Language Modeling Using Entanglement Enhanced Tensor Trains

Ellis Reyes[1] **Yi-Shin Chen**[1,2]
[1]Institute of Information Systems and Applications,
[2]Department of Computer Science,
National Tsing Hua University, Hsinchu, Taiwan
ellisreyesm@gmail.com yishin@gmail.com

Abstract

Tensor Train Language Models (TTLMs) offer significant memory savings by representing text sequences as tensor networks, but naive implementations struggle with long-range dependencies and limited flexibility. We introduce a modular TTLM framework that combine local and non-local context modules to achieve scalable language modeling. Our non-local modules, inspired by entanglement in quantum information theory, enable efficient modeling of long-range interactions between hidden states. Experiments on Penn Treebank and Wikitext datasets show that our modular TTLM, including entanglement-augmented variants, outperform naive baselines. These results highlight TTLMs as a promising, memory-efficient alternatives for modern language modeling.

Keywords: Tensor Train Language Models, Entanglement-Inspired Modules

1 Introduction

Language modeling is a fundamental problem in natural language processing (Bengio et al., 2003), requiring models to capture local and global correlations across sequences of tokens. Recurrent neural networks (RNNs) (Bengio et al., 1994) and their variants (e.g., LSTMs) (Hochreiter and Schmidhuber, 1997) have been previously used, but they often fail to model long-range dependencies due to vanishing gradients. Transformers (Vaswani et al., 2017), leveraging the attention mechanism, have established new performance benchmarks and are widely used. However, their quadratic time and memory complexity in sequence length remains a bottleneck (Tay et al., 2022) (Zaheer et al., 2020).

Previous approaches of Transformer variants have been explored to tackle this issue. Linformer (Wang et al., 2020), which computes a projection of the key and value matrices to lower ranks and Longformer (Beltagy et al., 2020), which uses sparse attention patterns instead of a full dense self-attention.

Tensor Train Language Models (Su et al., 2024) have recently emerged as a theoretically memory-efficient alternative, decomposing the input sequence into a chain of low-rank tensors. These tensor networks were originally introduced in quantum many-body physics (Eisert, 2013) as efficient representations of interactions in a highly dimensional space. In contrast with Transformer variants as *Linformer* and *Longformer*, Tensor Train Language Models provide an orthogonal approach: Instead of attention sparsification, they decompose the sequential weights into a chain of low-rank tensor cores, generating a *combinatorial space* where each token is represented by a core $G^{(k)}$, constructing a global representation $G^{(1)}[i_1] \cdots G^{(d)}[i_d]$ (Equation 1). The main idea is that tensor decompositions can capture hidden correlation patterns while keeping the model scalable and interpretable, a property that has been exploited in tasks ranging from Bayesian network discovery to hierarchical clustering of real-world data (Akamatsu et al., 2025).

This property provides an analogy with language, where long-range dependencies must be captured within memory constraints. Hence, our framework adopts an interdisciplinary view. We use entanglement-inspired modules to allow TTLMs to capture dependencies, following the observation that entanglement arises from local-interactions (Eisert, 2013). Nevertheless, current naive TTLM implemen-

Proceedings of the 37th Conference on Computational Linguistics and Speech Processing (ROCLING 2025), pages 258–265

tations face two main limitations: (1) difficulty in modeling global, non-local dependencies, and (2) inflexibility in their core architecture, which can constrain the expressiveness of the model. To our knowledge, this is the first work that explores TTLMs beyond theory, systematically enhancing their architecture and integrating entanglement-inspired non-local modules for practical large-scale language modeling

In short, we make the following contribution:

We propose a hybrid language model architecture that combines a TTLM that captures local context via sequence processing, with causal Entanglement modules to capture non-local context (such as attention and outer product mechanisms) that allow hidden states to interact beyond local context. As a foundation for our hybrid model, we introduce standard architectural improvements such as residual connections, biases, controlled initialization and weight tying into the baseline TTLM, allowing for a stronger base model where the entanglement modules operate. Ablation studies confirm the utility of these foundational enhancements.

