# Engram: Conditional Memory via Scalable Lookup

### A Comprehensive Technical Analysis

**Paper**: arXiv:2601.07372v1 (January 12, 2026)
**Authors**: DeepSeek-AI + Peking University
**Repository**: https://github.com/deepseek-ai/Engram

---

## Executive Summary

The Engram paper introduces **conditional memory** as a new sparsity axis for large language models, complementing the established paradigm of conditional computation (MoE). The central thesis is that Transformers lack a native knowledge lookup primitive, forcing them to simulate retrieval through expensive computation. Engram addresses this by storing N-gram pattern embeddings in hash-indexed tables, enabling $O(1)$ retrieval of static local patterns while preserving the Transformer backbone for dynamic reasoning.

Under rigorous iso-parameter and iso-FLOPs constraints, Engram-27B outperforms the MoE-27B baseline across diverse benchmarks: knowledge-intensive tasks (MMLU +3.4), general reasoning (BBH +5.0), and code/math (HumanEval +3.0). The paper's U-shaped allocation curve demonstrates that optimal sparse model design allocates approximately 20-25% of inactive parameters to Engram memory rather than MoE experts.

The architecture is strategically significant given documented constraints on Chinese AI compute capacity. Engram's ability to offload parameters to host DRAM (bypassing HBM bottlenecks) and extract more capability per FLOP directly addresses the hardware limitations acknowledged at the January 2026 Beijing AGI Summit (Reuters, 2026). The paper's structure—detailed concept validation at modest scale, infrastructure described but not benchmarked at frontier scale—matches DeepSeek's historical publication-to-deployment pattern, suggesting high probability of inclusion in their forthcoming V4 frontier model.

Key technical innovations include:

- **Tokenizer compression**: Surjective mapping collapsing semantically equivalent tokens (23% vocabulary reduction)
- **Multi-head hashing**: Collision-robust retrieval via $K$ independent hash functions
- **Context-aware gating**: Learned scalar gates filtering irrelevant or collision-contaminated retrievals
- **mHC integration**: Branch-specific gating enabling expressivity-efficiency tradeoffs
- **Memory-compute decoupling**: Prefetch-and-overlap strategy enabling <3% overhead when offloading 100B parameters to host memory

The paper opens significant unexplored territory, including domain-specialized Engram modules for high-accuracy applications (medical, legal, scientific) and the broader design space of memory-expressivity tradeoffs within the conditional memory paradigm.

---

## Section-by-Section Summary

### Section 1: Introduction

The paper opens by identifying a fundamental limitation in Transformer architectures: the absence of a native knowledge lookup primitive. While MoE provides conditional computation (sparse expert activation), Transformers lack conditional memory (sparse knowledge retrieval). This forces models to reconstruct static patterns—named entities, collocations, idioms—through iterative computation at every forward pass.

Engram proposes treating conditional memory as a first-class modeling primitive. The key insight is that local N-gram patterns are deterministic given the input tokens, enabling hash-based $O(1)$ retrieval rather than learned routing. This architectural choice decouples memory scaling from compute scaling: adding Engram parameters increases storage requirements but not per-token FLOPs.

The introduction frames the paper's core contribution as answering the allocation question: given fixed total parameters and activated parameters, how should sparse capacity be distributed between MoE experts (conditional computation) and Engram embeddings (conditional memory)?

## Section 2: Method

### Section 2.1: Tokenizer Compression

Standard subword tokenizers create multiple token IDs for semantically equivalent surface forms ("Apple", "apple", " apple", "APPLE"). This is catastrophic for N-gram lookup, as each variant maps to different hash slots.

Engram implements a surjective mapping $P : V \to V'$ that collapses equivalent tokens via:

- NFKC Unicode normalization
- Lowercasing
- Whitespace stripping
- Diacritic removal

This achieves 23% vocabulary reduction, improving coverage per embedding slot and reducing hash collision probability.

### Section 2.2: Multi-Head Hashing

The combinatorial space of N-grams ($|V|^3 \approx 10^{15}$ for trigrams with 128k vocabulary) cannot be stored explicitly. Engram uses $K$ independent hash functions mapping N-grams to table slots of size $M$:

$$\phi_{n,k} : \mathbb{Z}^n \to \{0, 1, \ldots, M-1\}$$

Multi-head hashing provides collision robustness: if two N-grams collide in one hash function, they almost certainly differ in others. With $K = 8$ heads and $M = 3 \times 10^6$ slots, the probability of total collision is approximately $10^{-52}$, making catastrophic collision effectively impossible.

### Section 2.3: Context-Aware Gating

Static lookup cannot resolve polysemy ("bank" as financial institution vs. riverbank) or filter hash collisions. Engram introduces a learned gating mechanism:

$$\alpha_t = \sigma\left(\frac{\mathrm{RMSNorm}(h_t)^\top \cdot \mathrm{RMSNorm}(k_t)}{\sqrt{d}}\right)$$

The gate $\alpha_t \in (0, 1)$ modulates the retrieved embedding based on compatibility with the Transformer's hidden state $h_t$. This reintroduces minimal dynamic computation ($O(d)$ for dot product) while preserving the efficiency benefits of static retrieval.

### Section 2.4: Multi-Branch Integration with mHC

Engram integrates with Manifold-Constrained Hyper-Connections (mHC; Xie et al., 2025), which expand the residual stream to $M$ parallel branches. The design shares expensive components (embedding table, value projection) while separating cheap components (key projections, gates):

- **Shared**: One embedding table $E$, one value projection $W_V$
- **Separate**: $M$ key projections $\{W_K^{(m)}\}$, $M$ scalar gates $\{\alpha^{(m)}\}$

This enables branch-specific decisions about memory utilization while amortizing storage costs.

### Section 2.5: Decoupling Memory from Compute

The section describes two system designs:

**Training**: Standard model parallelism shards Engram tables across GPUs, using All-to-All communication to gather active embeddings. This distributes but does not eliminate HBM requirements.

**Inference**: Deterministic hash-based addressing enables prefetch-and-overlap strategies. Since indices are known before the forward pass, embeddings can be asynchronously retrieved from host DRAM via PCIe while preceding layers compute. Engram modules at layers 2 and 15 provide sufficient computation buffer to mask transfer latency.

The paper describes (but does not benchmark) a multi-level cache hierarchy exploiting Zipfian N-gram distribution: frequent patterns in GPU HBM, common patterns in host DRAM, rare patterns on NVMe SSD.

## Section 3: Scaling vs. Sparsity

### Section 3.1: Allocation Under Fixed Constraints

The paper introduces the allocation ratio $\rho \in [0, 1]$, where $\rho$ determines the fraction of sparse parameters assigned to MoE experts versus Engram memory. Under fixed total and activated parameters, sweeping $\rho$ reveals a **U-shaped validation loss curve**:

- $\rho = 100\%$ (pure MoE): Loss = 1.7248
- $\rho \approx 75\text{-}80\%$ (optimal): Loss = 1.7109
- $\rho \to 0\%$ (Engram-dominated): Loss increases

The U-shape is replicated at two compute budgets ($2 \times 10^{20}$ and $6 \times 10^{20}$ FLOPs), suggesting stability across regimes. The optimal allocation dedicates approximately 20-25% of sparse capacity to Engram.

### Section 3.2: Infinite Memory Regime

With a fixed MoE backbone, sweeping Engram capacity from $10^5$ to $10^7$ slots reveals **log-linear scaling**: validation loss decreases linearly with log(slots). This suggests Engram can scale efficiently beyond tested ranges.

Comparison with OverEncoding (Huang et al., 2025a) shows Engram extracts more value from equivalent memory budgets, attributed to deeper injection points, context-aware gating, and tokenizer compression.

## Section 4: Experiments

### Section 4.1: Pre-training Setup

Models are trained for 50,000 steps on 262B tokens:

- **Dense-4B**: 4.1B total, 3.8B activated
- **MoE-27B**: 26.7B total, 3.8B activated (72 routed + 2 shared experts)
- **Engram-27B**: 26.7B total, 3.8B activated (55 routed + 2 shared experts, 5.7B Engram)
- **Engram-40B**: 39.5B total, 3.8B activated (55 routed + 2 shared experts, 18.5B Engram)

All models use DeepSeek-V3 tokenizer, MLA attention (DeepSeek-AI, 2024a), mHC ($M = 4$), and Muon optimizer.

### Section 4.2: Pre-training Results

Engram-27B outperforms MoE-27B across all benchmark categories:

| Category | Representative Benchmark | MoE-27B | Engram-27B | $\Delta$ |
|---|---|---|---|---|
| Knowledge | MMLU | 57.4 | 60.4 | +3.0 |
| Reasoning | BBH | 50.9 | 55.9 | +5.0 |
| Code | HumanEval | 37.8 | 40.8 | +3.0 |
| Math | GSM8K | 58.4 | 60.6 | +2.2 |

Engram-40B shows further gains on most benchmarks but underperforms Engram-27B on code tasks (HumanEval 38.4 vs 40.8). The paper attributes this to undertraining—the loss gap between Engram-40B and baselines continues widening at training end.

## Section 5: Long-Context Capability

### Section 5.1: Experimental Setup

Models undergo YaRN context extension (Peng et al., 2023) from 4k to 32k tokens over 5,000 additional steps.

### Section 5.2: Results and Analysis

The paper makes three controlled comparisons:

1. **Iso-Loss** (46k vs 50k steps): Matching pre-training loss isolates architectural effects. Engram dramatically outperforms on complex retrieval: Multi-Query NIAH 97.0 vs 84.2 (+12.8), Variable Tracking 87.2 vs 77.0 (+10.2).

2. **Iso-FLOPs** (50k vs 50k steps): Standard comparison shows Engram advantages compound with its better base quality.

3. **Extreme** (41k vs 50k steps): Engram at 82% training compute matches MoE on LongPPL while exceeding on RULER tasks.

The mechanism: Engram handles local patterns via $O(1)$ lookup, freeing attention capacity for global context management. Tasks requiring broad attention (Frequent Words Extraction: +26.3) show largest gains.

## Section 6: Analysis

### Section 6.1: Effective Depth

LogitLens analysis (nostalgebraist, 2020) shows Engram representations converge to prediction-ready states earlier (lower KL divergence at early layers). CKA analysis (Kornblith et al., 2019) reveals Engram layer 5 representations match MoE layer ~12 representations for named entities.

Interpretation: By offloading static pattern reconstruction to lookup, Engram effectively increases model depth—early layers can immediately begin reasoning rather than spending capacity on feature composition.

### Section 6.2: Structural Ablations

Layer placement sweep finds layer 2 optimal for single-module Engram (balances early intervention with contextual precision for gating). Multi-branch integration and context-aware gating are critical; depthwise convolution provides marginal benefit.

### Section 6.3: Sensitivity Analysis

Suppressing Engram output during inference reveals functional specialization:

- **Factual knowledge** (TriviaQA): Catastrophic collapse to 29% retained
- **Reading comprehension** (C3): Resilient at 93% retained

This demonstrates Engram becomes the primary repository for parametric knowledge, while the backbone retains comprehension and reasoning capabilities.

