Tensorizing Engram: Sharing Latents Across N-Gram Embeddings is Beneficial in LLMs

Wuyang Zhou [1] **Yuxuan Gu** [1] **Giorgos Iacovides** [1] **Yuning Qiu** [2] **Qibin Zhao** [2] **Danilo Mandic** [1]

Abstract

Modern language models represent text using discrete token-level embeddings, which forces recurring multi-token patterns to be learned implicitly across Transformer layers. Both Over-tokenized Transformers and Engram attempt to address this limitation by explicitly incorporating multi-token (n-gram) memories. However, they rely on separate hash tables for each n-gram order, which introduces hash collisions and prevents nested n-grams from sharing the underlying latent structures. To address these issues, we propose **T**ensorized E**ngram** (TN-gram), a compact memory module that represents tensorized n-gram embeddings through shared factors in the Canonical Polyadic (CP) form. TN-gram learns shared token-position factors together with order-absorption vectors to encode the embeddings of different n-gram order. Comprehensive experiments demonstrate that TN-gram matches or even outperforms Engram-style n-gram modules while requiring much fewer parameters.

1. Introduction

Modern Language Models (LMs) (OpenAI, 2023; Team, 2024) represents text primarily through token-level embeddings (Brown et al., 2020; Devlin et al., 2019; Vaswani et al., 2017). While this embedding design is simple and scalable, it requires recurring multi-token (n-gram) patterns, such as phrases, collocations, and syntactic fragments, to be constructed implicitly across Transformer (Vaswani et al., 2017) layers. This is inevitably inefficient, as model capacity must be split between reasoning and recovering frequent local dependencies that are naturally contiguous n-gram contexts, i.e., short sequences of n adjacent tokens (Chen & Goodman, 1999; Cheng et al., 2026; Basharin et al., 2024; Goodman, 2001; Huang et al., 2025; Manning & Schutze, 1999).

To utilize n-gram contexts, Over-tokenized Transformer (Huang et al., 2025; Liu et al., 2026) expands the embedding vocabulary with higher-order n-gram tokens, while DeepSeek's Engram (Cheng et al., 2026) injects n-gram information through external memory layers, outperforming the Over-tokenized Transformer. These works have demonstrated that scaling explicit n-gram memories can be more effective than scaling the number of experts in the Mixture-of-Experts (MoE) layers (Shazeer et al., 2017; Zhou et al., 2022), and can improve the language modeling performance of LLMs without incurring much more FLOPs.

However, direct modeling of the whole n-gram space is prohibitive in terms of the curse of dimensionality, as n-gram embedding requires exponentially growing number of parameters, i.e., $V^n \times d$, where V is the vocabulary size, and d is the model dimension. Assume $V = 30,000$ and $d = 4096$, then even the simplest 2-gram embedding already requires 3.69 trillion parameters. Thus, existing works utilizing explicit n-gram memories in LLMs usually resort to hash tables to obtain tractable lookup tables, but suffer from hash collisions where distinct n-grams share the same slot (Cheng et al., 2026; Huang et al., 2025; Liu et al., 2026). Furthermore, since a n-gram has V^n possible token combinations, if n is large (e.g., $n = 6$), the model might not encounter all V^n combinations in training, leaving their embeddings in the hash tables completely untrained.

Additionally, the separate hash tables for different n-gram orders in existing works (Cheng et al., 2026; Huang et al., 2025; Liu et al., 2026) completely overlook an important intrinsic structure, i.e., an $(n-1)$-gram is always contained within its higher-order n-grams. For example, assume "New", "York", and "City" are three individual tokens and we use all three n-gram orders (1-gram, 2-gram, and 3-gram). Representing the phrase "New York City" at the last position requires the 1-gram embedding of "City", the 2-gram embedding of "York City", and the 3-gram embedding of "New York City". This nested structure naturally suggests enforcing shared latent components across em-

[1]Department of Electrical and Electronic Engineering, Imperial College London, London, United Kingdom [2]AIP, RIKEN, Tokyo, Japan. Correspondence to: Wuyang Zhou <wuyang.zhou19@imperial.ac.uk>.

ICML 2026 The 2nd Workshop on Connecting Low-rank Representations in AI, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

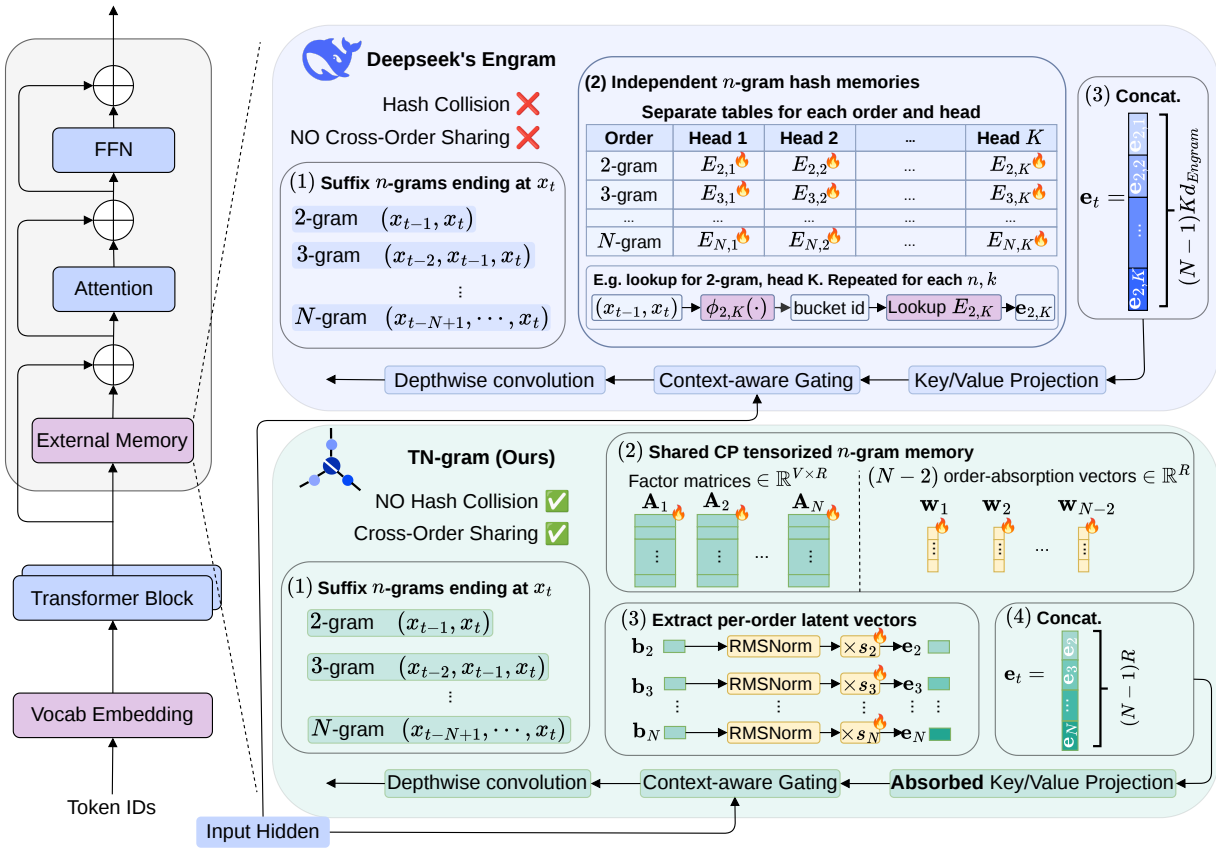

Figure 1. Architecture of Engram (Cheng et al., 2026) and the proposed TN-gram. Both methods inject explicit N-gram memory into selected Transformer blocks. Engram uses separate per-order, per-head hash tables $E_{n,k}$, while TN-gram replaces them with a tensorized CP memory module over shared factors ,$\mathbf{A}_1, \ldots, \mathbf{A}_N$, and order-absorption vectors, $\mathbf{w}_1, \ldots, \mathbf{w}_{N-2}$, enabling cross-order latent sharing within differernt n-gram orders.

beddings of different n-gram orders, which in turn enables training embeddings for higher-order n-grams even without encountering them in the training set. This motivates us to ask the following question:

Can we model the complete n-gram spaces without the drawbacks of conventional hash tables, by allowing the embeddings of different n-gram orders in the LLM to share common latent structures?