Experiments on the Penn Treebank and WikiText-2 datasets show that our modular TTLM, and the entanglement-augmented variants, outperform naive TTLMs. For instance, on PTB at rank 60, our best variant achieves a perplexity of 83.70 compared to 92.79 for the naive Large TTLM, a $\sim$ 9.8% reduction. A similar trend is observed on the WikiText-2, where our hybrid approach consistently outperform across most tested ranks, validating the scalability and robustness of our approach. These results suggest that TTLMs are a promising direction for scalable and memory-efficient alternatives to attention-based architectures.

2 Related Work

Beyond computational motivations, tensor networks originate from quantum many-body physics, where entanglement and entropy laws explain why high-dimensional systems can still be represented efficiently (Eisert, 2013). Recent work has begun to explore these ideas in machine learning to capture hidden correlation structures, showing tensor networks as an alternative paradigm for interpretable generative modeling (Akamatsu et al., 2025). These insights suggest that entanglement-inspired mechanisms may become a viable approach to align tensor-based architectures to capture long-range dependencies. Earlier work (Zhang et al., 2020) has explored tensor decomposition techniques as an alternative approach for language modeling to mitigate the quadratic complexity $\mathcal{O}(n^2)$ of self-attention. These methods factorize the exponential space of weight matrices into smaller tensor representations, enabling a more efficient memory scaling and inference on resource constrained hardware.

Tensor Decomposition for Language Modeling Early work (Su et al., 2024) (Zhang et al., 2020) demonstrated that tensor based representations, such as matrix product states (MPS) and tensor train (TT) approaches, can model sequences with significantly fewer parameters than Recurrent Neural Networks and Transformers. Moreover, these representations can efficiently compress LLMs components (Tomut et al., 2024) (Xu et al., 2023). The *Tensorized Transformer* (Ma et al., 2019) explore this representations using Block Term Tensor Decomposition (BTD) to achieve comparable perplexity to Transformers-XL with 2–5$\times$ fewer parameters. However, only the attention layer is compressed, the feed-forward, embedding and softmax matrices remain full size which makes the computation quadratic in complexity.

The *Tensor Train Language Model* (Su et al., 2024) introduced a fully Tensor Train architecture for language modeling as a proof-of-concept work, outperforming vanilla Recurrent Neural Networks and matching Transformer perplexity performance on limited context windows and small datasets. However, existing tensor-based approaches mainly focus on compression without providing modular architectures capable of capturing non-local dependencies. In contrast, our modular TTLM framework augmented with entanglement-inspired modules seeks to bridge the theoretical efficiency of tensor networks in quantum information with the practical requirements for language modeling. To the best of our

knowledge, this entangled dynamics of local and non-local context for TLLMs has not been explored.

3 Preliminaries: Tensor Train Decomposition

The *Tensor Train* (TT) decomposition (Oseledets, 2011), factorizes a high-order tensor into a contracted sequence (or train) of low-order *cores*. Let $W \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be an order-d tensor. In TT format, each element $W(i_1, \ldots, i_d)$ is expressed as a chain of matrix products:

$$W(i_1, \ldots, i_d) = G^{(1)}[i_1]\, G^{(2)}[i_2]\, \cdots\, G^{(d)}[i_d]. \quad (1)$$

where $G^{(k)} \in \mathbb{R}^{r_{k-1} \times n_k \times r_k}$ is the k-th TT-core, $G^{(k)}[i_k] \in \mathbb{R}^{r_{k-1} \times r_k}$ corresponds the i_k-th slice along the k-th mode, and $(r_0, r_1, \ldots, r_d)$ are the TT-ranks with $r_0 = r_d = 1$. The total parameter count scales as $\sum_{k=1}^{d} r_{k-1} n_k r_k$, which grows linearly with d, in contrast to the exponential $\prod_k n_k$ parameters of a dense tensor.

3.1 Tensor Train Language Models

Tensor Train Language Models use the principles of TT decomposition introduced in Section 3 to generate a parameter-efficient approach for sequence modeling. Following a standard RNN update rule $h_t = f(x_t, h_{t-1})$, where $h_t \in \mathbb{R}^R$ is the hidden state of dimension R (TT-rank or rank), x_t is the input token at time t, and f is the cell function implemented using the tensor contractions.