### Section 6.4: System Efficiency

Table 4 demonstrates 100B parameter Engram offloaded to host DRAM incurs <3% throughput overhead (8,858 vs 9,032 tokens/sec on 4B backbone). This validates the prefetch-and-overlap strategy.

## Section 7: Related Work

The paper positions Engram against:

- **N-gram language models** (Shannon, 1948; Jurafsky & Martin, 2024): Engram modernizes the concept with learned embeddings and neural integration
- **OverEncoding** (Huang et al., 2025a): Prior N-gram embedding work limited to input layer averaging
- **Product key memory** (Lample et al., 2019): Attention-based retrieval vs. Engram's hash-based deterministic lookup
- **Retrieval-augmented generation** (Lewis et al., 2020): External document retrieval vs. Engram's internal parametric memory

---

# Detailed Technical Analysis

## 1. Classical N-gram Models: Foundation and Connection to Engram

### Lay Analogy: Your Phone's Predictive Keyboard

Imagine texting a friend. You type **"I'll meet you at the"** and your phone suggests **"airport"**, **"office"**, or **"usual"**. How does it know? It has memorized millions of text messages and learned that certain words frequently follow certain phrases.

Your phone doesn't analyze the entire conversation—it just looks at the **last few words** and consults a giant lookup table: *"When people type 'at the', what do they usually type next?"*

This is exactly how an N-gram model works:

- It memorizes patterns from training text
- It only looks at the **immediate local context** (the last $N-1$ words)
- Prediction is a **table lookup**, not computation

The "N" in N-gram refers to the window size. A **3-gram (trigram)** model looks at the previous 2 words to predict the next one.

#### Mathematical Foundation

**The Goal: Assign Probability to Sequences** Given a sentence $W = (w_1, w_2, \dots, w_T)$, we want to compute $P(W)$—the probability of this exact word sequence occurring.

Using the **chain rule of probability**, we decompose this as:

$$P(w_1, w_2, \dots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, w_2, \dots, w_{t-1})$$

**Problem**: Computing $P(w_t \mid w_1, \dots, w_{t-1})$ requires conditioning on the *entire* history. For a vocabulary of size $|V|$ and sequence length $T$, we'd need to estimate $|V|^T$ parameters—astronomically intractable.

**The Markov Assumption (Key Simplification)** The N-gram model makes the $(N-1)$**th-order Markov assumption**: the probability of the next word depends only on the previous $(N-1)$ words:

$$P(w_t \mid w_1, \dots, w_{t-1}) \approx P(w_t \mid w_{t-N+1}, \dots, w_{t-1})$$

This truncates history to a fixed-size window, making the model tractable.

#### Notation Summary

| Symbol | Meaning |
|---|---|
| $w_t$ | Word at position $t$ |
| $V$ | Vocabulary (set of all words) |
| $\lvert V \rvert$ | Vocabulary size |
| $N$ | The "N" in N-gram (context window + target) |
| $w_{t-N+1}^{t-1}$ | Shorthand for $(w_{t-N+1}, \dots, w_{t-1})$ — the context |
| $C(\cdot)$ | Count function (occurrences in training corpus) |

**Maximum Likelihood Estimation** We estimate probabilities by **counting** co-occurrences in a training corpus:

$$P_{\text{MLE}}(w_t \mid w_{t-N+1}^{t-1}) = \frac{C(w_{t-N+1}, \dots, w_{t-1}, w_t)}{C(w_{t-N+1}, \dots, w_{t-1})}$$

In plain English:

$$P(\text{next word} \mid \text{context}) = \frac{\text{Times we saw (context + next word) together}}{\text{Times we saw (context) at all}}$$

**Concrete Examples** **Bigram ($N = 2$)**: Conditions on 1 previous word

$$P(\text{mat} \mid \text{the}) = \frac{C(\text{"the mat"})}{C(\text{"the"})} = \frac{1,247}{89,432} \approx 0.014$$

**Trigram ($N = 3$)**: Conditions on 2 previous words

$$P(\text{mat} \mid \text{on}, \text{the}) = \frac{C(\text{"on the mat"})}{C(\text{"on the"})} = \frac{342}{5,891} \approx 0.058$$

Notice: The trigram probability is higher because "on the" provides more specific context than just "the".

**Full Sentence Probability** For the sentence **"the cat sat on the mat"** using a trigram model:

$$P(\text{sentence}) = P(\text{the} \mid \langle s \rangle, \langle s \rangle) \times P(\text{cat} \mid \langle s \rangle, \text{the}) \times P(\text{sat} \mid \text{the}, \text{cat}) \times \dots$$

Where $\langle s \rangle$ is a special start-of-sentence token.

**Visual Representation: Trigram Decomposition**

```
                    TRIGRAM MODEL (N=3)

   Sentence: "the cat sat on the mat"


     Context   Context   Target      Probability
      w_{t-2}   w_{t-1}    w_t

       <s>       <s>       the       P(the | <s>, <s>)
       <s>       the       cat       P(cat | <s>, the)
       the       cat       sat       P(sat | the, cat)
       cat       sat       on        P(on | cat, sat)
       sat       on        the       P(the | sat, on)
       on        the       mat       P(mat | on, the)
       the       mat      </s>       P(</s> | the, mat)


   The sliding window moves through the sentence:

   [<s>  <s>  the] cat  sat  on  the  mat  </s>
        [<s>  the  cat] sat  on  the  mat  </s>
             [the  cat  sat] on  the  mat  </s>
                  [cat  sat  on] the  mat  </s>
                       [sat  on  the] mat  </s>
                            [on  the  mat] </s>
                                 [the  mat  </s>]
```

**The Lookup Table Structure**

```
                  N-GRAM PROBABILITY TABLE
                  (Trigram Example)

          Context (Key)                 Next Word Probabilities

      "the cat"                      sat: 0.12, is: 0.08, ...
      "cat sat"                      on: 0.25, down: 0.18, ...
      "sat on"                       the: 0.35, a: 0.22, ...
      "on the"                       mat: 0.18, floor: 0.15, ...
      "how are"                      you: 0.85, things: 0.08, ...|
           ...                                 ...


                    ↓ At inference time ↓

        Input: "sat on"  →  Lookup  →  Output: P(the|sat,on) = 0.35
                                O(1)
```

**Handling Edge Cases: Smoothing**

A critical problem: **unseen N-grams get probability zero**, which makes entire sentences have $P = 0$.

**Example**: If "quantum cat" never appeared in training:

$$P(\text{sat} \mid \text{quantum}, \text{cat}) = \frac{C(\text{"quantum cat sat"})}{C(\text{"quantum cat"})} = \frac{0}{0} = \text{undefined}$$

**Solutions** (smoothing techniques):

1. **Laplace (Add-1) Smoothing**: Add 1 to all counts

$$P_{\text{Laplace}}(w_t \mid w_{t-1}) = \frac{C(w_{t-1}, w_t) + 1}{C(w_{t-1}) + |V|}$$

2. **Kneser-Ney Smoothing**: Sophisticated interpolation using lower-order models
3. **Backoff**: If trigram unseen, fall back to bigram, then unigram

**Computational Complexity**

| Operation | Time Complexity | Space Complexity |
|---|---|---|
| Training (counting) | $O(T)$ where $T$ = corpus size | $O(\|V\|^N)$ worst case |
| Inference (lookup) | $O(1)$ per prediction | — |
| Storage | — | $O(M)$ where $M$ = unique N-grams |

The $O(1)$ **lookup** is the key property that makes N-grams attractive—and what the Engram paper exploits.

**Key Properties Summary**

| Property | Assessment |
|---|---|
| Lookup complexity | $O(1)$—instant retrieval |
| Interpretability | High—direct frequency counts |
| Long-range dependencies | None—limited to $(N-1)$ context |
| Generalization | None—"cat sat" ≠ "dog sat" |
| Data sparsity | Severe—most N-grams unseen |

**Connection to Engram**

Engram modernizes N-gram models by:

| Classic N-gram | Engram |
|---|---|
| Stores probability distributions | Stores dense embedding vectors |
| Exact string matching | Hash-based approximate matching |
| Zero probability for unseen N-grams | Hash collisions handled by gating |
| No semantic generalization | Tokenizer compression groups equivalent forms |
| Standalone model | Module within Transformer backbone |

The core insight remains the same: **local patterns don't require deep computation—they can be retrieved in $O(1)$ time**, freeing neural network depth for tasks that actually require reasoning.

---

## 2. Tokenizer Compression: Canonical Projection for N-gram Efficiency

### The Problem: Tokenizer Fragmentation

Standard subword tokenizers (BPE, SentencePiece) are designed for **lossless text reconstruction**, not semantic coherence. This creates a proliferation of token IDs that represent the same underlying concept:

```
Raw text: "Apple"  → Token ID: 12847
Raw text: "apple"  → Token ID: 18234
Raw text: " apple" → Token ID: 31092  (note the leading space)
Raw text: " Apple" → Token ID: 45123
Raw text: "APPLE"  → Token ID: 67891
Raw text: "äpple"  → Token ID: 89012  (diacritic variant)
```

All six tokens refer to the same semantic concept, but each has a completely different ID. For N-gram lookup, this is catastrophic:

- The trigram **"the red apple"** and **"The red Apple"** would hash to entirely different embedding slots

- You'd need to observe both variants in training to learn both patterns
- The combinatorial explosion is severe: if each position has 6 variants, a trigram has $6^3 = 216$ possible ID combinations for semantically identical content

#### The Solution: Canonical Projection

The paper implements a **surjective mapping** $P : V \to V'$ that collapses semantically equivalent tokens into canonical representatives:

```
P: V → V'  (surjective = many-to-one)

P(12847) = P(18234) = P(31092) = P(45123) = P(67891) = P(89012) = 7234
   ↑         ↑         ↑          ↑          ↑          ↑          ↑
"Apple"   "apple"   " apple"  " Apple"   "APPLE"    "äpple"   canonical
                                                                "apple"
```

The mapping applies several normalization steps:

1. **NFKC Unicode Normalization**: Converts compatibility characters to canonical forms (ﬁ → fi, ² → 2, ä → a in some modes)
2. **Lowercasing**: A → a
3. **Whitespace stripping**: " apple" → "apple"
4. **Diacritic removal** (implied): café → cafe

#### What Appendix C Reveals

The paper's Table 6 shows the most aggressively merged groups:

| Canonical Token | Merge Count | Original Variants |
|---|---|---|
| '␣' (whitespace) | 163 | \t, \n, \r, ␣, ␣␣, \n\n, ␣␣␣, ␣\n, ... |
| 'a' | 54 | A, a, ␣a, ␣A, á, ä, ã, ą, à, å, â, ... |
| 'o' | 40 | O, o, ␣o, ␣O, ó, ö, ô, õ, ő, ò, ... |
| 'e' | 35 | E, e, ␣e, ␣E, é, è, ę, ě, ê, ... |
| 'i' | 30 | I, i, ␣I, ␣i, í, ì, î, ï, ... |

The 23% vocabulary reduction means roughly 30,000 token IDs are collapsed into existing canonical forms.