To answer this question, we propose the **T**ensorized **Engram** (**TN-gram**), which models the whole n-gram space in a tensorized format using compact tensor networks in order to avoid hash collisions introduced by hash tables. More specifically, we represent each n-gram order in their Canonical Polyadic (CP) decomposition format and enforce the sharing of CP components across n-gram orders. This shared higher-order low-rank structure allows for the training of the embeddings of unseen n-grams. Additionally, benefiting from the super-compression capabilities of tensor networks, this drastically reduces the number of parameters needed to model the complete n-gram space from $\mathcal{O}(V^n \times d)$, which scales exponentially with the n-gram order, to $\mathcal{O}(nVR + dR)$, which scales linearly in n under fixed tensor rank R.

In summary, our contributions are as follows:

- We introduce **TN-gram**, a tensorized n-gram memory architecture that models the complete n-gram space efficiently using CP tensor network representations instead of hash tables.
- TN-gram utilizes shared CP factors across the embeddings of different n-gram orders, effectively utilizing the intrinsic nested structure of multiple n-gram orders, i.e., an $(n-1)$-gram is always contained in the higher-order n-grams.
- Comprehensive experiments demonstrate that TN-gram achieves comparable or better performance and requires less parameters at the same n-gram order compared to hashing-based solutions such as Engram (Cheng et al., 2026).

2. Related Works

Scaling embedding layers and explicit n-gram memories. Recent work has explored scaling language model representations beyond standard token-level embeddings. Over-tokenized Transformer (Huang et al., 2025) expands the embedding table to include higher-gram tokens and then combines their embeddings with token embeddings, showing that local multi-token units can provide useful inductive bias. Engram (Cheng et al., 2026) instead reformulates explicit n-gram embedding as conditional memory, extracting suffix n-grams ending at the current token for n-gram orders 2 to N through deterministic multi-head hashing. Each extracted n-gram is hashed into an order-specific memory table, from which memory vectors are retrieved and projected into Key/Value representations. A context-aware gate compares the Key with the current hidden state and filters the Value before adding it to the residual stream. Similar to Cheng et al. (2026), authors in Liu et al. (2026) also discovered the effectiveness in scaling the embedding layers compared to scaling the experts in the Mixture-of-Experts (MoE) layers. These works motivate embedding scaling as an orthogonal parameter-allocation direction once expert scaling in MoE reaches diminishing returns. However, despite their effectiveness, these approaches are limited by the hashed memory design. Finite lookup tables can cause hash collisions (Liu et al., 2026) , and separate tables for each n-gram prevent related contexts from sharing parameters.

Tensor networks for LLMs. Tensor network methods are especially useful for achieving an efficient representation of large-scale neural networks due to their super-compressed capabilities. Early work showed that dense fully connected layers can be replaced with Tensor-Train tensor networks, significantly reducing parameters while preserving expressivity (Novikov et al., 2015). More recently, TensorLLM applies Tucker decomposition for post-training compression and denoising (Gu et al., 2025), while TensorGPT compresses embedding layers for deployment on edge-devices (Xu et al., 2023) using Tensor-Train decomposition. LoRETTA (Yang et al., 2024) and TeRA (Gu et al., 2026) use tensorized low-rank updates for parameter-efficient fine-tuning (PEFT), and KromHC (Zhou et al., 2026) represents the tensorized residual hyper-connection weights with Kronecker-structured matrices to reduce parameter complexity. Other studies analyze how tensor-network rank affects the efficiency-expressivity tradeoff (Zhou et al., 2025) and use LLMs to guide tensor network structure and rank search (Iacovides et al., 2025).

3. Preliminaries

Tensor notations. A tensor is a multi-dimensional array that generalizes vectors and matrices to higher orders (Cichocki et al., 2015). A scalar, vector, matrix and tensor is denoted as $a, \mathbf{a}, \mathbf{A}, \mathcal{A}$, respectively. An order-N tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ has N modes, with $\mathcal{T}(i_1, \ldots, i_N)$ denoting the entry at index $(i_1, \ldots, i_N)$ (Cichocki et al., 2016). The mode-n product multiplies a tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ with a matrix $\mathbf{A} \in \mathbb{R}^{J \times I_n}$ along its n-th mode, producing a tensor $\mathcal{T} \times_n \mathbf{A} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$. Its element-wise entry is given by

$$\begin{aligned}&(\mathcal{T} \times_n \mathbf{A})(i_1, \ldots, i_{n-1}, j, i_{n+1}, \ldots, i_N) \\ &= \sum_{i_n=1}^{I_n} \mathcal{T}(i_1, \ldots, i_n, \ldots, i_N)\mathbf{A}(j, i_n).\end{aligned} \tag{1}$$

Canonical Polyadic tensor network decomposition. A fundamental multi-linear tensor network for representing large tensors is the Canonical Polyadic (CP) decomposition (Kolda & Bader, 2009), which approximates $\mathcal{T}$ as a sum of R rank-one components as follows

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}^{(1)}(r) \circ \mathbf{a}^{(2)}(r) \circ \cdots \circ \mathbf{a}^{(N)}(r), \tag{2}$$

or in the element-wise format

$$\mathcal{T}(i_1, \ldots, i_N) = \sum_{r=1}^{R} \mathbf{A}^{(1)}(i_1, r)\mathbf{A}^{(2)}(i_2, r) \cdots \mathbf{A}^{(N)}(i_N, r), \tag{3}$$

where $\circ$ denotes the outer product, R is the tensor rank, $\mathbf{A}^{(k)} \in \mathbb{R}^{I_k \times R}$ is the factor matrix for mode k, and $\mathbf{a}_r^{(k)} \in \mathbb{R}^{I_k}$ is the r-th column of $\mathbf{A}^{(k)}$. CP decomposition reduces the number of parameters from $\prod_{k=1}^{N} I_k$ in the original $\mathcal{T}$ to $\sum_{k=1}^{N} I_k R$, where $R \ll I_k$, achieving super-compression via representing the original tensor in a higher-order low-rank format.

The n-gram concept in Engram. Traditionally, n-gram language models represent local dependencies in discrete token sequences by making a finite-context Markov assumption, where the probability of the next token depends solely on the preceding $n-1$ tokens (Manning & Schutze, 1999). While classical n-gram models (Bojanowski et al., 2017; Brants et al., 2007; Nguyen, 2024) compute these probabilities through empirical corpus counts, recent efforts such as Engram (Cheng et al., 2026) modernize this concept by formalizing n-grams as a conditional memory module for LLMs. More specifically, Engram uses the local context to perform $\mathcal{O}(1)$ lookups (Pagnoni et al., 2025; Tito Svenstrup et al., 2017; Yu et al., 2025) into an external embedding table. To circumvent the intractable parameter size of the complete combinatorial space, Engram adopts a multi-head hashing strategy. Given a maximum order N, Engram includes all context orders $n \in \{2, \ldots, N\}$. For each order n and head $k \in \{1, \ldots, K\}$, a deterministic hash function

maps the local context $x_{t,n}$ to an index, retrieving an embedding $\mathbf{e}_{t,n,k}$. The final memory vector $\mathbf{e}_t$ concatenates the retrieved embeddings across all n-gram orders and hash heads as follows

$$\mathbf{e}_t = \mathop{\Big\|}_{n=2}^{N} \mathop{\Big\|}_{k=1}^{K} \mathbf{e}_{t,n,k}, \tag{4}$$

where $||$ denotes concatenation and K is the number of hash heads.