The input token x_t is first mapped to an embedding vector E'_t. For example, in small variants like *TTLM-Tiny* (Su et al., 2024) The hidden state h_{t-1} is transformed by an input-to-hidden learnable weight matrix W_{ih}, and the result computes a matrix-vector product with E'_t, followed by an activation function:

$$h_t = (h_{t-1} W_{ih} + b_{ih}) \cdot E'_t \quad (2)$$

Larger variants like *TTLM-Large* (Su et al., 2024) first process the input embedding E_t by an additional hidden-to-hidden weight matrix W_{hh} before reshaping it into the embedding E'_t:

$$h_t = (h_{t-1} W_{ih} + b_{ih}) \cdot \text{reshape}(E_t W_{hh}) \quad (3)$$

This allows for a more detailed input dependent state transition.

4 Methodology

We propose a Tensor Train Language Model that converges (i) a recurrent TTLM for local context and (ii) non-local *entanglement* modules that enable hidden states interaction to capture long-range dependencies.

Figure 1 illustrates the overall architecture: The TTLM processes the input sequence $(x_1, \ldots, x_N)$ recurrently (left), where the state at each time h_t depends on the previous state h_{t-1} and the current input (x_t). The resulting sequence of hidden states, $h_1, \ldots, h_N$, where N represents the total length of the input sequence, is then aggregated by causal entanglement modules (right) to capture non-local dependencies before making a final prediction.

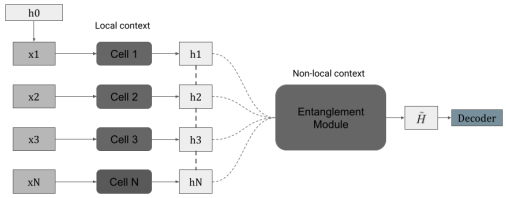

Figure 1: Modular TTLM with Entanglement.

Our key assumption here is that by entangle the local and non-local modules the expressiveness of the model increases. Our intuition is by allowing the hidden states to interact globally with non-local past context, we expect to capture long-range dependencies by the connection of the current state and its history. Moreover, the modularity could enable ablation studies to compare different behaviors among the entanglement variants to analyze outcomes for specific types of data and tasks.

4.1 Modular TTLM (Local Context)

Our hybrid architecture introduces an enhanced Tensor Train Language Model based on the TTLM-Large implementation from Section 3.1 and equation 3. The hidden state $h_t \in \mathbb{R}^R$ is updated from h_{t-1} and the current embedding $E_t \in \mathbb{R}^{R^2}$ as follows:

$$
\begin{aligned}
v_t &= (h_{t-1} W_{ih} + b_{ih}) \\
E'_t &= \text{Reshape}(E_t W_{hh} + b_{hh}) \\
h_t &= \text{Activation}\big(v_t \cdot E'_t + \alpha \cdot h_{t-1} + b_{\text{cell}}\big)
\end{aligned} \tag{4}
$$

In Equation 4, the previous state is transformed by a linear computation (W_{ih}, b_{ih}) where W_{ih} is a learnable weight matrix that connects input and hidden states, this matrix determines which past memory slices are relevant for the current position. The transition matrix E'_t is computed by reshaping the current embedding E_t by another linear computation (W_{hh}, b_{hh}). Here, W_{hh} is a learnable weight matrix that determines how the current input should influence the update. The reshaping flatten the transition matrix E'_t into $R \times R$. Conceptually, E'_t represents a flattened TT-core slice $G^{(k)}[i_k]$ from the decomposition chain seen in Eq. 1.

With this approach, we separate the relevant past memory slices into the transformed state v_t and the current state influence into the transition matrix E'_t. Hence the core interaction involves the matrix-vector product $v_t \cdot E'_t$. The flow of the gradient is improved with a residual connection to preserve information by adding the previous state back scaled by a hyperparameter α. In this implementation the biases (b_{ih}, b_{hh}) modify the inputs in the core tensor contraction computation and the bias b_{cell} is used as an extra learnable parameter to shift the overall output independently of the internal linear transformations before it passes through an activation function such as tanh. This activation introduces non-linearity, though omitting it (using a linear activation) aligns with baseline configurations (Su et al., 2024).