#### Concrete Step-by-Step Example

The complete Engram embedding process for a real input:

#### Input Sentence

```
"The Milky Way galaxy"
```

#### Step 1: Standard Tokenization (DeepSeek-V3 tokenizer)

```
Text:     "The Milky Way galaxy"
Tokens:   ["The", " Mil", "ky", " Way", " galaxy"]
Raw IDs:  [1847,   29341,  8472,  15234,  31847]
```

**Step 2: Tokenizer Compression (Canonical Projection)** Apply $P()$ to each raw ID:

```
Raw ID    Text        Normalization Steps            Canonical ID
------    ----        ---------------------          ------------
1847      "The"       lowercase → "the"              → 892
29341     " Mil"      strip space, lower → "mil"     → 4521
8472      "ky"        lowercase → "ky"               → 8470
15234     " Way"      strip space, lower → "way"     → 2847
31847     " galaxy"   strip space, lower → "galaxy"  → 12453

Canonical IDs: [892, 4521, 8470, 2847, 12453]
```

**Critical point**: If the input had been "THE MILKY WAY GALAXY" or " the milky way galaxy", the canonical IDs would be **identical**.

**Step 3: Form N-gram Contexts** For a trigram model ($N = 3$), extract suffix N-grams at each position:

```
Position t=0: g_{0,3} = (<s>, <s>, 892)          → "the"
Position t=1: g_{1,3} = (<s>, 892, 4521)         → "the mil"
Position t=2: g_{2,3} = (892, 4521, 8470)        → "the milky"
Position t=3: g_{3,3} = (4521, 8470, 2847)       → "milky way"
Position t=4: g_{4,3} = (8470, 2847, 12453)      → "way galaxy"
```

**Step 4: Multi-Head Hashing** For each N-gram, apply $K$ different hash functions to get embedding indices:

```
For g_{3,3} = (4521, 8470, 2847) representing "milky way":

Hash Head 1:  _{3,1}(4521, 8470, 2847) = 2847391 mod 3000017 = 847374
Hash Head 2:  _{3,2}(4521, 8470, 2847) = 9182734 mod 3000017 = 182700
...
Hash Head 8:  _{3,8}(4521, 8470, 2847) = 1928374 mod 3000017 = 928357
```

The hash function is multiplicative-XOR:

$$\phi(x_1, x_2, x_3) = ((x_1 \cdot p_1) \oplus (x_2 \cdot p_2) \oplus (x_3 \cdot p_3)) \mod M$$

Where $p_1, p_2, p_3$ are different prime multipliers per head, and $M$ is a prime table size.

**Step 5: Embedding Retrieval** Look up embeddings from each table:

```
e_{3,3,1} = E_{3,1}[847374]    ^{d/K}    # Head 1 embedding
e_{3,3,2} = E_{3,2}[182700]    ^{d/K}    # Head 2 embedding
...
e_{3,3,8} = E_{3,8}[928357]    ^{d/K}    # Head 8 embedding
```

**Step 6: Concatenate Across Heads and N-gram Orders**

```
For position t=3 ("way"):

From bigrams (N=2):
  e_{3,2} = [e_{3,2,1}  e_{3,2,2}   ...   e_{3,2,8}]  # "milky" → "way"

From trigrams (N=3):
  e_{3,3} = [e_{3,3,1}  e_{3,3,2}   ...   e_{3,3,8}]  # "the milky" → "way"

Final memory vector:
  e  = [e_{3,2}  e_{3,3}]   ^{d_{mem}}
```

**Step 7: Context-Aware Gating** The retrieved embedding $e_3$ is static—it doesn't know the actual context. The gating mechanism modulates it:

$$k_3 = W_K \cdot e_3$$

$$v_3 = W_V \cdot e_3$$

$$\alpha_3 = \sigma\left(\frac{\text{RMSNorm}(h_3)^\top \cdot \text{RMSNorm}(k_3)}{\sqrt{d}}\right)$$

$$\tilde{v}_3 = \alpha_3 \cdot v_3$$

If the context $h_3$ (from preceding Transformer layers) is incompatible with the retrieved memory (e.g., hash collision retrieved "Milky Way candy bar" context when we need "Milky Way galaxy"), $\alpha_3 \to 0$ and the memory is suppressed.

**Visual Diagram of the Full Process**

```
                         ENGRAM EMBEDDING PIPELINE

  INPUT: "The Milky Way"



    STANDARD TOKENIZER
    (BPE/SentencePiece)



  Raw IDs: [1847, 29341, 8472, 15234]
    "The"    " Mil"  "ky"   " Way"



       TOKENIZER COMPRESSION P: V → V'

       • NFKC normalization
       • Lowercasing
       • Whitespace stripping
       • Diacritic removal



  Canonical IDs: [892, 4521, 8470, 2847]
     "the"    "mil"    "ky"    "way"

                      KEY INSIGHT: These canonical IDs are
                      IDENTICAL for "THE MILKY WAY",
                      "the milky way", " The Milky Way", etc.



           FORM N-GRAM CONTEXTS

    Bigrams:   (the,mil) (mil,ky) (ky,way)
    Trigrams:  (the,mil,ky) (mil,ky,way)



           MULTI-HEAD HASHING

    For each N-gram, K hash functions:

    (mil,ky,way)        → idx: 847374
                         → idx: 182700
                         → idx: 293847
                       ...



                EMBEDDING TABLE LOOKUP (O(1))
```

```
E_{3,1}[847374]  → [0.12, -0.34, 0.87, ...]  (d/K dims)
E_{3,2}[182700]  → [0.45, 0.23, -0.11, ...]  (d/K dims)
...

Concatenate all heads & N-gram orders:
e_t = [e_{2-gram}   e_{3-gram}]   ^{d_{mem}}



           CONTEXT-AWARE GATING

h_t
(from Transformer)

e_t   → W_K   → k_t     →  _t =  (h_t · k_t / √d)
      → W_V   → v_t     → ṽ_t =  _t · v_t

If context mismatches retrieved memory:  _t → 0 (suppress)
If context aligns with retrieved memory:  _t → 1 (use it)



RESIDUAL CONNECTION TO BACKBONE

H^( ) ← H^( ) + Conv(ṽ_t)
```

### Why This Matters for N-gram Efficiency

**Without Tokenizer Compression** Consider training on a corpus with these occurrences:

```
"the Milky Way" appears 10,000 times
"The Milky Way" appears 8,000 times
"THE MILKY WAY" appears 500 times
" the milky way" appears 3,000 times
```

Without compression, these are **four separate N-gram entries**, each with fewer training examples. The embedding for each variant is learned independently.

**With Tokenizer Compression** All 21,500 occurrences contribute to a **single canonical N-gram**:

```
(the, milky, way) → single embedding learned from 21,500 examples
```

This provides:

1. **Better statistics**: More training signal per pattern
2. **Smaller tables**: 23% fewer slots needed
3. **Better generalization**: Rare variants (like “THE MILKY WAY”) benefit from common variants
4. **Reduced hash collisions**: Fewer unique N-grams means lower collision probability per slot

### The Surjective Function in Practice

The mapping $P$ is implemented as a precomputed lookup table:

```python
# Pseudocode for tokenizer compression
class TokenizerCompressor:
    def __init__(self, tokenizer):
        self.projection = {}  # Raw ID → Canonical ID

        # Group tokens by normalized form
```

```
        normalized_groups = defaultdict(list)
        for token_id in range(tokenizer.vocab_size):
            token_text = tokenizer.decode([token_id])
            canonical = self.normalize(token_text)
            normalized_groups[canonical].append(token_id)

        # Assign canonical IDs
        canonical_id = 0
        for canonical_text, raw_ids in normalized_groups.items():
            for raw_id in raw_ids:
                self.projection[raw_id] = canonical_id
            canonical_id += 1

        self.compressed_vocab_size = canonical_id  # ~77% of original

    def normalize(self, text):
        text = unicodedata.normalize('NFKC', text)  # Unicode normalization
        text = text.lower()                          # Lowercase
        text = text.strip()                          # Strip whitespace
        # Additional normalizations...
        return text

    def compress(self, token_ids):
        return [self.projection[tid] for tid in token_ids]
```

The compression happens **only for Engram indexing**—the main Transformer backbone still uses the original token IDs and embeddings. This is crucial: you want the model to distinguish "Apple" (company) from "apple" (fruit) in its representations, but for N-gram pattern matching, the surface form variations shouldn't matter.

---

## 3. Multi-Head Hashing: Principled Collision Mitigation

### The Combinatorial Problem

For vocabulary size $|V| = 128,000$ and trigrams ($N = 3$):

$$|\text{possible trigrams}| = |V|^3 = 128,000^3 \approx 2.1 \times 10^{15}$$

You cannot allocate 2 quadrillion embedding slots. So you must compress this space.

### Single-Head Hashing: The Naive Approach

A single hash function maps the astronomical N-gram space to a manageable table:

$$\phi : \mathbb{Z}^N \to \{0, 1, \dots, M-1\}$$

Where $M$ might be 3 million (a prime, for better distribution).

```
("the", "milky", "way")     →  slot 847,374
("quantum", "field", "theory")     →  slot 2,391,847
("the", "red", "apple")     →  slot 847,374   ← COLLISION!
```

**The problem**: When two unrelated N-grams collide, they share the same embedding. The model learns a muddled average of all colliding patterns. With $2 \times 10^{15}$ N-grams and $3 \times 10^6$ slots, the expected collisions per slot is ~700 million N-grams—catastrophic.

But most N-grams are vanishingly rare or never occur in training. The *effective* collision rate depends on the training distribution, which follows Zipf's law. Still, collisions are inevitable and damaging with a single head.

### Multi-Head Hashing: Collision Mitigation

The key insight is probabilistic: **if two N-grams collide in one hash function, they almost certainly won't collide in $K$ independent hash functions**.

With $K = 8$ heads, each with table size $M$:

$$P(\text{collision in all heads}) = \left(\frac{1}{M}\right)^K$$

For $M = 3,000,017$ (prime) and $K = 8$:

$$P(\text{total collision}) = \left(\frac{1}{3 \times 10^6}\right)^8 \approx 10^{-52}$$

This is astronomically unlikely. In practice, two N-grams will share *some* heads but not *all* heads:

```
N-gram A: "the milky way"
  Head 1: slot 847,374  ←
  Head 2: slot 182,700       collision
  Head 3: slot 2,918,374
  Head 4: slot 501,283  ←
  Head 5: slot 1,847,291
  Head 6: slot 928,174
  Head 7: slot 2,103,847
  Head 8: slot 384,192

N-gram B: "the red apple"
  Head 1: slot 847,374  ←    same slot (collision)
  Head 2: slot 2,847,123      different
  Head 3: slot 918,234        different
  Head 4: slot 501,283  ←   same slot (collision)
  Head 5: slot 2,918,473      different
  Head 6: slot 129,384        different
  Head 7: slot 1,029,384      different
  Head 8: slot 2,918,473      different
```

The concatenated embeddings are:

$$e_A = [E_1[847374] \| E_2[182700] \| E_3[2918374] \| \ldots \| E_8[384192]]$$

$$e_B = [E_1[847374] \| E_2[2847123] \| E_3[918234] \| \ldots \| E_8[2918473]]$$

Even with 2 collisions out of 8 heads, 75% of the embedding dimensions are distinct. The representations remain distinguishable.