Unlike Over-tokenized Transformer (Huang et al., 2025) which simply expands the embedding vocabulary beyond 1-gram, Engram introduces a context-aware gating mechanism to filter the retrieved static memory using the hidden state $\mathbf{h}_t$ at the current sequence position t. It computes Key and Value vectors via linear projections, i.e., $\mathbf{k}_t = \mathbf{W}_K \mathbf{e}_t$ and $\mathbf{v}_t = \mathbf{W}_V \mathbf{e}_t$, and then computes a scalar gate:

$$\alpha_t = \sigma\left(\frac{\mathrm{RMSNorm}(\mathbf{h}_t)^\top \mathrm{RMSNorm}(\mathbf{k}_t)}{\sqrt{d}}\right), \tag{5}$$

where $\sigma(\cdot)$ is the sigmoid function and d is the representation dimension. The gated output $\alpha_t \mathbf{v}_t$ dynamically modulates the static n-gram memory. Although this approach successfully offloads the reconstruction of local patterns from the Transformer layers to external lookups, its reliance on independent hash tables per n-gram order fundamentally prevents the sharing of latent structures between nested n-gram contexts.

4. Methodology

We first review how explicit n-gram contexts are extracted and used in Engram-style memory modules. We then introduce our proposed TN-gram, which replaces order-specific hashed lookup tables with a shared tensorized n-gram CP representation space. Finally, we describe how the resulting memory modules are integrated into Transformer blocks through context-aware gating and projection absorption. Figure 1 provides an overview of this pipeline and compares the separately hashed memory design of Engram with our proposed shared CP-tensorized memory in TN-gram.

4.1. Suffix n-gram Extraction

At each position t, we extract suffix n-grams ending at the current token. Given $x_{1:T} = (x_1, \ldots, x_T)$ and maximum order N, the order-n context is $x_{t-n+1:t} = (x_{t-n+1}, \ldots, x_t)$ for $n \in \{2, \ldots, N\}$, with left padding when the context extends beyond the beginning of the sequence. For example, the 2-gram and 3-gram contexts are (x_{t-1}, x_t) and (x_{t-2}, x_{t-1}, x_t). Similar to (Cheng et al., 2026), we consider $n \geq 2$ because 1-gram information is already captured by the standard token embedding. Each such context indexes an n-gram latent vector, whose compact parameterization is described next.

4.2. Tensorizing n-gram Embeddings in TN-gram

The collection of latent vectors indexed by the suffix contexts above can be viewed as an n-gram embedding table. For a single n-gram order, the original n-gram space is $\mathbb{R}^{V^n \times d}$, representing a conventional table that stores one d-dimensional vector for each of the V^n possible contexts. Therefore, we can naturally tensorize it to an order-$(n+1)$ tensor $\in \mathbb{R}^{V \times V \times \cdots \times V \times d}$ where the first n modes index token positions and the final mode indexes the latent dimension. This tensor view provides a structure that flat hashed tables lack, i.e., suffix contexts of different orders are nested. For example, the 2-gram, (x_{t-1}, x_t), is the suffix of the 3-gram, (x_{t-2}, x_{t-1}, x_t), suggesting that their representations should share latent patterns rather than be learned independently. TN-gram exploits this structure by sharing the CP factors for the tensors of different orders $n \in \{2, \ldots, N\}$.

4.3. TN-gram Formulation

Extracting lower-order n-grams from the largest-order N-gram. Let N be the largest n-gram order that we wish to include. For any order $n \leq N$, TN-gram represents the corresponding embedding tensor using the shared CP factors. More specifically, the embedding tensor for N-gram is defined as follows

$$\begin{aligned}\mathcal{T}^{(N)} = \mathcal{I} \times_1 \mathbf{A}_1 \times_2 \cdots \times_{N-n} \mathbf{A}_{N-n} \times_{N-n+1} \\ \mathbf{A}_{N-n+1} \cdots \times_N \mathbf{A}_N \times_{N+1} \mathbf{F},\end{aligned} \tag{6}$$

where the order-$(N+1)$ tensor, $\mathcal{T}^{(N)} \in \mathbb{R}^{V \times V \times \cdots \times d}$, is the N-gram embedding tensor, $\mathcal{I} \in \mathbb{R}^{R \times \cdots \times R}$ is a super-diagonal identity tensor, and $\times_i$ denotes the mode-i product. The N shared CP factors, $\mathbf{A}_1, \ldots, \mathbf{A}_N \in \mathbb{R}^{V \times R}$, are so-called token-position factor matrices ordered from oldest to newest token, while $\mathbf{F} \in \mathbb{R}^{d \times R}$ is the shared model dimension factor matrix, and R is the tensor rank.

For a lower-order n-gram, since the lower-order n-gram context simply occupy the last n token modes of the N-gram, we use shared vectors, $\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_{N-2} \in \mathbb{R}^R$, to obtain its embedding tensor

$$\mathcal{T}^{(n)} = \mathcal{T}^{(N)} \times_1 \hat{\mathbf{w}}_1 \times_2 \cdots \times_{N-n} \hat{\mathbf{w}}_{N-n}. \tag{7}$$

Due to the multilinear property of tensor networks, notice that $\mathbf{A}_k$ can be absorbed into $\hat{\mathbf{w}}_k$ for $1 \leq k \leq N-n$. In practice, we simply need to learn order-absorption vectors,$\mathbf{w}_k$, as $\mathbf{w}_k = \mathbf{A}_k \hat{\mathbf{w}}_k$. Therefore, the embedding tensor for n-gram can be efficiently written with the shared token-position factor matrices, $\mathbf{A}_1, \ldots, \mathbf{A}_N \in \mathbb{R}^{V \times R}$, and shared order-absorption matrices, $\mathbf{w}_1, \ldots, \mathbf{w}_{N-2} \in \mathbb{R}^R$, as follows

$$\begin{aligned}\mathcal{T}^{(n)} = \mathcal{I} \times_1 \mathbf{w}_1 \times_2 \cdots \times_{N-n} \mathbf{w}_{N-n} \times_{N-n+1} \\ \mathbf{A}_{N-n+1} \cdots \times_N \mathbf{A}_N \times_{N+1} \mathbf{F},\end{aligned} \tag{8}$$

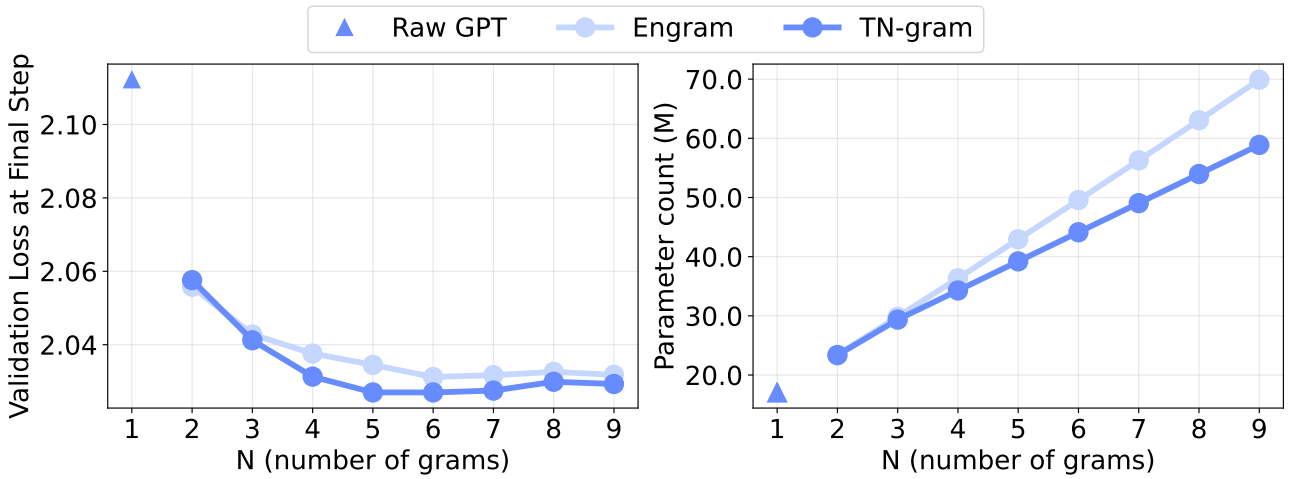

Figure 2. **Left**: Validation loss of the proposed TN-gram improves as N-gram order increases, with most of the gain achieved by 5-gram and little improvement thereafter. **Right**: Parameter count of both methods grows roughly linearly with N, with TN-gram using fewer parameters than the Engram baseline for the same N, with a CP tensor rank of 1200.