We employ weight tying for efficiency to share the weights between E_t and the output projection layer. All learnable parameters within the cell (the entire computational unit) $[W_{ih}, b_{ih}, W_{hh}, b_{hh}, b_{\text{cell}}]$ are initialized using a scaled uniform distribution $[-0.1/R, 0.1/R]$ for smooth initialization and stability, especially at higher ranks.

The Modular TTLM processes the input sequence $X = (x_1, \ldots, x_N)$ sequentially. The resulting sequence of hidden states

$$
H = [h_1, h_2, \ldots, h_N] \in \mathbb{R}^{N \times R}
$$

captures the local context and serves as the input to the causal Entanglement Blocks described in the next section.

4.2 Entanglement Modules (Non-local Context)

While the Modular TTLM introduced in Section 4.1 processes local dependencies, because of the recurrent nature potential information from the distant positions could be lost. To tackle this limitation and capture non-local dependencies, we introduce the Entanglement modules. From information theory perspective, Shannon entropy measures the expected information gain before observing the outcome (Baez, 2024), hence our intuition for the entanglement modules is to reduce this uncertainty by allowing hidden previous states to share information. These modules are designed to be causal, hence the computation of the output feature for time t only depends on the input hidden states up to that time step.

Chunked Low-Rank Attention. Given the hidden states $H \in \mathbb{R}^{N \times R}$, we divide each sequence of length N into $M = \lceil N/C \rceil$ non-overlapping chunks of size C. This summaries are linearly projected to obtain the keys and values. The chunked attention update is given by:

$$
\tilde{H} = H \; + \; \text{softmax}\Big(\frac{Q K_{\text{proj}}^{\top}}{\sqrt{R}} \; + \; \mathbf{C}\Big) V_{\text{proj}}, \tag{5}
$$

where $\mathbf{C}$ is the causal mask. For each hidden state at time t belonging to chunk j all entries with $m \geq j$ are masked, hence each query can attend only to summaries of earlier chunks conserving causality.

This implementation reduces the complexity from $O(N^2 R)$ to $O(NMR)$. Like Linformer (Wang et al., 2020), the method is low-rank in sequence length, but instead of learned projection matrices, it uses a pooling compression where each chunk summary is the mean of the TTLM hidden states within that window. These pooled representations provide a compact and non-local summary of the past context.

Causal Hadamard Pooling. Given hidden states $H \in \mathbb{R}^{N \times R}$, we use an Exponential Moving Average (EMA) vector $\mathbf{e}_t$ for each sequence

in the batch:

$$\mathbf{e}_t = \alpha\, \mathbf{e}_{t-1} + (1-\alpha)\, h_t, \qquad \alpha = \sigma(\lambda),$$

where λ is a learnable scalar shared across layers and σ denotes the sigmoid function. In this implementation, the EMA is a summary of all past hidden states, hence each new update depends only on the previous average and the current input. Each output state is updated by a Hadamard interaction with its EMA given by:

$$\tilde{h}_t = h_t + g_{\text{ent}} \left(h_t \odot \tfrac{1}{\sqrt{R}} \mathbf{e}_t\right), \tag{6}$$

where g_{ent} is a learnable gating parameter and $1/\sqrt{R}$ normalizes the interaction. A light feed-forward network finalizes the representation to get the output $\tilde{H}$. This computation scales linearly $O(NR)$, since the EMA is computed once per step and reused. Conceptually, the module performs a causal outer-product interaction between the current state and a low-rank summary of the past, allowing each position to integrate non-local information.