**Visual: How Multi-Head Reduces Collision Damage**

```
                    SINGLE HEAD vs MULTI-HEAD HASHING


  SINGLE HEAD (K=1):


  "the milky way"
                        → slot 847,374  → [shared embedding]
  "the red apple"                           (CONTAMINATED)

  Problem: 100% of embedding is shared on collision



  MULTI-HEAD (K=8):


  "the milky way"    Head 1   Head 2   Head 3   Head 4   ...   Head 8
```

```
       → [847K]  [183K]  [2.9M]  [501K]   ...    [384K]
                ↓        ↓        ↓        ↓              ↓
             [ e      e       e       e     ...     e  ]
                ↑                  ↑
            collision           unique
                ↓                  ↓
             [ e      e'      e'      e     ...     e' ]
                ↓        ↓        ↓        ↓              ↓
       → [847K]  [2.8M]  [918K]  [501K]   ...    [2.9M]

"the red apple"     Head 1   Head 2   Head 3   Head 4   ...   Head 8

Result: Only 2/8 = 25% of embedding dimensions collide
        75% of representation is DISTINCT



COLLISION PROBABILITY:

  K=1:  P(full collision) = 1/M            3×10
  K=8:  P(full collision) = (1/M)^8        10 ²

The multi-head design makes "catastrophic collision" essentially
impossible while gracefully degrading on partial collisions.
```

**Theoretical Foundation: Established Techniques**

Multi-head hashing is a variant of several established methods:

**1. Feature Hashing (The "Hashing Trick")** Introduced by Weinberger et al. (2009) for high-dimensional sparse features:

$$\phi(x) = \sum_j \xi(j) \cdot x_j \cdot e_{h(j)}$$

Where $h()$ is a hash function and $\xi()$ is a sign function to reduce bias. Used extensively in large-scale ML (Vowpal Wabbit, scikit-learn's HashingVectorizer).

**2. Count-Min Sketch** A probabilistic data structure using multiple hash functions to estimate frequencies (Cormode & Muthukrishnan, 2005):

```
h   h   h   h

 3   0   1   2    ← row 1

 1   4   0   1    ← row 2

 2   1   3   0    ← row 3
```

```
Estimate = min(counts across all hash positions)
```

The minimum across heads gives a collision-robust estimate.

**3. Bloom Filters** Test set membership with multiple hash functions—false positives possible, false negatives impossible. Same probabilistic principle: collision in all $K$ hashes is exponentially unlikely.

**4. Random Projections (Johnson-Lindenstrauss)** The JL lemma guarantees that random projections preserve pairwise distances (Johnson & Lindenstrauss, 1984):

$$\|f(x) - f(y)\|_2 = (1 \pm \epsilon)\|x - y\|_2$$

With high probability, for appropriate target dimension.

#### Why Semantically-Uninformed Hashing Works

The hash functions are **semantically uninformed**—"milky way" (galaxy) and "milky way" (candy bar) might partially collide, while "milky way" and "andromeda galaxy" won't benefit from any shared structure.

But this isn't a bug, it's a feature:

1. **Semantic similarity is handled by the Transformer backbone**, not Engram
2. **Engram stores surface-level N-gram patterns**, which are about co-occurrence, not meaning
3. **The gating mechanism provides semantic filtering** after retrieval

The hash function's job is simply to provide a *consistent*, *deterministic*, *uniform* mapping. It doesn't need to be smart—it just needs to avoid systematic bias.

#### The Mathematical Guarantee

For $K$ independent hash functions with table size $M$, the expected number of "clean" dimensions (no collision with any other active N-gram) follows a balls-into-bins analysis.

If there are $n$ active N-grams in a batch:

$$E[\text{collisions per head}] = n - M\left(1 - \left(1 - \frac{1}{M}\right)^n\right) \approx \frac{n^2}{2M}$$

For typical batch sizes ($n \sim 4096$ tokens $\times$ 2 N-gram orders $= 8192$) and $M = 3M$:

$$E[\text{collisions per head}] \approx \frac{8192^2}{2 \times 3 \times 10^6} \approx 11$$

So ~11 collisions per head per batch, but the probability of the same pair colliding across all 8 heads is negligible.

#### The Gating Mechanism as Second Defense

Even when partial collisions occur, the context-aware gating provides semantic filtering:

```
# Retrieved embedding might contain collision noise
e_t = retrieve_ngram_embedding(context)  # Partially contaminated

# But the Transformer hidden state knows the true context
h_t = transformer_layers(input)  # Has global context

# Gating checks: "Does this retrieved memory match my context?"
alpha = sigmoid(dot(h_t, W_K @ e_t) / sqrt(d))

if context_matches_memory:
    alpha → 1.0  # Use the memory
else:
    alpha → 0.0  # Suppress (probably collision noise)
```

This is why Figure 7 in the paper (gating visualization) shows selective activation—the model learns to ignore retrieved embeddings when they don't match the actual context.

#### Summary Assessment

| Aspect | Assessment |
|---|---|
| Is it dimensionality reduction? | Yes—from $\lvert V\rvert^N$ to $K \times M$ dimensions |
| Is it semantically informed? | No—the hash function is semantically uninformed, but the design is principled |
| Is it lossy? | Yes, but multi-head makes catastrophic loss exponentially unlikely |
| Is it novel? | No—it's a neural adaptation of feature hashing / count-min sketch |

| Aspect | Assessment |
|---|---|
| Why does it work? | Probabilistic guarantees + gating mechanism + Zipfian sparsity of N-grams |

The semantically-uninformed mapping is actually a strength: it requires no learning, is deterministic (enabling prefetching), and provides theoretical guarantees on collision rates. The semantic heavy lifting is delegated to the Transformer backbone and the learned gating mechanism.

---

## 4. Context-Aware Gating: Lightweight Dynamic Filtering

### The Fundamental Tension

The paper presents a dichotomy:

- **Static memory**: $O(1)$ lookup, no computation, context-blind
- **Dynamic computation**: $O(d^2)$ per layer, full expressivity, context-aware

But pure static lookup has a fatal flaw: **the same N-gram can mean different things in different contexts.**

```
"The bank was steep"         → riverbank (geography)
"The bank was closed"        → financial institution
"The bank shot was perfect" → billiards term
```

All three share the bigram "The bank", which would retrieve the **same static embedding**. Without some mechanism to disambiguate, Engram would inject irrelevant or contradictory information.

### The Gating Computation

#### The Static Component (Context-Independent)

```
# These depend ONLY on the N-gram hash-identical every time "the bank" appears
e_t = lookup_ngram_embedding(token_ids)      # Static:  ^{d_mem}
k_t = W_K @ e_t                               # Static:  ^d (linear projection)
v_t = W_V @ e_t                               # Static:  ^d (linear projection)
```

At this point, everything is deterministic. Given the same input tokens, you get identical $k_t$ and $v_t$ regardless of surrounding context.

#### The Dynamic Component (Context-Dependent)

```
# h_t comes from preceding Transformer layers-FULLY context-dependent
h_t = transformer_output[t]  # Has seen full sequence via attention

# The gate computation is where static meets dynamic
q_t = RMSNorm(h_t)           # Normalized query (dynamic)
k_t_norm = RMSNorm(k_t)      # Normalized key (static)

# Scalar attention score
alpha_t = sigmoid(dot(q_t, k_t_norm) / sqrt(d))  #   (0, 1)

# Gated output
v_tilde = alpha_t * v_t      # Dynamic scaling of static vector
```

**Geometric Interpretation** The output $\tilde{v}$ lives on a **one-dimensional ray** in embedding space:

```
v_t (static direction)
 ↑

    =1.0 (full activation)

           →

    =0.5 (partial)
```

```
                    →

                  0 (suppressed)

             origin
```

**The direction is fixed; only the magnitude varies.** This is fundamentally less expressive than full dynamic computation, where both direction and magnitude are context-dependent.

### Comparison: Gating vs. Full Dynamic Computation

#### Full Attention (What Transformers Do)

```
# Every aspect is context-dependent
Q = W_Q @ H          # Queries from all positions
K = W_K @ H          # Keys from all positions
V = W_V @ H          # Values from all positions

attention = softmax(Q @ K.T / sqrt(d))  # Full N×N interaction
output = attention @ V                   # Weighted combination of ALL values
```

**Expressivity**: Output can be any linear combination of value vectors, with weights determined dynamically by full sequence context.

**Cost**: $O(N^2 \cdot d)$ for sequence length $N$, dimension $d$

#### Full FFN (What MoE Experts Do)

```
# Arbitrary nonlinear transformation
hidden = activation(W_1 @ x + b_1)     # Up-project
output = W_2 @ hidden + b_2             # Down-project
```

**Expressivity**: Can approximate any continuous function (universal approximation)

**Cost**: $O(d \cdot d_{ff})$ where $d_{ff}$ is typically $4d$

#### Engram Gating (What This Paper Does)

```
# Scalar modulation of static vector
alpha = sigmoid(dot(h_t, k_t) / sqrt(d))  # Single dot product
output = alpha * v_t                       # Scalar multiplication
```

**Expressivity**: Output constrained to ray defined by $v_t$; only magnitude varies

**Cost**: $O(d)$ for the dot product + $O(d)$ for the scaling = $O(d)$

### Cost Comparison Table

| Mechanism | FLOPs per Token | Expressivity |
|---|---|---|
| Full Attention | $O(N \cdot d)$ | Any weighted combination of values |
| FFN Layer | $O(d \cdot d_{ff}) \approx O(4d^2)$ | Universal function approximation |
| MoE Expert | $O(d \cdot d_{ff}/\text{num\_experts})$ | Same, but sparse |
| **Engram Gating** | $O(d)$ | **Scalar scaling of fixed direction** |

Engram gating is **orders of magnitude cheaper** but **dramatically less expressive**.