To obtain the embedding for a specific order-n context $(x_1, \ldots, x_n)$, since the CP core $\mathcal{I}$ is super-diagonal, the first N mode-n products in Eq. (8) reduce to an element-wise (Hadamard) product of the selected token factors and the order-absorption vectors, in the form

$$\begin{aligned}\mathbf{b}_n(x_{N-n+1:N}) = \mathbf{w}_1 \odot \cdots \odot \mathbf{w}_{N-n} \\ \odot \mathbf{A}_{N-n+1}(x_{N-n+1}, :) \odot \cdots \odot \mathbf{A}_N(x_N, :),\end{aligned} \tag{9}$$

where $\odot$ denotes the Hadamard product and $\mathbf{A}_i[x, :]$ selects the xth row of $\mathbf{A}_i$.

Under the standard CP-factorized parameterization, the embedding vector would have been obtained by multiplying the vector $\mathbf{b}_n(x_{N-n+1:N}) \in \mathbb{R}^R$ with $\mathbf{F} \in \mathbb{R}^{d \times R}$ as $\mathbf{h}^{(n)} = \mathbf{b}_n \mathbf{F}^\top$. For numerical stability in practice, we first normalize and rescale the token-space representation as

$$\mathbf{e}_n = s_n \mathrm{RMSNorm}\left(\mathbf{b}_n\right), \tag{10}$$

where $s_n = \exp(l_n)$ is a learnable scalar for n-gram order $n \in \{2, \ldots, N\}$. RMSNorm stabilizes these vectors, while the learned scalar s_n lets the model adaptively weigh n-gram contexts of different order. To obtain the embedding vector, we simply need to do $\mathbf{h}^{(n)} = \mathbf{e}_n \mathbf{F}^\top \in \mathbb{R}^d$. In Section 4.3, we detail how TN-gram avoids materializing this multiplication by absorbing $\mathbf{F}$ into the subsequent Key/Value projections, allowing the memory module to operate directly on $\mathbf{e}_n$. We ablate these design choices comprehensively in Section 5.1.

Efficient key and value projections by absorbing F. We use the Engram-style context-aware gating mechanism (Cheng et al., 2026) to filter n-gram memory using the current hidden state, $\mathbf{h}_t$. Given the token-space representations $\mathbf{e}_2, \ldots, \mathbf{e}_N$ from Eq. (10), a naive implementation would first project each order to model dimension using the CP model dimension factor $\mathbf{F}$, concatenate the resulting embeddings, and then apply Key/Value projections to obtain $\mathbf{d}_{\text{concat}} = \left[\mathbf{e}_2 \mathbf{F}^\top \mid \cdots \mid \mathbf{e}_N \mathbf{F}^\top\right]$. For the Key projection, let $\mathbf{W}_K = \begin{bmatrix} \mathbf{W}_{K,2} \\ \vdots \\ \mathbf{W}_{K,N} \end{bmatrix} \in \mathbb{R}^{(N-1)d_{\text{model}} \times d_{\text{model}}}$, we have

$$\begin{aligned}\mathbf{k}_t = \mathbf{d}_{\text{concat}} \mathbf{W}_K &= \sum_{n=2}^{N} \left(\mathbf{e}_n \mathbf{F}^\top\right) \mathbf{W}_{K,n} \\ &= \sum_{n=2}^{N} \mathbf{e}_n \left(\mathbf{F}^\top \mathbf{W}_{K,n}\right).\end{aligned} \tag{11}$$

Since both multiplications are linear, we can absorb $\mathbf{F}^\top \mathbf{W}_{K,n}$ into a single learned matrix $\mathbf{M}_{K,n} \in \mathbb{R}^{R \times d_{\text{model}}}$, and do the same for the Value projection. In practice, we concatenate the token-space vectors directly, $\mathbf{e}_{\text{concat}} = [\mathbf{e}_2 \mid \cdots \mid \mathbf{e}_N] \in \mathbb{R}^{1 \times (N-1)R}$, and compute

$$\mathbf{k}_t = \mathbf{e}_{\text{concat}} \mathbf{M}_K, \qquad \mathbf{v}_t = \mathbf{e}_{\text{concat}} \mathbf{M}_V, \tag{12}$$

where $\mathbf{M}_K, \mathbf{M}_V \in \mathbb{R}^{(N-1)R \times d_{\text{model}}}$. This avoids materializing intermediate matrix multiplications and removes the constraint that each order-specific Key/Value map must have the same model dimension factor $\mathbf{F}$. Finally, TN-gram follows the same Engram context-aware gating mechanism (Cheng et al., 2026) as detailed in Eq. (5), where the Value vector, $\mathbf{v}_t$, is filtered by a scalar gate computed from the current hidden state and Key vector, $\mathbf{k}_t$, passed through a depthwise causal convolution, and added to the residual stream. The full gating and convolution equations are given in Appendix D.

Table 1. Comparison of Engram and TN-gram with raw GPT at 9 and 18 layers.

Method	$L = 9, N = 5$ $(R = 1024)$				$L = 18, N = 5$ $(R = 1800)$			
	Δ Params $\downarrow$	Train Loss $\downarrow$	Val BPB $\downarrow$	CORE $\uparrow$	Δ Params $\downarrow$	Train Loss $\downarrow$	Val BPB $\downarrow$	CORE $\uparrow$
Raw GPT	–	2.209	1.251	0.06	–	1.792	1.086	0.107
Engram	26M	2.142	1.209	0.072	60M	1.771	**1.070**	0.115
TN-gram **(Ours)**	**19M**	**2.141**	**1.208**	**0.083**	**48M**	**1.770**	1.071	**0.120**

4.4. Sharing Latents across n-gram Embeddings

The formulation of TN-gram uniquely ensures that the embeddings of a specific n-gram are constructed from the shared factors across different n-gram orders. We formalize this into Theorem 4.1, which states that the token-space extraction vectors of a $n-1$-gram is a linear combination of the token-space extraction vectors of a n-gram.

Theorem 4.1. ***Shared latents across n-grams embeddings.*** *Let $\mathbf{b}_n(x_{N-n+1:N})$ denote the unnormalized token-space extraction vector of an n-gram as defined in Eq. (9), with tokens ordered from oldest to newest. For any $n \in \{3, \ldots, N\}$ where N-gram is the largest order, if $\mathbf{A}_{N-n+1}$ has full column rank, the token-space extraction vector of the shorter $(n-1)$-gram context, obtained by dropping the oldest token x_{N-n+1}, satisfies*

$$\mathbf{b}_{n-1}(x_{N-n+2:N}) = \sum_{x_{N-n+1} \in \mathcal{V}} c_{x_{N-n+1}} \mathbf{b}_n(x_{N-n+1:N}), \tag{13}$$

where $c_{x_{N-n+1}} \in \mathbb{R}$ depends only on $\mathbf{w}_{N-n+1}$ and $\mathbf{A}_{N-n+1}$.