4.3 Prediction and Training

We obtain an enriched hidden sequence $\tilde{H} = [\tilde{h}_1; \ldots; \tilde{h}_N]$ which integrates both local and non-local context as described in Sections 4.2 and 4.1. Each state is projected to the vocabulary space with a linear decoder:

$$\ell_t = W_{\text{dec}}\, \tilde{h}_t + b_{\text{dec}}, \qquad p(y_t \mid x_{<t}) = \text{softmax}(\ell_t). \tag{7}$$

Here, $W_{\text{dec}} \in \mathbb{R}^{V \times R}$ and $b_{\text{dec}} \in \mathbb{R}^V$ are the parameters of the output projection, where V is the vocabulary size. The linear transformation maps the hidden state $\tilde{h}_t$ to the logits ℓ_t, which are then converted into a probability distribution over the vocabulary using a softmax function. The model is trained using standard cross-entropy loss over all time steps:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N} \sum_{t=1}^{N} \log p(y_t \mid x_{<t}). \tag{8}$$

This training objective optimizes the conditional probability of the next token.

5 Experimental Setup

We implemented our models in PyTorch (Paszke et al., 2019) using the PyTorch Lightning framework (Falcon and The PyTorch Lightning team, 2019). We evaluated our approach on a classic word-level and open domain language modeling task using (1) the Penn Treebank (PTB) dataset (Marcus et al., 1994), which contains 929k tokens for training, 73k for validation, and 82k for testing, with a vocabulary size of 10k unique words. PTB is suitable to evaluate compact language models with short context length. And (2) The WikiText-2 (WT2) (Merity et al., 2016), which contains 2.1M tokens for training, 218k for validation, and 246k for testing, with a vocabulary size of about 33k words. WT2 provides a larger and more natural vocabulary than PTB which makes it suitable to evalute language models for longer dependencies. The main evaluation metric is perplexity (PPL) on the test set, where lower perplexity indicates better language modeling performance. Specifically, we compare the following models:

1. TTLM Baselines variants: TTLM-small and TTLM-large (Su et al., 2024).
2. Modular TTLM (Ours): Modular TTLM-small and Modular TTLM-large.
3. Modular TTLM + Entanglement (Ours): Chunked Low-Rank Attention and Hadamard Pooling.
4. Transformer Baseline: Transformer with L blocks, multi head self-attention, pre-norm, learned positional embeddings. Configured to match the parameter count of the corresponding TTLM variant (Vaswani et al., 2017).

All models were trained under identical optimization settings for fairness. To compare TTLM baselines and our Modular TTLM, we used a learning rate of 0.001, dropout rate of 0.25, batch size of 32 and Adam optimizer. For our Modular TTLM variants, additional architectural hyperparameters were set to $\alpha_{\text{res}} = 0.5$ and use_tanh = False.

For the Modular TTLM + Entanglement aditional parameters were used such as AdamW optimizer with weight decay 0.2, dropout rate of 0.5, a 1000 steps warm up, a chunk size of 10 and entanglement_gate_init at 0.0.

To ensure a fair comparison, the Transformer baseline is sized to match the total parameter count of the corresponding Modular TTLM-Large variant. We choose the embedding dimension, number of layers, Heads and feed-forward width such that the amount of parameters closely matching the TTLM run at rank R. For example, at rank 60 we choose a $(1024, 7, 8, 2048)$ transformer run. For the matching at rank R the embedding dimension and number of layers were modified, Heads and feed-fordward width stayed constant at 8 and 2048.

All experiments were performed for up to 50 epochs with early stopping, gradient clipping at 0.25, a batch size of 32, sequence length (bptt) of 35, embedding size of 400 and a fixed random seed of 42 was used for all runs to ensure reproducibility. All experiments were conducted on one NVIDIA A100-SXM4-80GB GPU.

5.1 Scaling Law Evaluation

To analyse the scaling properties of the proposed Modular TTLM variants, we train each model with increasing TT-rank values ($R \in \{5, 10, 15, 20\}$) and report the corresponding test perplexity for a fair comparison with the baseline. This experiment highlights how the expressiveness of the Modular TTLM improves with rank, in analogy to scaling laws observed in large scale language models.

6 Results

6.1 Scaling with TT-Rank

We first study the effect of increasing TT-rank on language modeling performance. Table 1 reports test perplexity on PTB for the baselines variants (Su et al., 2024): Large *TTLM-L*, Small *TTLM-S* and our variants: Modular Large *M-TTLM-L*, Modular Small *M-TTLM-S*. Both modular models consistently outperform their corresponding baselines across sverall ranks, with the largest improvements observed at higher ranks. This shows that the proposed architectural enhancements improve the expresiveness capabilities of the model.