### The Computational Hierarchy

Engram creates a **tiered computation strategy**:

```
                    COMPUTATIONAL HIERARCHY


  TIER 1: Static Lookup (O(1))
```

```
• Hash N-gram → retrieve embedding
• Zero computation, pure memory access
• Handles: stereotyped patterns, named entities, collocations

                         ↓

TIER 2: Lightweight Gating (O(d))

• Single dot product + sigmoid
• Filters out irrelevant/colliding retrievals
• Handles: polysemy, hash collisions, context mismatch

                         ↓

TIER 3: Full Transformer Computation (O(d^2))

• Attention + FFN/MoE
• Full dynamic reasoning
• Handles: composition, inference, long-range dependencies
```

The hypothesis is that **many tokens don't need Tier 3 processing**—they're predictable from local context and can be handled by Tiers 1+2. This frees Tier 3 capacity for tokens that actually require reasoning.

**Expressivity Limitations of Gating**

**1. Cannot Change Direction**

```
Context A: "The river bank was steep"
Context B: "The investment bank was profitable"

Both retrieve: v_t = [0.3, -0.2, 0.8, ...]  (same static vector)

Context A gating:  _A * v_t = 0.9 * [0.3, -0.2, 0.8, ...]
Context B gating:  _B * v_t = 0.7 * [0.3, -0.2, 0.8, ...]

The outputs point in THE SAME DIRECTION-only magnitude differs.
The model cannot rotate "bank" toward financial vs. geographical meanings.
```

**Who handles this?** The Transformer backbone. The Engram contribution is added residually, and subsequent attention layers can still distinguish contexts.

**2. Cannot Compose Information**

```
"The large red ball"

Engram retrieves:
  - bigram ("large", "red") → e_1
  - bigram ("red", "ball") → e_2
  - trigram ("large", "red", "ball") → e_3

But these are independent lookups-Engram cannot compute
the COMPOSITIONAL meaning "a ball that is both large and red"
beyond what's stored in e_3.
```

**Who handles this?** The Transformer's attention mechanism, which can dynamically compose features across positions.

**3. Cannot Reason**

```
"If it's raining, then the ground is wet. It's raining. Therefore..."

No static N-gram lookup can complete this-it requires:
  - Understanding conditional structure
```

```
- Applying modus ponens
- Generating "the ground is wet"
```

**Who handles this?** The full Transformer stack, which the paper argues now has more "effective depth" because it's not wasting early layers on pattern matching.

#### The Spectrum of Staticness

```
FULLY STATIC                                                 FULLY DYNAMIC



 Raw       Engram     Engram      Per-head       Full
 N-gram    (single    (multi-     Gating         Attention
 Lookup    gate)      branch)     (alternative)

 e_t        ·v_t     Σ  _m·v_t  Σ  _k·e_k      softmax(QK)·V

 O(1)      O(d)       O(M·d)      O(K·d)         O(N·d)

 No        1D ray     M rays      K-dim          Full span
 control              (M=4)       subspace       of values
```

Engram with multi-branch gating sits at a sweet spot: **enough dynamism to filter collisions and polysemy, cheap enough to not defeat the purpose of static lookup.**

#### Alternative Design: Per-Head Gating

The current architecture uses a single scalar gate for the entire retrieved embedding:

```
e_t = concat([e_head_1, e_head_2, ..., e_head_K])  # All heads retrieved
alpha = compute_gate(h_t, e_t)                      # Single scalar
output = alpha * project(e_t)                       # All-or-nothing
```

An alternative would be **per-head gating:**

```
e_heads = [e_head_1, e_head_2, ..., e_head_K]       # All heads retrieved
alphas = [compute_gate(h_t, e_k) for e_k in e_heads]  # K separate gates
output = concat([ _k * e_k for  _k, e_k in zip(alphas, e_heads)])
```

**Why this could work:**

1. **Selective collision filtering**: If head 3 collided but heads 1,2,4-8 didn't, you could suppress only head 3
2. **Feature-specific modulation**: Different heads might capture different aspects
3. **Smoother expressivity gradient**: $K$-dimensional control surface instead of scalar

**Why the paper doesn't do this:**

1. **Computational cost**: $K$ separate dot products instead of 1
2. **Multi-branch integration already provides this**: mHC with $M = 4$ branches offers similar granularity
3. **Empirical sufficiency**: The ablations suggest current design works well enough

#### Empirical Validation: Does Gating Actually Discriminate?

Figure 7 in the paper shows gating activation qualitatively:

```
"Only Alexander the Great could tame the horse Bucephalus."

Gating activation (  values):
  "Alexander"     → low (not end of pattern)
  "the"           → low
  "Great"         → HIGH (completes "Alexander the Great")
  "could"         → low
  "tame"          → low
  "the"           → low
  "horse"         → low
  "Bucephalus"    → medium (entity, but less stereotyped)
```

The gating successfully identifies where static patterns END—which is exactly where the retrieved embedding is most reliable (the full N-gram was seen in training). At positions mid-pattern or on novel combinations, gating suppresses the retrieval.

---

## 5. mHC-Engram Integration: Expressivity-Efficiency Tradeoffs

### Manifold-Constrained Hyper-Connections (mHC) Background

The paper assumes familiarity with mHC (Xie et al., 2025).

**Standard Residual Connection** Traditional Transformers use a single residual stream:

$$h^{(\ell+1)} = h^{(\ell)} + f(h^{(\ell)})$$

All information flows through one pathway. The residual connection preserves the input, and $f()$ adds new information.

**Hyper-Connections (HC)** Hyper-Connections expand this to $M$ parallel branches with learnable mixing:

```
                    STANDARD RESIDUAL vs HYPER-CONNECTIONS


  STANDARD RESIDUAL (M=1):

       h                  → h + f(h)
                                              ↑
                           → f(·)



  HYPER-CONNECTIONS (M=4):

       h                       → h '
                             learnable
       h              mixing        → h '
                                weights
       h                       → h '

       h                       → h '
                                      ↓
                      → f(·)

  Each output h_m' is a learned combination of:
    • All input branches h ...h
    • The transformation output f(combined_input)
```

**The "Manifold-Constrained" Part** The key innovation in mHC is constraining the connection weights to lie on a manifold that preserves certain geometric properties (like gradient flow stability). Without this constraint, having $M$ branches with arbitrary mixing weights can cause training instabilities.

Mathematically:

$$H_{\text{out}} = A \cdot H_{\text{in}} + B \cdot f(C \cdot H_{\text{in}})$$

where $A, B, C$ satisfy manifold constraints ensuring stable gradients.

**The Engram-mHC Integration Design**

**What's Shared (Efficiency)**

```python
# ONE embedding table for all branches
E = shared_ngram_embedding_table  # Massive: billions of parameters

# ONE value projection for all branches
W_V = shared_value_projection     # Shape: d × d_mem

# The expensive retrieval happens ONCE
e_t = E[hash(ngram)]              # O(1) lookup, same for all branches
v_t = W_V @ e_t                   # O(d × d_mem), computed once
```

**What's Separate (Expressivity)**

```python
# M DIFFERENT key projections, one per branch
W_K = [W_K_1, W_K_2, W_K_3, W_K_4]  # M separate matrices

# Each branch computes its OWN gate
for m in range(M):
    k_t_m = W_K[m] @ e_t                                 # Branch-specific key
    alpha_m = sigmoid(dot(h_t[m], k_t_m) / sqrt(d))  # Branch-specific gate
    u_t[m] = alpha_m * v_t                                # Branch-specific output
```

**Visual Representation of the Integration**

```
                    ENGRAM × mHC INTEGRATION


                         N-gram tokens




                       Hash + Lookup    ← SHARED (O(1), one retrieval)
                       E[hash(·)]




                         e_t (static embedding)






          W_V          W_V          W_V     ← SHARED (one projection)
        (shared)     (shared)     (shared)




           v_t          v_t          v_t       (identical value vectors)




          W_K_1        W_K_2        W_K_3     ← SEPARATE (M projections)




           k_t_1        k_t_2        k_t_3     (different key vectors)



        h_t[1]        h_t[2]       h_t[3]     ← From mHC branches
```

```
    (query)      (query)     (query)

 _1 =  (q·k_1)  _2 =  (q·k_2)  _3 =  (q·k_3)  SEPARATE gates

    _1 · v_t      _2 · v_t      _3 · v_t    (different magnitudes)

   Branch 1      Branch 2      Branch 3     → to mHC residual
```

**The Expressivity-Efficiency Spectrum**

```
MAXIMUM EFFICIENCY                                         MAXIMUM EXPRESSIVITY
(Minimum Parameters)                                       (Maximum Parameters)
```

| Config A | Config B | Config C (PAPER'S CHOICE) | Config D | Config E |
|---|---|---|---|---|
| • 1 embed table<br>• 1 W_V<br>• 1 W_K<br>• 1 gate (shared) | • 1 embed table<br>• 1 W_V<br>• M W_K<br>• 1 gate (shared) | • 1 embed table<br>• 1 W_V<br>• M W_K<br>• M gates<br>• 1 conv | • 1 embed table<br>• M W_V<br>• M W_K<br>• M gates<br>• M conv | • M embed tables<br>• M W_V<br>• M W_K<br>• M gates<br>• M conv |
| All branches get identical contribution | Same key space, but branches vote together | Different gates, same value direction | Different value projections per branch | Completely independent per branch |
| Params: ~P | Params: ~P+Md² | Params: ~P+Md² | Params: ~P+2Md² | Params: ~M×P |
| Expressivity: 1 scalar for all | Expressivity: 1 scalar (joint) | Expressivity: M scalars (independent) | Expressivity: M vectors (M rays) | Expressivity: M independent vectors |

**The Tradeoff Relationship**

The relationship is partially inverse, but with nuance:

**The inverse relationship:**

```
                    Engram Contribution

     Pure Static                      Pure Dynamic
     (no gating)                      (full attention)


     Fixed embedding          Query-dependent
     Same for all           selection over all
     contexts                 memory positions
```

```
    O(1) cost                          O(N·d) cost
    No expressivity                    Full expressivity


                         Gating
                        (O(d))

                       Intermediate
                       cost &
                       expressivity
```

**Orthogonal scaling axes:**

The paper's design achieves **orthogonal scaling**:

- mHC scales expressivity via **branch parallelism** (more information pathways)
- Engram scales capacity via **memory size** (more stored patterns)

These are *somewhat* independent axes:

```
                         Memory Size (Engram)

                     Low               High



       Low      Small,          Large
                simple          memory,
   Branches     model           simple
   (mHC)                        routing


       High     Small           PAPER'S
                memory,         CHOICE
                complex         (27B total)
                routing
```

**Available Tuning Knobs**

1. **Number of branches** $M$: More branches = more expressive gating, more parameters
2. **What's shared vs. separate**: Could separate $W_V$ for direction control per branch
3. **Gating granularity**: Vector gating instead of scalar for per-dimension control
4. **Hash heads per branch**: Different heads for different branches

---

## 6. Memory-Compute Decoupling: Training vs. Inference Strategies

### Training: Distributed HBM Sharding

The paper describes training system design:

> "During training, to accommodate large-scale embedding tables, we employ standard model parallelism by sharding the tables across available GPUs. An All-to-All communication primitive is used to gather active rows in the forward pass and dispatch gradients in the backward pass."