5. Experiments

To evaluate the performance of our proposed TN-gram method, we conducted experiments on LLM pretraining at two scales, i.e., 9 transformer layers and 18 transformer layers. Our experimental setup builds on Parameter-Golf (OpenAI, 2026). All models were trained on data from FineWeb (Penedo et al., 2024). Our results show that TN-gram achieves performance on par or better than Engram across all evaluated n-gram orders, while requiring fewer trainable parameters.

Effect of n-gram Order. Figure 2 depicts how validation loss and parameter count vary with the maximum n-gram order N. Validation loss generally improves as N increases, with most gains achieved by $N = 5$ and only marginal improvements and sometimes degradations thereafter.

An exception occurs at $N = 2$, where TN-gram has slightly higher validation loss than Engram. This is expected, since the main advantage of TN-gram comes from sharing latent factors across nested n-gram orders, i.e., using lower-order suffixes to support higher-order contexts. When $N = 2$, there is only one explicit n-gram order, which means the cross-order sharing mechanism is absent. In this case, TN-gram behaves as a CP-factorized bigram memory, whereas Engram can assign more flexible independent embeddings through its hashed lookup table. Moreover, 2-gram contexts are much denser and better observed than higher-order n-grams, so Engram suffers less from the occurrence sparsity limitations that become more significant at larger N.

As N increases, TN-gram can exploit its shared factors and order-absorption vectors more effectively, which leads to better validation performance. Since most of the performance gains are achieved by $N = 5$, with only marginal improvements beyond that, we use $N = 5$ for the main comparisons in Tables 1 - 3.

Accuracy-Efficiency Tradeoff. Table 1 compares Engram and TN-gram against a raw GPT baseline at two transformer depths. We report four metrics: (1) the number of additional parameters relative to the raw GPT; (2) training loss; (3) validation bits-per-byte (BPB); and (4) the CORE score on downstream tasks. BPB is computed as $\mathcal{L}_{\text{BPB}} = \frac{\mathcal{L}_{\text{cross-entropy}}}{\ln(2)} \times \frac{\text{Total tokens}}{\text{Total bytes}}$, while CORE score aggregates downstream task accuracy (See more details in Appendix B).

For a fair comparison, we chose the tensor rank in TN-gram to be $R = 1024$ and $R = 1800$ for the 9- and 18-layer models, respectively, so that TN-gram has less number of additional parameters to Engram. Also, both Engram and TN-gram use a 5-gram configuration. Across both depths, TN-gram achieved the best CORE score while adding fewer parameters than Engram. For 9 layers, it also obtained the best validation BPB, reducing BPB from 1.251 to 1.208. For 18 layers, it matches Engram in validation BPB (1.071 vs. 1.070) while improving CORE from 0.107 to 0.120. These results suggest that TN-gram provides a stronger accuracy-efficiency tradeoff, since its shared CP-factorized memory couples well-observed lower-order n-grams with sparser higher-order ones, allowing latent structure from frequent patterns to inform the representations of rarer long-context patterns.

Downstream Evaluation. Tables 2 and 3 compare Raw GPT, Engram, and TN-gram on downstream tasks. Across

Table 2. Downstream performance comparison on multiple-choice tasks. Accuracy is reported. Best result per task is bolded.

L	Method	Avg	HS-ZS	ARC-E	ARC-C	COPA	CSQA	PIQA	OBQA	HS	LSAT-AR	BoolQ	LangID
9	Raw GPT	33.48	30.00	30.89	19.45	52.00	20.72	57.40	20.40	29.31	23.48	59.17	**25.51**
	Engram	34.14	30.14	**31.52**	**20.48**	**55.00**	21.13	57.51	23.80	**29.87**	22.61	58.17	25.36
	TN-gram	**35.44**	**30.23**	31.10	19.45	54.00	**30.71**	**57.62**	**24.40**	29.75	**25.22**	**61.99**	25.35
18	Raw GPT	36.85	34.57	34.05	**22.70**	58.00	26.78	62.02	22.80	33.99	24.35	60.55	25.50
	Engram	37.16	**35.56**	35.90	22.44	**60.00**	26.13	60.99	22.20	**34.60**	23.04	**62.14**	**25.72**
	TN-gram	**37.82**	35.05	**36.07**	22.44	**60.00**	**28.58**	**63.00**	**23.60**	34.41	**25.22**	62.08	25.53

Table 3. Downstream performance comparison on language-modeling and BIG-bench tasks. Accuracy is reported. Best result per task is bolded.

L	Method	Avg	BBH-QA	LAMB	BBH-Dyck	BBH-CS	BBH-Op	SQuAD	CoQA	WG	WGrande
9	Raw GPT	21.68	3.23	20.53	13.40	38.41	**13.33**	0.34	2.00	53.48	50.36
	Engram	22.43	4.46	22.94	**13.90**	**38.94**	11.90	0.45	5.45	53.48	50.36
	TN-gram	**22.72**	**6.29**	**23.31**	**13.90**	37.42	12.38	**0.79**	**5.93**	**53.85**	**50.59**
18	Raw GPT	25.35	9.05	28.64	**16.90**	39.77	15.71	0.96	7.95	57.51	51.62
	Engram	26.05	**13.99**	**32.04**	14.80	40.98	12.38	0.92	8.18	**60.07**	51.07
	TN-gram	**26.29**	12.32	31.83	15.20	**41.36**	**16.19**	**1.35**	**8.92**	56.41	**53.04**

both 9- and 18-layer settings, TN-gram consistently improves average downstream accuracy over the raw GPT baseline and Engram. On multiple-choice tasks, TN-gram achieves the best average accuracy, improving from 33.48 to 35.44 for 9 layers and from 36.85 to 37.82 for 18 layers. Similar gains were observed on language-modeling and BIG-bench tasks, where TN-gram obtains the highest average score at both depths. These results suggest that TN-gram improves both validation loss and downstream generalization.

Effect of Tensor Rank We next investigate whether additional explicit-memory capacity is more useful when allocated to independently hashed memories or to the shared tensorized latent space. For Engram, we scale capacity by increasing the number of hashing heads, which adds more order-specific lookup tables. For our TN-gram, we scale capacity by increasing the CP tensor rank R, which enlarges the number of shared latent components used to construct all n-gram orders.

Figure 3 shows a clear difference between two scaling directions. Increasing the number of Engram hashing heads yields modest improvements in validation loss, suggesting that duplicating independent hash memories has diminishing returns. This is consistent with Engram's design, in which additional heads increase the number of retrieved slots, but do not leverage the shared latent structure across nested n-gram contexts.

By contrast, increasing the CP rank in TN-gram produces a more consistent reduction in validation loss. A larger rank

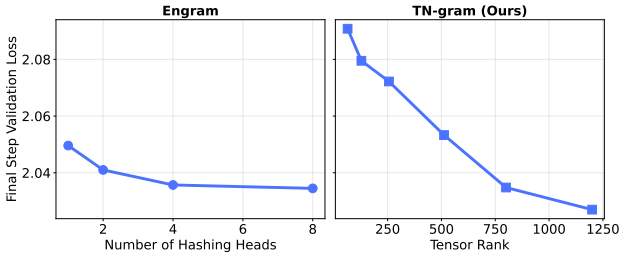

Figure 3. Scaling explicit n-gram memory capacity. **Left**: Increasing the number of hashing heads in Engram yields only limited validation-loss improvement. **Right**: increasing the CP tensor rank R in TN-gram consistently improves validation loss.

directly increases the expressivity of the shared tensorized n-gram space, while slightly increasing the parameter count. More importantly, higher rank improves the shared latent components from which different n-gram orders are composed, rather than simply adding more isolated memories.

