving the association between addresses and stored values is important for downstream reuse.

Replacing the transferred table with random memory also decreases average QA to 34.6. Interestingly, random memory achieves the same test PPL as the transferred condition (8.7), yet performs substantially worse on downstream QA. This provides another example in which intrinsic perplexity alone does not predict whether a memory representation will be useful for downstream tasks.

Transfer primarily improves initialization rather than the asymptotic ceiling. Training a fresh target memory from scratch eventually reaches an average QA score of 38.4, essentially matching the 38.5 obtained by transfer. We therefore do not interpret transfer as providing a substantially higher asymptotic performance ceiling. Instead, together with the scaling results in Appendix I.2, the evidence supports a more specific interpretation: the transferred memory provides a structured initialization that can reach useful target-side performance with substantially less adaptation than learning an equivalent memory representation from scratch.

This distinction also explains why perplexity and downstream QA do not rank all controls identically. For example, the parameter-matched FFN reaches the best intrinsic PPL but substantially worse QA, while scratch memory eventually matches transferred QA after sufficient target-side optimization. Successful transfer therefore depends on the interaction between memory structure, addressing, and the reader interface, rather than on parameter count or perplexity reduction alone.

Takeaway. The downstream gains cannot be explained by parameter count alone: successful transfer depends on meaningful memory addressing and a sufficiently expressive reader interface, while its main advantage over scratch training is better target-side data efficiency rather than a higher asymptotic ceiling.

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 explicitly frame the paper as a study of frozen-memory extraction across backbones and identify reader quality as a key determinant of the gains. These claims match the transfer matrix, OpenQA reader ablations, and the stated scope limitations.

   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: The paper contains a dedicated Limitations section covering model scale, tokenizer coverage, and reader scope, and the Conclusion explicitly narrows the claims to Engram-style hashed memory and aligned factual/reasoning-heavy settings.

   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 present formal theorems or proof-based theoretical results; it provides problem formulation, equations, and empirical analysis only.

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: Sections 3–4 and the appendix specify the models, datasets, memory configuration, optimization protocol, token budgets, seeds, ablations, and evaluation settings needed to reproduce the reported transfer and OpenQA results.

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: [Yes]

   Justification: The submission provides an anonymized public repository URL in the abstract for code and reproducible setups.

   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 Experimental Settings section and appendix specify the datasets, sequence lengths, model families, memory size, source and target training stages, learning rates, schedules, seeds, evaluation metrics, and benchmark suites used in the paper.

   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: The paper reports mean$\pm$std across seeds, 95% bootstrap confidence intervals for perplexity, and paired t-tests where appropriate, and it states these conventions in the Experimental Settings section.

   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: [No]

   Justification: The paper reports the GPU type used for the experiments, but it does not provide per-run wall-clock time, memory footprint, or total project compute for all experimental groups.

   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: To the best of our knowledge, the research complies with the NeurIPS Code of Ethics; the paper preserves anonymity and discusses provenance, privacy, and responsible reuse concerns in the Ethics Considerations section.

   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: The Ethics Considerations section discusses positive impacts such as modular memory updates and reduced retraining cost, and negative impacts such as provenance-sensitive or memorized content being propagated across target models.

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: This submission does not release new model checkpoints, memory tables, or datasets, so no release-specific safeguard policy is described here.

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: [No]

Justification: The paper cites the original datasets and model families used in the experiments, but it does not explicitly enumerate the licenses and terms of use for each external asset in the manuscript.

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 does not claim a new public release of datasets or checkpoints beyond the anonymized code and reproduction repository already linked in the paper.

    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 does not involve crowdsourcing 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: The paper does not involve human subjects research.

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: The core method and experiments do not use LLMs as a non-standard methodological component beyond the pretrained backbones being studied.

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.