This is **not** offloading to host memory. During training:

- The embedding tables are split across multiple GPUs
- Each GPU holds $1/N$ of the table in its HBM
- All2All communication gathers the needed embeddings
- Gradients flow back via the same All2All primitive

```
                         TRAINING: DISTRIBUTED SHARDING
```

```
 GPU 0 (HBM)          GPU 1 (HBM)          GPU 2 (HBM)          GPU 3 (HBM)

  E[0:M/4]          E[M/4:M/2]          E[M/2:3M/4]         E[3M/4:M]

  Shard 0             Shard 1             Shard 2             Shard 3




                              All-to-All
                           Communication


                   Each GPU receives embeddings it needs
                   for its local batch of tokens

MEMORY REQUIREMENT: Total Engram params / Number of GPUs per GPU
STILL IN HBM: Yes, distributed but still on-device
```

**Training still requires substantial HBM**—it's just distributed.

**Inference: Host Memory Offloading with Prefetching**

The dramatic memory savings apply to inference:

> "During inference, this deterministic nature enables a prefetch-and-overlap strategy. Since memory indices are known prior to the forward pass, the system can asynchronously retrieve embeddings from abundant host memory via PCIe."

```
                     INFERENCE: HOST MEMORY OFFLOADING


                         HOST MEMORY (DRAM)


                       ENGRAM EMBEDDING TABLE
                            (100B params)

                       ~200GB at FP16
                       (abundant, cheap)




                             PCIe Transfer
                          (async, prefetched)




                             GPU (HBM)



                        Transformer Backbone
                        (4B-8B params)
                        ~8-16GB




                        Prefetch Buffer            ← Small buffer for
```

```
                         (active embeddings)          current batch
                         ~few MB


                       Total HBM: ~10-20GB
                       (NOT 100GB+)
```

#### Why Prefetching Works for Inference But Not Training

The critical enabler is **deterministic addressing**:

**Inference**: Indices Known in Advance

```
# At inference time, the input sequence is fixed
input_tokens = [1847, 29341, 8472, 15234, 31847]

# We can compute ALL Engram indices BEFORE the forward pass
ngram_indices_layer_2 = compute_hashes(input_tokens, layer=2)
ngram_indices_layer_15 = compute_hashes(input_tokens, layer=15)

# Start prefetching while Layer 1 computes
async_prefetch(ngram_indices_layer_2)  # PCIe transfer begins

# By the time we reach Layer 2, embeddings are already in GPU buffer
layer_1_output = transformer_layer_1(input)
engram_embeddings = await_prefetch()  # Already arrived!
layer_2_output = engram_layer_2(layer_1_output, engram_embeddings)
```

**Training**: Gradient-Dependent Updates

```
# During training, we need gradients for EVERY accessed embedding
forward:
    e_t = E[hash(ngram)]  # Retrieved from host?
    loss = compute_loss(model(e_t))

backward:
    grad_e_t = d_loss / d_e_t  # Need to send gradient BACK to host
    E[hash(ngram)] -= lr * grad_e_t  # Update in host memory

# Problem 1: PCIe bandwidth for gradients back to host
# Problem 2: Optimizer states (Adam momentum, variance) where do they live?
# Problem 3: Gradient accumulation across distributed batches
```

Training requires bidirectional, latency-sensitive communication that doesn't tolerate PCIe bottlenecks as gracefully.

#### Memory Accounting Examples

**Traditional 100B model**:

$$100\text{B params} \times 2 \text{ bytes (FP16)} = 200\text{GB HBM}$$

Requires 3× H100 80GB (tensor parallel) or 2× H200 141GB.

**Engram 100B (8B backbone + 92B Engram)**:

| Component | Location | Size |
|---|---|---|
| Transformer Backbone | GPU HBM | ~16GB |
| Engram Tables | Host DRAM | ~184GB |
| **Total GPU HBM** | | **~16GB** |

Can run on 1× RTX 4090 24GB + 256GB host RAM.

**Important caveat**: Not equivalent models! The 8B backbone limits reasoning depth. But for knowledge retrieval tasks, may be competitive.

#### Key Insight: Breaking the 1GB/1B Rule

The traditional rule of thumb (1GB HBM per 1B parameters) doesn't apply to Engram inference because:

1. Engram parameters can reside in host DRAM
2. Prefetch-overlap masks PCIe latency
3. Zipfian caching further reduces effective latency

This enables running larger parameter models on memory-constrained hardware, but the Engram parameters provide "memory capacity" not "reasoning depth."

---

## 7. The U-Shaped Allocation Curve: Empirical Findings and Open Questions

### What The Paper Says

> "This observed U-shape confirms the structural complementarity between the two modules: - **MoE-dominated ($\rho \to 100\%$)**: The model lacks dedicated memory for static patterns, forcing it to inefficiently reconstruct them through depth and computation. - **Engram-dominated ($\rho \to 0\%$)**: The model loses conditional computation capacity, hurting tasks that require dynamic, context-dependent reasoning; memory cannot replace computation in this regime."

### What The Paper Doesn't Explain

1. **A formal model** predicting the optimal allocation ratio
2. **An explanation** for why the optimum is at ~75-80% specifically (rather than 50% or 90%)
3. **Analysis of the curve shape**—why U-shaped rather than V-shaped, linear, or asymmetric?
4. **Scale dependence**—does the optimal $\rho$ shift as total parameters increase?
5. **Task dependence**—is $\rho^*$ different for knowledge-intensive vs. reasoning tasks?

### Plausible Hypotheses

**Hypothesis 1: Diminishing Returns on Each Axis** Both MoE experts and Engram slots likely have diminishing returns following power laws:

$$L_{\text{MoE}}(n) = A \cdot n^{-\alpha}$$

$$L_{\text{Engram}}(m) = B \cdot m^{-\beta}$$

Under fixed parameter budget $P_{\text{sparse}}$:

$$n \cdot p_{\text{expert}} + m \cdot p_{\text{slot}} = P_{\text{sparse}}$$

The U-shape emerges if both functions are convex and the coefficients create an interior optimum.

**Hypothesis 2: Functional Specialization with Coverage Requirements** If ~30-40% of tokens are predictable from local patterns and ~60-70% require reasoning, optimal allocation should roughly match this split. The paper's finding ($\rho^* \approx 0.75\text{-}0.80$) is slightly higher, suggesting MoE is somewhat less efficient at its task than Engram at its task.

**Hypothesis 3: Hash Collision Saturation** Beyond some point, additional Engram capacity hits diminishing returns due to training signal sparsity for rare N-grams, while MoE experts can generalize across inputs.

### Assessment

The U-shaped curve is an **empirical finding**, not a **derived scaling law**. The paper provides no predictive equations for optimal allocation in new configurations. Practitioners can use $\rho \approx 0.75\text{-}0.80$ as a starting point but have no principled way to adjust for specific use cases without running their own sweeps.

---

## 8. Scaling Experiments: Zero-Sum Allocation vs. Additive Scaling

**Experiment 1: Zero-Sum Allocation (Section 3.1)**

**Setup**: Fixed total parameters $P_{\text{tot}}$, fixed activated parameters $P_{\text{act}}$. Sweep allocation ratio $\rho$.

**Finding**: U-shaped loss curve with optimal $\rho \approx$ 75-80%. Reallocating ~20-25% of sparse budget from MoE to Engram improves performance.

```
                    EXPERIMENT 1: ZERO-SUM ALLOCATION


Total Sparse Budget: 10B parameters (fixed)

 = 100%:  MoE gets 10B     Engram gets 0B     → Loss: 1.7248
 = 80%:   MoE gets 8B      Engram gets 2B    → Loss: 1.7109 (optimal)
 = 60%:   MoE gets 6B      Engram gets 4B    → Loss: ~1.715
 = 40%:   MoE gets 4B      Engram gets 6B    → Loss: ~1.725

KEY INSIGHT: Trading MoE capacity for Engram capacity improves loss
             up to a point, then hurts.
```

**Interpretation**: Under fixed constraints, some Engram is better than pure MoE. This is genuine reallocation—MoE capacity decreases as Engram increases.

**Experiment 2: Additive Scaling (Section 3.2)**

**Setup**: Fixed MoE backbone ($P_{\text{tot}} \approx$ 3B, $P_{\text{act}} = 568$M). Sweep Engram capacity from 0.3B to 13B.

**Finding**: Log-linear scaling—loss decreases linearly with log(slots). No saturation observed.

```
                    EXPERIMENT 2: ADDITIVE SCALING


Fixed MoE Backbone: 3B parameters (UNCHANGED across all runs)

Config A:  MoE: 3B  +  Engram: 0.3B   = Total: 3.3B   → Loss: ~1.81
Config B:  MoE: 3B  +  Engram: 1B     = Total: 4B     → Loss: ~1.78
Config C:  MoE: 3B  +  Engram: 5B     = Total: 8B     → Loss: ~1.76
Config D:  MoE: 3B  +  Engram: 13B    = Total: 16B    → Loss: ~1.74

The PROPORTION of Engram increases (9% → 81%), but this is because
the denominator grows, NOT because MoE shrinks.

NOTHING IS OFFLOADED-MoE capacity is preserved.
```

**Interpretation**: With unconstrained memory, more Engram is always better (within tested range). Nothing is "offloaded"—MoE is unchanged, Engram is added.

**Critical Distinction**

| Aspect | Experiment 1 | Experiment 2 |
|---|---|---|
| MoE params | Decreases as $\rho \downarrow$ | Constant |
| Engram params | Increases as $\rho \downarrow$ | Increases |
| Total params | Constant | Increases |
| Nature | Reallocation | Addition |

### The OverEncoding Comparison

OverEncoding (Huang et al., 2025a) also uses hash-based N-gram embeddings but:

- Injects at layer 0 only (no prefetch overlap)
- Uses fixed averaging (no gating)
- No tokenizer compression

```
                    OVERENCODING vs ENGRAM


OVERENCODING:

• N-gram embeddings retrieved at INPUT LAYER (Layer 0)
• Integration: AVERAGING with vocabulary embedding
• No gating mechanism
• No context-awareness

    input_embedding = 0.5 * vocab_embed[token] + 0.5 * ngram_embed[hash]

ENGRAM:

• N-gram embeddings retrieved at INTERMEDIATE LAYERS (e.g., 2, 15)
• Integration: GATED RESIDUAL addition
• Context-aware gating (can suppress irrelevant retrievals)
• Tokenizer compression

    hidden = hidden + gate(context, ngram_embed) * project(ngram_embed)
```

Engram extracts more value from equivalent memory budget due to deeper injection, gating, and compression. The comparison demonstrates that *how* you integrate N-gram memory matters as much as *whether* you include it.

---

## 9. Strategic Publication Patterns and V4 Deployment Probability

### The Undertraining Signal

The paper explicitly acknowledges:

> "Finally, scaling to Engram-40B further reduces pre-training loss and improves performance across most benchmarks. Although it does not yet strictly dominate Engram-27B on every task, **this is likely an artifact of under-training**. We observe that **the training loss gap between Engram-40B and the baselines continues to widen towards the end of training**, suggesting that the expanded memory capacity has not yet fully saturated within the current token budget."