5.1. Ablation Study

Effect of vocabulary size. Experiments in Tables 1, 2, 3, and Figure 2 use vocabulary size 1024. To investigate whether the advantages of TN-gram scale to a larger vocabulary, Figure 4 (left) compares Engram and TN-gram at vocabulary size 8192, reporting loss reduction relative to the raw GPT baseline. Both methods improve training and validation loss, confirming that explicit N-gram modeling remains beneficial even at higher vocabulary sizes. TN-gram yields larger reductions than Engram, improv-

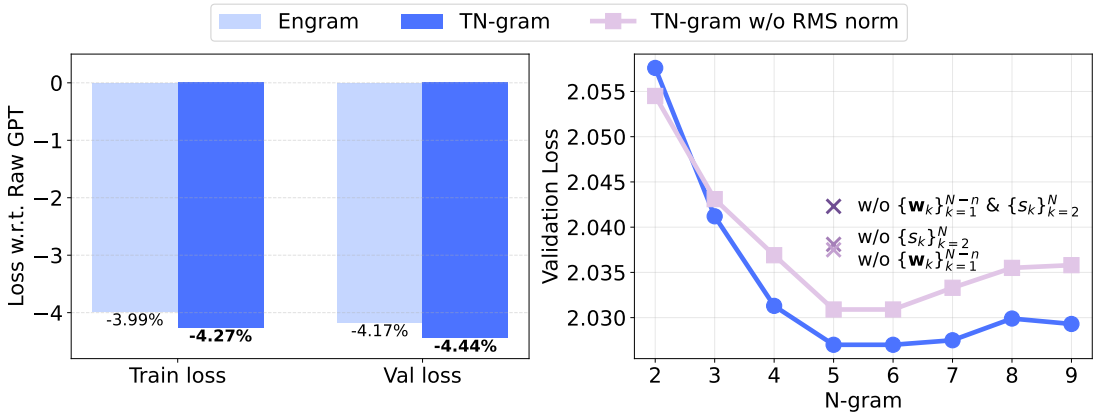

Figure 4. Ablation Study. **Left**: Loss reduction relative to raw GPT for Engram and TN-gram for the vocabulary size 8192 with all 5-grams. **Right**: Validation loss of N-gram order for TN-gram ablations, showing effects of removing RMS normalization, learnable scale $\{s_k\}_{k=2}^{N}$, and order-absorption vectors $\{\mathbf{w}_k\}_{k=1}^{N-n}$.

ing training loss by 4.27% and validation loss by 4.44%, compared to 3.99% and 4.17% for Engram. This suggests that TN-gram provides a more effective parameterization of N-gram statistics even as the vocabulary size increases.

Ablation of TN-gram components. Section 4.3 introduces three components for stabilizing the learnt TN-gram representations: RMS normalization, learnable per-order scaling $\{s_k\}_{k=2}^{N}$, and learnable order-absorption vectors $\{\mathbf{w}_k\}_{k=1}^{N-n}$ for unspecified context modes. We ablate these components to assess their individual contributions. As shown in Figure 4 (right), each removal increases validation loss, confirming that all three components are beneficial. In particular, removing the learnable scales has a larger effect than removing the order-absorption vectors alone, while removing both leads to the worst performance. The variant without RMS normalization is consistently worse than the full TN-gram across n-gram orders, highlighting the importance of normalization for stabilizing CP-based n-gram representations.

6. Conclusion

We have introduced the **T**ensorized E**ngram** (**TN-gram**), a compact explicit n-gram memory module for incorporating local multi-token context into Transformer blocks. Unlike Engram-style approaches that use separate hash tables for each n-gram order, TN-gram represents the tensorized n-gram spaces with shared CP factors. This design removes hash collision and, more importantly, enables lower- and higher-order n-gram contexts to share latent structure through common token-position factors and learnable order-absorption vectors. We anticipate that this work will open future avenues for explorations in tensorized n-gram memories.

Limitations and future work. TN-gram improves parameter efficiency, but its current implementation is not yet fully optimized for wall-clock speed. Compared with hash-based Engram, TN-gram performs additional structured operations such as factor gathering, Hadamard products, normalization, and scaling. These operations are simple and highly parallelizable, and we expect the remaining overhead to be substantially reduced with customized CUDA kernels. Another practical consideration is the choice of tensor rank R. In our experiments, R is selected to match Engram's parameter budget, but the best rank may vary with model scale, vocabulary size, maximum n-gram order, and compute constraints. Since tensor rank selection is a challenging and active research area, developing adaptive rank-selection strategies for TN-gram is an open direction for future work.

Impact Statement

The proposed TN-gram may have positive societal impact by improving parameter efficiency for explicit n-gram memories, potentially contributing to more efficient and sustainable language models. By compactly modeling recurring multi-token patterns, it may reduce the parameter overhead in the neural network layers.

References

Basharin, A., Chertkov, A., and Oseledets, I. Faster language models with better multi-token prediction using tensor decomposition. *arXiv preprint arXiv:2410.17765*, 2024.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., et al. PIQA: Reasoning about Physical Commonsense in Natural Language. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 34, pp. 7432–7439, 2020.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

Brants, T., Popat, A., Xu, P., Och, F. J., and Dean, J. Large Language Models in Machine Translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 858–867, 2007.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language Models are Few-shot Learners. *Advances in Neural Information Processing Systems*, 33: 1877–1901, 2020.

Chen, S. F. and Goodman, J. An Empirical Study of Smoothing Techniques for Language Modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

Cheng, X., Zeng, W., Dai, D., Chen, Q., Wang, B., Xie, Z., Huang, K., Yu, X., Hao, Z., Li, Y., et al. Conditional Memory via Scalable Lookup: A New Axis of Sparsity for Large Language Models. *arXiv preprint arXiv:2601.07372*, 2026.

Cichocki, A., Mandic, D., De Lathauwer, L., Zhou, G., Zhao, Q., Caiafa, C., and PHAN, H. A. Tensor Decompositions for Signal Processing Applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163, 2015. doi: 10.1109/MSP.2013.2297439.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. P. Tensor Networks for Dimensionality Reduction and Large-scale Optimization Part 1 Low-rank Tensor Decompositions. *Foundations and Trends in Machine Learning*, 9(4-5):249–429, 2016.

Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300/.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 conference of the North American chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Goodman, J. T. A Bit of Progress in Language Modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

Gordon, A., Kozareva, Z., and Roemmele, M. SemEval-2012 Task 7: Choice of Plausible Alternatives: An Evaluation of Commonsense Causal Reasoning. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pp. 394–398, Montréal, Canada, 7-8 June 2012. Association for Computational Linguistics. URL https://aclanthology.org/S12-1052/.

Gu, Y., Zhou, W., Iacovides, G., and Mandic, D. TensorLLM: Tensorising Multi-Head Attention for Enhanced Reasoning and Compression in LLMs. *IEEE International Joint Conference on Neural Networks (IJCNN 2025)*, 2025.

Gu, Y., Zhou, W., Iacovides, G., and Mandic, D. TeRA: Vector-based random tensor network for high-rank adaptation of large language models. In *The 64th Annual Meeting of the Association for Computational Linguistics*, 2026.

Huang, H., Zhu, D., Wu, B., Zeng, Y., Wang, Y., Min, Q., and Xun, Z. Over-tokenized transformer: Vocabulary is generally worth scaling. In Singh, A., Fazel, M., Hsu, D., Lacoste-Julien, S., Berkenkamp, F., Maharaj, T., Wagstaff, K., and Zhu, J. (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 26261–26282. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/huang25bb.html.

Iacovides, G., Zhou, W., Li, C., Zhao, Q., and Mandic, D. Domain-aware Tensor Network Structure Search. *arXiv preprint arXiv:2505.23537*, 2025.

Jordan, K., Jin, Y., Boza, V., You, J., Cesista, F., Newhouse, L., and Bernstein, J. Muon: An Optimizer for Hidden Layers in Neural Networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kolda, T. G. and Bader, B. W. Tensor Decompositions and Applications. *SIAM Review*, 51(3):455–500, 2009. doi: 10.1137/07070111X. URL https://doi.org/10.1137/07070111X.