### Evidence of Undertraining

1. **Loss gap widening**: Engram-40B advantage over baselines increases toward training end
2. **Inconsistent benchmark dominance**: Engram-40B regresses on code tasks (HumanEval 38.4 vs Engram-27B 40.8)

### Historical Pattern: DeepSeek Publication → Deployment

| Innovation | Paper Date | Deployed In | Lag |
|---|---|---|---|
| DeepSeekMoE | Jan 2024 | V2 (May 2024) | ~4 months |
| MLA | May 2024 | V2 (May 2024) | <1 month |
| Aux-loss-free | Nov 2024 | V3 (Dec 2024) | ~1 month |
| mHC | Dec 2024 | V3 (Dec 2024) | <1 month |
| **Engram** | **Jan 2026** | **V4 (???)** | **???** |

**Pattern**: Every major DeepSeek paper has been deployed. No exception.

**The Strategic Publication Pattern**

```
                    RESEARCH PAPER AS STRATEGIC SIGNAL


   PHASE 1: INTERNAL R&D (Not Published)

   • Full-scale experiments at frontier compute
   • Production infrastructure development
   • Integration with existing model architecture (V3)
   • Iterative refinement based on internal benchmarks

                             ↓

   PHASE 2: ACADEMIC PUBLICATION (This Paper)

   • Establish intellectual priority
   • Validate core concepts at reduced scale
   • Describe (but don't fully benchmark) production design
   • Signal direction to recruit talent and shape field
   • Deliberately omit frontier-scale results

                             ↓

   PHASE 3: PRODUCT LAUNCH (V4 Announcement)

   • Reveal frontier-scale performance
   • Cite own paper as foundation
   • Competitors now 6-12 months behind
   • Academic paper provides legitimacy
```

**Probability Assessment**

| Scenario | Probability | Description |
|---|---|---|
| Full adoption | ~35% | Engram as described, scaled to 100B+ |
| Modified adoption | ~40% | Engram-like module with undisclosed modifications |
| Partial/optional | ~15% | Engram in specific variants only |
| Deferred/abandoned | ~10% | Internal issues prevent deployment |

**Combined probability of some adoption**: ~75%

**Expected timeline**: Based on historical patterns (papers 1-4 months before deployment), V4 announcement likely in **Q1-Q2 2026**.

---

## 10. Long-Context Enhancement: The Attention Capacity Mechanism

### The Experimental Design

The paper compares models with matched base quality (iso-loss) to isolate architectural effects from general capability differences. This methodology is itself a contribution—many long-context papers conflate these factors.

### Key Results: Iso-Loss Comparison

Both models at pre-training loss = 1.63:

| Metric | MoE-27B (50k) | Engram-27B (46k) | Δ |
|---|---|---|---|
| LongPPL - Book | 4.38 | 4.19 | -0.19 |
| LongPPL - Paper | 2.91 | 2.84 | -0.07 |
| LongPPL - Code | 2.49 | 2.45 | -0.04 |
| NIAH - Multi-Query | 84.2 | 97.0 | **+12.8** |
| Variable Tracking | 77.0 | 87.2 | **+10.2** |
| Frequent Words Extraction | 73.0 | 98.6 | **+25.6** |

**The Mechanism: Attention Capacity Hypothesis**

Standard Transformers use attention for both local patterns and long-range dependencies. At long contexts, these compete:

```
                    ATTENTION CAPACITY HYPOTHESIS


STANDARD TRANSFORMER:



                ATTENTION BUDGET

         Local Patterns    Global Patterns
         "the cat sat"     "earlier, John"
                           "mentioned that"
            ~60%                ~40%



    At long contexts, local patterns "crowd out" global attention.


ENGRAM TRANSFORMER:



                ATTENTION BUDGET

               Global Patterns Only
         "earlier, John mentioned that..."

                   ~90%




              ENGRAM (separate budget)

               Local Patterns Only
              "the cat sat", "New York"
                   O(1) lookup



    By handling local patterns separately, attention can focus
    entirely on long-range dependencies.
```

**Task-Specific Pattern**

| Task Type | Engram Advantage | Explanation |
|---|---|---|
| LongPPL (average perplexity) | Modest (+2-5%) | Most tokens are locally predictable |
| Single-needle retrieval | None (~0%) | Easy for both architectures |
| Multi-query retrieval | Large (+15%) | Attention-limited in MoE |
| Variable tracking | Large (+16%) | Requires multi-hop retrieval |
| Frequent word extraction | Huge (+36%) | Requires global scanning |

The pattern confirms the mechanism: Engram's advantage scales with how much the task requires global (vs. local) attention.

---

## 11. The Effective Depth Hypothesis: Bypassing Pattern Reconstruction

### The Claim

> "By equipping the model with an explicit knowledge lookup capability, Engram effectively mimics an increase in model depth by relieving the model of the early stages of feature composition."

### What Early Layers Do

Standard Transformers spend layers 1-6 progressively reconstructing static patterns. The paper cites Ghandeharioun et al. (2024) to illustrate with "Diana, Princess of Wales":

```
        ENTITY RESOLUTION: "Diana, Princess of Wales"
        (How a standard Transformer reconstructs a known entity)


  Input: "... Diana, Princess of Wales ..."
  Task: Internally represent who this entity is

  Layer    Latent State (via PatchScope)           What's Happening

  1-2      "Wales: Country in the United Kingdom"  Just the last token
  3        "Wales: Country in Europe"              Still just geography
  4        "Princess of Wales: Title held by       Starting to see title
            female sovereigns..."
  5        "Princess of Wales: Title given to      Title semantics emerge
            the wife of the Prince of Wales..."
  6        "Diana, Princess of Wales (1961-1997),  FINALLY: Full entity
            the first wife of Prince Charles..."


  OBSERVATION:
  It takes 6 LAYERS just to reconstruct a well-known entity.
  This is static knowledge-it's the same every time this phrase appears.
  These layers are essentially rebuilding a lookup table at runtime.

  ENGRAM ALTERNATIVE:
  Layer 2: Engram looks up trigram "Princess of Wales"
           → Retrieves pre-computed embedding encoding the full entity
  Layer 3+: Can immediately proceed to REASONING about Diana
```

### LogitLens Evidence

LogitLens (nostalgebraist, 2020) projects each layer's hidden state through the final LM head to measure "prediction readiness":

$$\mathrm{KL}(P^{(\ell)} \| P^{\mathrm{final}})$$

**Finding**: Engram shows systematically lower KL divergence at early layers—representations converge to prediction-ready states faster.

```
                  KL DIVERGENCE BY LAYER


   KL Divergence

   10

    8
                MoE-27B
    6

    4

    2

    0                         → Layer
      0    5    10   15   20   25   30

        MoE-27B          Engram-27B
```

#### CKA Evidence

CKA (Centered Kernel Alignment; Kornblith et al., 2019) measures representational similarity between layers.

**Finding**: Engram layer 5 representations match MoE layer ~12 representations for named entities. The "soft alignment index" quantifies this shift:

| Engram Layer $j$ | Soft Alignment $a_j$ | "Depth Bonus" $(a_j - j)$ |
|---|---|---|
| 0 | ~2 | +2 layers |
| 5 | ~12 | +7 layers |
| 10 | ~17 | +7 layers |
| 15 | ~21 | +6 layers |
| 20 | ~24 | +4 layers |
| 25 | ~27 | +2 layers |

Early-to-mid Engram layers gain the most "effective depth" because that's where static pattern reconstruction would normally occur.

#### The Mechanism Summarized

```
                  ENGRAM'S "EFFECTIVE DEPTH" MECHANISM


   STANDARD TRANSFORMER:

   Layer 0   → Raw token embeddings
   Layer 1   → Local bigram features
   Layer 2   → Trigram patterns emerging
   Layer 3   → Entity boundaries detected
   Layer 4   → Entity types recognized
   Layer 5   → Multi-token entities composed        ← RECONSTRUCTION
   Layer 6   → Entity semantics resolved             ← COMPLETE HERE
```

```
Layer 7   → Begin relational reasoning          ← REASONING STARTS
...
Layer 30  → Final prediction

ENGRAM TRANSFORMER:

Layer 0   → Raw token embeddings
Layer 1   → Basic contextualization
Layer 2   → ENGRAM INJECTION                    ← STATIC PATTERNS
                                                  RETRIEVED O(1)
            → Representations NOW equivalent to MoE Layer 6-7
Layer 3   → Can immediately begin reasoning     ← REASONING STARTS
...
Layer 30  → Final prediction

NET EFFECT:
Engram "skips" ~4-5 layers of reconstruction work.
Those layers can now contribute to reasoning instead.
A 30-layer Engram model has ~34-35 "effective layers" of reasoning.
```

---

## 12. Functional Separation and Domain-Specialized Engram Potential

### The Sensitivity Experiment

The experiment completely suppresses Engram output during inference ($\alpha \to 0$ for all positions).

#### Results by Task Category

```
                    RETAINED PERFORMANCE BY TASK TYPE


TASK CATEGORY              BENCHMARK        RETAINED     INTERPRETATION


READING COMPREHENSION      C3               93%          Backbone holds
(Context provides answer)  RACE-Middle      89%          comprehension
                           RACE-High        84%          capability
                           DROP             81%


COMMONSENSE REASONING      HellaSwag        85%          Backbone holds
(World knowledge needed)   ARC-Challenge    81%          most common-
                           PIQA             81%          sense patterns


KNOWLEDGE-INTENSIVE        CMMLU            78%          Moderate
REASONING                  MMLU             75%          degradation-
(Facts + reasoning)        MMLU-PRO         72%          needs both


CODE                       CruxEval         76%          Mixed-patterns
(Patterns + logic)         MBPP             68%          matter for
                           HumanEval        58%          generation


ALGORITHMIC REASONING      BBH              67%          Surprising
(Multi-step logic)         GSM8K            62%          dependency-
                           MGSM             44%          maybe via
```

```
                    MATH              36%          pattern recog


FACTUAL KNOWLEDGE   TriviaQA-ZH       44%          CATASTROPHIC
(Pure recall)       PopQA             44%          COLLAPSE
                    TriviaQA          29%          ← Engram IS
                                                     the knowledge
```

**Interpretation: Functional Separation**

Engram becomes the **primary repository for parametric knowledge**. Factual knowledge is inherently N-gram structured ("The capital of France" → "Paris"), making it ideal for static lookup. The backbone retains comprehension and reasoning capabilities.

```
                ARCHITECTURAL CAPABILITY MAPPING


                        FULL MODEL
                     (Backbone + Engram)




      BACKBONE                              ENGRAM
   (MoE Experts +                       (N-gram Memory)
    Attention)

 • Reading comprehens.                 • Factual knowledge
 • Commonsense reason.                 • Entity recognition
 • Context extraction                  • Pattern completion
 • Logical inference                   • Stereotyped phrases
 • Long-range deps                     • Local collocations


                     REQUIRES BOTH
                 • Knowledge reasoning
                      • Code generation
                      • Math problem solving
```

**Implication: Domain-Specialized Engram**

If Engram disproportionately stores factual knowledge, domain-specialized Engram modules could dramatically improve accuracy in knowledge-intensive applications:

**Medical Engram Potential**

```
                MEDICAL ENGRAM: POTENTIAL DESIGN


DOMAIN CHARACTERISTICS:
```

```
Medical text is HIGHLY N-GRAM STRUCTURED:

• Drug names: "acetaminophen", "metformin hydrochloride"
• Dosages: "500mg twice daily", "10mg/kg body weight"
• Conditions: "type 2 diabetes mellitus", "acute myocardial infarction"
• Anatomical terms: "left anterior descending artery"
• Procedures: "laparoscopic cholecystectomy"
• Interactions: "contraindicated with MAO inhibitors"

These are LOCAL, STATIC patterns-perfect for Engram.



POTENTIAL ARCHITECTURE:


Base Model:
  General backbone (reasoning, comprehension)        ~20B params
  General Engram (common knowledge)                  ~10B params

Medical Specialist:
  Same backbone                                      ~20B params
  General Engram (retained)                          ~10B params
  Medical Engram (domain-specific)                   ~50B params
    • Trained on PubMed, clinical notes, FDA data
    • Drug-drug interactions
    • Diagnostic criteria
    • Treatment protocols

Total: 80B params, but only 20B activated per token
Massive factual capacity with modest compute cost.
```

**Advantages of Domain-Specialized Engram**

1. **Efficiency**: Scales knowledge without scaling inference cost
2. **Modularity**: Swap domain modules without backbone retraining
3. **Updateability**: Incremental updates without catastrophic forgetting
4. **Auditability**: Deterministic retrievals enable knowledge provenance

**Future Direction: Mixture of Memories (MoM)**

```
        FUTURE DIRECTION: MIXTURE OF MEMORIES (MoM)


Just as MoE routes computation to specialized experts,
MoM could route retrieval to specialized memory modules:

                    Input Tokens




                  Memory Router
                  (Learned)
```

```
   General           Medical           Legal
   Engram            Engram           Engram
   (50B)             (50B)            (30B)



                    Gated Fusion



                     Backbone
                     (20B MoE)

Total: 150B+ parameters
Activated: ~25B per token
Specialized knowledge for each domain
Single backbone for shared reasoning
```

---

## 13. Scale Claims vs. Empirical Validation: Strategic Omissions

**Systematic Gap Analysis**

```
                 SCALE CLAIMS vs. EMPIRICAL VALIDATION

CLAIM                                   VALIDATED?    GAP

U-shaped allocation is stable           Partially     Only 2 compute
across compute regimes                                budgets tested
                                                      (2×10² , 6×10² )
                                                      V3 uses ~10²

Log-linear Engram scaling continues     Partially     Tested to ~13B
indefinitely                                          Claims apply to
                                                      100B+

100B Engram offloading with <3%         Throughput    Capability NOT
overhead                                only          tested at 100B

Multi-level cache hierarchy             Described     No empirical
exploits Zipfian distribution                         validation

Engram-40B undertraining implies        Claimed       No extended
further gains possible                                training run

Prefetch-overlap strategy scales        Architecture  Not tested at
to production serving                   described     high QPS/batch

Context-extension gains persist         32k context   No 128k or 1M
at longer contexts                      only          context testing
```

```
MoE + Engram composition optimal        Asserted      No comparison
for frontier models                                   at frontier
                                                      scale
```

**Evidence of Intentional Omission**

1. **Selective precision**: Extremely detailed on some aspects (hyperparameters to 4 decimals), conspicuously vague on scale
2. **Infrastructure over-specification**: Section 2.5 describes production-grade systems unnecessary for 27B validation
3. **Explicit undertraining acknowledgment**: They tell you results are incomplete
4. **Architecture alignment with V3**: Uses V3 tokenizer, MLA, mHC—not generic research

**The Competitive Timing Dimension**

```
                 COMPETITIVE TIMING ANALYSIS


IF DEEPSEEK PUBLISHES FULL FRONTIER RESULTS:

• OpenAI/Anthropic/Google immediately start replication
• US labs have more compute to iterate faster
• DeepSeek's head start erodes quickly

BY PUBLISHING CONCEPT WITHOUT SCALE VALIDATION:

• Establishes intellectual priority (can cite own work)
• Competitors must independently validate scale
• Buys time to ship V4 before replication
• When V4 launches, competitors are still experimenting

THE STRATEGIC CALCULUS:

• DeepSeek's advantage is NOT compute (they have less)
• DeepSeek's advantage IS architectural innovation speed
• Publishing establishes priority, withholding preserves lead time
• Optimal strategy: publish concept, withhold scale results

This is EXACTLY what the paper does.
```

---

# Future Implications

## The Beijing AGI Summit Context

At the January 2026 Beijing AGI summit, senior figures from Chinese AI labs made striking admissions (South China Morning Post, 2026; Wall Street Journal, 2026):

- **Zhipu AI co-founder Tang Jie**: "The truth may be that the gap is actually widening"
- **Alibaba scientist Lin Junyang**: "Less than 20%" chance of overtaking US in 3-5 years; US labs enjoy "one to two orders of magnitude more training compute"

External assessments support this pessimism (Asialink, 2026; Science Business, 2026):

- US controls ~75% of global AI compute; China ~15%
- US AI supercomputing capacity approximately 9× China's
- HBM (not just GPUs) identified as binding constraint (Design-Reuse, 2026)

## Engram as Strategic Response

The Engram architecture directly addresses acknowledged constraints:

| Constraint | Engram Response |
|---|---|
| HBM bottleneck | Host DRAM offloading via prefetch |
| Compute disadvantage | More capability per FLOP |
| Hardware access limits | Architectural efficiency as substitute for scale |

This alignment is not coincidental. Engram represents the **operationalization of efficiency as competitive strategy**.

## The Paradox of Constraint-Driven Innovation

Export controls may paradoxically accelerate Chinese architectural innovation:

- US labs with abundant compute have less pressure to innovate on efficiency
- Chinese labs, facing structural constraints, are forced into architectural creativity
- Innovations that extract more from less benefit everyone but are *discovered* under constraint

If US labs have 10× compute but DeepSeek extracts 2× more per FLOP, the effective gap narrows to 5×. With compounding efficiency innovations, this represents the scenario where China narrows the capability gap despite hardware disadvantage.

## DeepSeek V4 Prediction

### Evidence for Engram Adoption

1. **Track record**: Every major DeepSeek paper has been deployed
2. **Strategic fit**: Directly addresses HBM constraints and compute efficiency
3. **Architecture alignment**: Paper uses V3 components (tokenizer, MLA, mHC)
4. **Publication timing**: Matches historical pattern (paper 1-4 months before deployment)
5. **Scale validation gap**: Frontier results deliberately withheld

### Signals to Watch

**Strong adoption signals**: - Follow-up paper showing Engram at 100B+ scale - Production-grade distributed training code release - V4 technical report citing this paper in architecture section - DeepSeek communications mentioning "conditional memory"

**Weak/negative signals**: - Long delay between paper and any follow-up - Other DeepSeek papers explore alternative efficiency approaches - No mention of Engram in subsequent communications

## Research Directions

### Domain-Specialized Engram

Research questions: 1. Does domain-specific Engram training improve domain accuracy? 2. What's the optimal $\rho^*$ for knowledge-intensive domains? 3. Can domain and general Engram compose without interference? 4. Can Engram retrievals provide citations for knowledge provenance?

### Technical Extensions

- Integration with other sparse primitives (Mixture-of-Depths, early exit)
- Learned addressing (beyond fixed hash functions)
- Higher-order N-grams with larger memory budgets
- Continual learning for Engram modules

### Broader Implications

If Engram's thesis is correct—that Transformers "waste" depth on static pattern reconstruction—implications extend beyond efficiency:

- **Interpretability**: Cleaner separation between knowledge storage and reasoning
- **Editability**: Modify factual knowledge without affecting reasoning capabilities
- **Verification**: Audit knowledge sources via retrieval provenance

---

## References


Cormode, G., & Muthukrishnan, S. (2005). An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1), 58-75.

DeepSeek-AI. (2024a). DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

DeepSeek-AI. (2024b). DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*.

DeepSeek-AI. (2026). Engram: Conditional memory via scalable lookup—A new axis of sparsity for large language models. *arXiv preprint arXiv:2601.07372*.

Design-Reuse. (2026). China's AI chip ambitions limited by HBM memory supply, notes report. Retrieved from https://www.design-reuse.com/news/

Ghandeharioun, A., Kim, B., & others. (2024). PatchScope: A framework for inspecting language model representations at scale. *Proceedings of the 41st International Conference on Machine Learning*.

Huang, X., & others. (2025a). OverEncoding: Hash-based N-gram embeddings for vocabulary expansion. *arXiv preprint*.

Johnson, W. B., & Lindenstrauss, J. (1984). Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26, 189-206.

Jurafsky, D., & Martin, J. H. (2024). *Speech and language processing* (3rd ed.). Pearson.

Kornblith, S., Norouzi, M., Lee, H., & Hinton, G. (2019). Similarity of neural network representations revisited. *Proceedings of the 36th International Conference on Machine Learning*, 3519-3529.

Lample, G., Sablayrolles, A., Ranzato, M., Denoyer, L., & Jégou, H. (2019). Large memory layers with product keys. *Advances in Neural Information Processing Systems*, 32.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 9459-9474.

nostalgebraist. (2020). Interpreting GPT: The logit lens. *LessWrong*. Retrieved from https://www.lesswrong.com/posts/

Peng, B., Alcaide, E., Anthony, Q., Albalak, A., Arcadinho, S., Biderman, S., ... & Zheng, Y. (2023). YaRN: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*.

Reuters. (2026, January 10). China is closing US technology lead despite constraints, AI researchers say. Retrieved from https://www.reuters.com/world/china/

Science Business. (2026). State of AI 2025: Five key charts for Europeans. Retrieved from https://sciencebusiness.net/news/ai/

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27(3), 379-423.

South China Morning Post. (2026, January). China AI has less than 20% chance to exceed US over next 3-5 years: Alibaba scientist. Retrieved from https://www.scmp.com/tech/big-tech/

Wall Street Journal. (2026, January). China AI race: US chips. Retrieved from https://www.wsj.com/tech/ai/

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., & Attenberg, J. (2009). Feature hashing for large scale multitask learning. *Proceedings of the 26th International Conference on Machine Learning*, 1113-1120.

Xie, S., & others. (2025). mHC: Manifold-constrained hyper-connections for improved gradient flow in deep networks. *arXiv preprint*.