Levesque, H. J., Davis, E., and Morgenstern, L. The Winograd Schema Challenge. *KR*, 2012(13th):3, 2012.

Li, J., Fang, A., Smyrnis, G., Ivgi, M., Jordan, M., Gadre, S., Bansal, H., Guha, E., Keh, S., Arora, K., et al. DataComp-LM: In Search of the Next Generation of Training Sets for Language Models. *Advances in Neural Information Processing Systems*, 37:14200–14282, 2024.

Liu, H., Zhang, J., Wang, C., Hu, X., Lyu, L., Sun, J., Yang, X., Wang, B., Li, F., Qian, Y., et al. Scaling Embeddings Outperforms Scaling Experts in Language Models. *arXiv preprint arXiv:2601.21204*, 2026.

Manning, C. and Schutze, H. *Foundations of Statistical Natural Language Processing*. MIT press, 1999.

Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2381–2391, 2018.

Nguyen, T. Understanding Transformers via n-gram Statistics. *Advances in Neural Information Processing Systems*, 37:98049–98082, 2024.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing Neural Networks. *Advances in Neural Information Processing Systems*, 28, 2015.

OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023.

OpenAI. OpenAI Model Craft Challenge: Parameter Golf, 2026. URL https://github.com/openai/parameter-golf.

Pagnoni, A., Pasunuru, R., Rodriguez, P., Nguyen, J., Muller, B., Li, M., Zhou, C., Yu, L., Weston, J. E., Zettlemoyer, L., et al. Byte Latent Transformer: Patches Scale Better Than Tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9238–9258, 2025.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Penedo, G., Kydlíček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., and Wolf, T. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264/.

Reddy, S., Chen, D., and Manning, C. D. CoQA: A Conversational Question Answering Challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266, 2019. doi: 10.1162/tacl_a_00266. URL https://aclanthology.org/Q19-1016/.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *Communications of the ACM*, 64(9):99–106, 2021.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., et al. Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models. *Transactions on Machine Learning Research*, 2023.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421/.

Team, L. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

Tito Svenstrup, D., Hansen, J., and Winther, O. Hash Embeddings for Efficient Word Representations. *Advances in Neural Information Processing Systems*, 30, 2017.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is All You Need. *Advances in Neural Information Processing Systems*, 30, 2017.

Xu, M., Xu, Y. L., and Mandic, D. P. TensorGPT: Efficient Compression of Large Language Models Based on Tensor-Train Decomposition. *arXiv preprint arXiv:2307.00526*, 2023.

Yang, Y., Zhou, J., Wong, N., and Zhang, Z. LoRETTA: Low-Rank Economic Tensor-Train Adaptation for Ultra-Low-Parameter Fine-Tuning of Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3161–3176, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.174. URL https://aclanthology.org/2024.naacl-long.174/.

Yu, D., Cohen, E., Ghazi, B., Huang, Y., Kamath, P., Kumar, R., Liu, D., and Zhang, C. Scaling Embedding Layers in Language Models. *arXiv preprint arXiv:2502.01637*, 2025.

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. HellaSwag: Can a Machine Really Finish Your Sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, 2019.

Zhong, W., Cui, R., Guo, Y., Liang, Y., Lu, S., Wang, Y., Saied, A., Chen, W., and Duan, N. AGIEval: A Human-Centric Benchmark for Evaluating Foundation Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2299–2314, 2024.

Zhou, W., Iacovides, G., Konstantinidis, K., Kisil, I., and Mandic, D. Understanding the Rank of Tensor Networks via an Intuitive Example-Driven Approach. *arXiv preprint arXiv:2507.10170*, 2025.

Zhou, W., Gu, Y., Iacovides, G., and Mandic, D. KromHC: Manifold-constrained hyper-connections with kronecker-product residual matrices. In *Forty-third International Conference on Machine Learning*, 2026.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A. M., Le, Q. V., Laudon, J., et al. Mixture-of-Experts with Expert Choice Routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

A. Proof of Theorem 1

Theorem A.1. ***Shared Latents Across n-grams.*** *Let $\mathbf{b}_n(x_{N-n+1:N})$ denote the unnormalized token-space extraction vector of an n-gram as defined in Eq. (9), with tokens ordered from oldest to newest. For any $n \in \{3, \dots, N\}$ where N-gram is the largest order, if $\mathbf{A}_{N-n+1}$ has full column rank, the token-space extraction vector of the shorter $(n-1)$-gram context, obtained by dropping the oldest token x_{N-n+1}, satisfies*

$$\mathbf{b}_{n-1}(x_{N-n+2:N}) = \sum_{x_{N-n+1} \in \mathcal{V}} c_{x_{N-n+1}} \mathbf{b}_n(x_{N-n+1:N}), \tag{14}$$

where $c_{x_{N-n+1}} \in \mathbb{R}$ depends only on $\mathbf{w}_{N-n+1}$ and $\mathbf{A}_{N-n+1}$.

Proof. Let $n \in \{3, \dots, N\}$ and let $\mathbf{A} = \mathbf{A}_{N-n+1} \in \mathbb{R}^{V \times R}$ and $\mathbf{w} = \mathbf{w}_{N-n+1} \in \mathbb{R}^R$. We define the shared suffix vector as follows

$$\mathbf{q}(x_{N-n+2:N}) := \mathbf{w}_1 \odot \cdots \odot \mathbf{w}_{N-n} \odot \mathbf{A}_{N-n+2}(x_{N-n+2}, :) \odot \cdots \odot \mathbf{A}_N(x_N, :) \ \in \mathbb{R}^R, \tag{15}$$

which collects every factor shared by orders n and $n-1$. Based on Eq. (9), we can write the two extraction vectors as

$$\mathbf{b}_n(x_{N-n+1:N}) = \mathbf{A}(x_{N-n+1}, :) \odot \mathbf{q}(x_{N-n+2:N}), \ \mathbf{b}_{n-1}(x_{N-n+2:N}) = \mathbf{w} \odot \mathbf{q}(x_{N-n+2:N}). \tag{16}$$

Since $\mathbf{A}$ has full column rank, its rows span $\mathbb{R}^R$. Therefore, there exist coefficients $\{c_{x_{N-n+1}}\}_{x_{N-n+1} \in \mathcal{V}}$ satisfying $\mathbf{w} = \sum_{x_{N-n+1} \in \mathcal{V}} c_{x_{N-n+1}} \mathbf{A}(x_{N-n+1}, :)$. Writing out the Hadamard product gives

$$\begin{aligned} \mathbf{b}_{n-1}(x_{N-n+2:N}) = \mathbf{w} \odot \mathbf{q} &= \left(\sum_{x_{N-n+1} \in \mathcal{V}} c_{x_{N-n+1}} \mathbf{A}(x_{N-n+1}, :) \right) \odot \mathbf{q} \\ &= \sum_{x_{N-n+1} \in \mathcal{V}} c_{x_{N-n+1}} \underbrace{\mathbf{A}(x_{N-n+1}, :) \odot \mathbf{q}}_{\mathbf{b}_n(x_{N-n+1:N})}, \end{aligned} \tag{17}$$

where $c_{x_{N-n+1}}$ depends only on $\mathbf{w} = \mathbf{w}_{N-n+1}$ and $\mathbf{A} = \mathbf{A}_{N-n+1}$. □

B. Downstream Tasks

We evaluate downstream task performance using a broad suite of tasks covering commonsense reasoning, scientific question answering, reading comprehension, language modeling, symbolic reasoning, and factual knowledge. The multiple-choice group includes HellaSwag (Zellers et al., 2019) zero-shot and 10-shot variants for grounded commonsense completion, ARC-Easy and ARC-Challenge (Clark et al., 2018) for elementary and challenging scientific reasoning, COPA (Gordon et al., 2012) and CommonsenseQA (Talmor et al., 2019) for causal and commonsense inference, PIQA (Bisk et al., 2020) for physical commonsense, OpenBookQA (Mihaylov et al., 2018) for open-domain science facts, AGI-Eval LSAT-AR (Zhong et al., 2024) for analytical reasoning, BoolQ (Clark et al., 2019) for yes/no question answering, and BigBench (Srivastava et al., 2023) language identification.

The language-modeling and generation-style group includes LAMBADA (Paperno et al., 2016) for long-range next-word prediction, SQuAD (Rajpurkar et al., 2016) and CoQA (Reddy et al., 2019) for extractive and conversational question answering, Winograd (Levesque et al., 2012) and Winogrande (Sakaguchi et al., 2021) for pronoun and commonsense coreference resolution, and BigBench (Srivastava et al., 2023) tasks such as Wikidata QA, Dyck languages, CS algorithms, operators, and repeat-copy logic, which test factual recall, formal-language structure, algorithmic reasoning, symbolic operations, and sequence-copying ability.

Following DataComp-LM (Li et al., 2024), we report CORE as an aggregate downstream metric. CORE averages the centered accuracy across tasks, where each task score is normalized so that random guessing corresponds to 0 and perfect accuracy corresponds to 1. This makes results more comparable across tasks with different chance levels.

Together, these benchmarks test whether TN-gram's validation-loss improvements translate into broader downstream generalization. We report both individual task accuracies and CORE, using CORE as a compact summary of overall downstream performance.

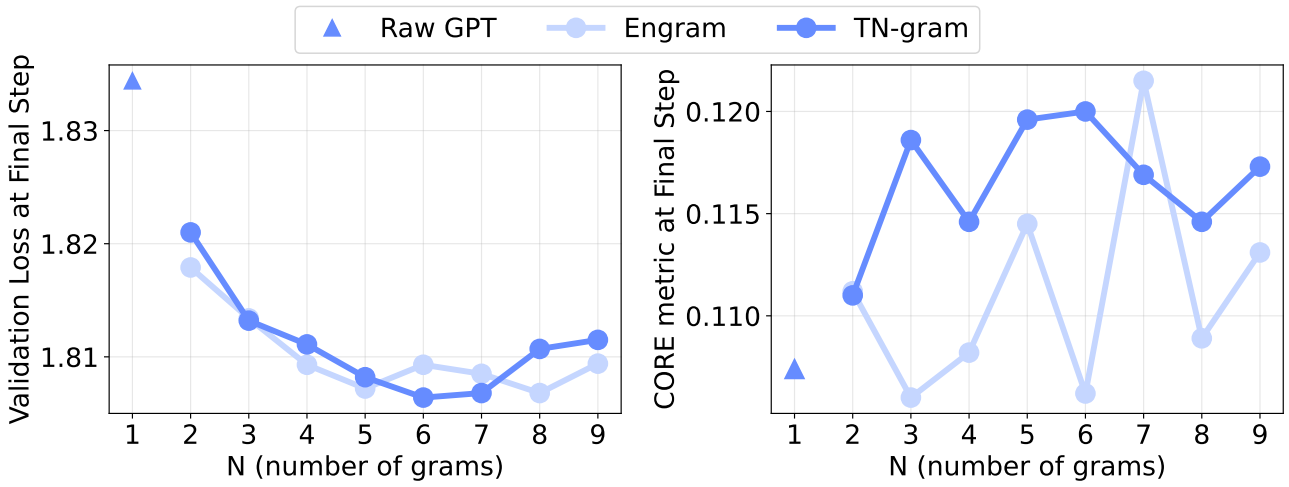

Figure 5. Final-step performance of 18-layer transformer models across n-gram orders N. **Left**: TN-gram yields the lowest validation loss than Engram and raw GPT baseline. The tensor rank in TN-gram is configured such that it always has less parameters than Engram. **Right**: TN-gram achieves higher CORE score, with strongest gain at larger N.

C. Scaling n-gram Order in a 19-Layer Transformer

To evaluate the scaling performance of our proposed TN-gram, we conducted experiments using 18-layer transformers with embedding dimension 1024, and trained it for 10,000 steps. The validation loss and the CORE score at the final step are shown in Figure 5.

D. Context-Aware Gating

This section details the Engram-style (Cheng et al., 2026) context-aware gating mechanism used to inject TN-gram memory into the Transformer. At position t, let $\mathbf{h}_t \in \mathbb{R}^{d_{\text{model}}}$ denote the current hidden state, and let $\mathbf{k}_t, \mathbf{v}_t \in \mathbb{R}^{d_{\text{model}}}$ denote the memory Key and Value vectors produced from the extracted n-gram representations. The gate compares the current context with the memory Key and suppresses memory values that are poorly aligned with the hidden state.

We first compute a normalized alignment score and transform it into a scalar gate:

$$\alpha_t = \frac{\text{RMSNorm}(\mathbf{h}_t)^\top \text{RMSNorm}(\mathbf{k}_t)}{\sqrt{d_{\text{model}}}}, \qquad g_t = \sigma\left(\text{sign}(\alpha_t)\sqrt{\max(|\alpha_t|, \epsilon)}\right), \tag{18}$$

where $\sigma(\cdot)$ is the sigmoid function and $\epsilon = 10^{-6}$. The signed square-root transformation reduces the magnitude of extreme dot products while preserving their sign. The gated memory is $\tilde{\mathbf{v}}_t = g_t \mathbf{v}_t$.

Finally, following Engram (Cheng et al., 2026), we apply a short depthwise causal convolution to the sequence of gated values before adding the memory output to the residual stream. Let $\tilde{\mathbf{v}}$ denote the sequence of gated values over time. With kernel size $w = 3$, dilation $\delta = N$, and SiLU activation, the memory output added to the Transformer hidden state is

$$\mathbf{y}_t = \tilde{\mathbf{v}}_t + \text{SiLU}(\text{Conv1D}\left(\text{RMSNorm}(\tilde{\mathbf{v}})\right))_t. \tag{19}$$

E. Hyperparameters

Table 4 reports the shared training and model hyperparameters. Unless otherwise noted, all models are trained for 6000 steps on length-1024 sequences with a global batch size of 524,288 tokens. Depending on the size of the model, training was performed using either 4 $\times$ NVIDIA A100 (80GB) GPUs or a single L40S GPU (48GB). The backbone is a 9-layer Transformer with hidden size 512, 8 query heads, 4 KV heads, RoPE base 10,000, tied input/output embeddings, and a two-layer MLP with expansion factor 2. Engram and TN-gram insert Engram memory modules in layers 1 and 7.

We use Muon (Jordan et al., 2024) for matrix-shaped Transformer parameters and Adam (Kingma & Ba, 2014) for embeddings, scalar/control parameters, and the LM head. Muon uses learning rate 0.04, Nesterov momentum ramped from

Table 4. Shared hyperparameters.

Name	Value
Sequence length	1024
Global batch size	524,288 tokens
Training steps	6000
Transformer blocks	9
Hidden dimension	512
Attention heads / KV heads	8 / 4
MLP expansion factor	2
RoPE base	10,000
Tied embeddings	Yes
Engram layers	1, 7
Random seed	1337

0.85 to 0.95 over the first 500 steps, and 5 Newton–Schulz steps for orthogonalization. Adam uses $\beta_1 = 0.9$, $\beta_2 = 0.95$, and $\epsilon = 10^{-8}$. For tied embeddings, the shared embedding table uses learning rate 0.05. Scalar/control parameters use learning rate 0.04. Learning rates are constant until the final 1200 steps, then linearly decayed to zero.

Table 5. Optimizer hyperparameters.

Name	Value
Matrix optimizer	Muon
Matrix LR	0.04
Muon momentum	0.85 $\rightarrow$ 0.95 over 500 steps
Adam $\beta_1, \beta_2, \epsilon$	0.9, 0.95, 10^{-8}
Tied embedding LR	0.05
Scalar/control LR	0.04
Warmdown steps	1200