Table 2 shows the results on WikiText-2, where the same trend is observed: increasing rank lowers perplexity, Modular TTLM outperform the baselines. This shows the robustness of the models indicating that modeling global interactions is helpful for the larger WT2 vocabulary.

Rank	TTLM-L	TTLM-S	M-TTLM-L	M-TTLM-S
5	156.0	161.7	151.7	169.8
10	119.9	127.5	116.4	127.8
15	106.7	117.1	103.3	116.2
20	102.1	111.2	**96.7**	**109.2**

Table 1: Perplexity on PTB test set across TT-ranks.

Rank	TTLM-L	TTLM-S	M-TTLM-L	M-TTLM-S
5	129.1	132.9	127.8	140.0
10	100.4	109.4	100.0	109.3
15	89.4	100.9	88.1	100.8
20	85.5	96.7	**82.0**	**95.5**

Table 2: Perplexity on WikiText-2 test set across TT-ranks.

6.2 Effect of Entanglement Modules at Higher Ranks

We next compare our Modular TTLM-Large baseline (*M-TTLM-L*) against its entanglement-augmented variants: *M-TTLM-A* (chunked low-rank attention) and *M-TTLM-H* (Hadamard pooling) for higher ranks ($R \in \{40, 60, 70\}$). The scaling experiments in Section 6.1 showed that performance generally improves with rank, motivating our expectation that entanglement modules could further enhance expressiveness by capturing non-local context. Tables 3 and 4 show results on PTB and WikiText-2.

Both entanglement modules yield modest yet consistent improvements over our Modular TTLLM-Large, with *M-TTLM-A* performing best on PTB and WikiText-2. All our variants outperform the baseline TTLM-Large and Transformer on PTB. We attribute the poor improvement on WikiText-2 at higher ranks to the short context window used in our experiments ($bptt = 35$).

WikiText-2 contains longer sequences and a more diverse vocabulary than PTB, which likely makes it more sensitive to context windows. This could also explains why Transformer models perform better on this dataset. While improvements are modest, the results hint that causal non-local interactions might offer a way toward increasing the expressiveness of recurrent language modeling.

Rank	M-TTLM-L	M-TTLM-A	M-TTLM-H	Trans	TTLM-L
40	87.2	86.1	85.9	87.8	95.7
60	85.1	83.7	84.0	83.6	92.8
70	85.2	84.4	84.6	86.2	93.5

Table 3: Perplexity on PTB across higher ranks for Modular TTLM-Large (M-TTLM-L) and entanglement variants.

Rank	M-TTLM-L	M-TTLM-A	M-TTLM-H	Trans	TTLM-L
40	75.1	74.0	74.2	70.2	77.8
60	74.2	73.5	73.6	73.7	79.7
70	74.9	74.5	73.6	69.4	81.5

Table 4: Perplexity on WikiText-2 across higher ranks for Modular TTLM-Large (M-TTLM-L) and entanglement variants.

6.3 Efficiency Analysis

To evaluate computational requirements, we measure training wall time and GPU memory usage at TT-rank 60 for PTB and WikiText-2 (Table 5). We observe that the introduction of entanglement modules does not significantly increase training or inference GPU memory consumption which suggests that the proposed modules can be integrated with minimal computational cost while maintaining scalability as introduced in the formulation of the entanglement modules in the method section 4.

Our Rank 60 model demonstrates performance on PTB (Table 3) and WikiText-2 (Table 4) that is comparable to the Transformer baseline. Table 5 highlights the efficiency advantages, showing lower inference memory usage and similar or lower training memory usage compared to the Transformer. It is important to note that while parameter counts are comparable, the Transformer hyperparameters (embedding dimension, layers, heads) were chosen to match our TTLM model size. Performance may vary with different Transformer configurations, as Transformer performance is sensitive to these architectural choices.

6.4 Scaling Context Length on WikiText-2

To explore how our model performs and scales on a larger and diverse dataset like the WIkitext-2, we investigated the output of increasing the context length available to the model during training. We run separate instances of our Modular TTLM, with rank $R = 60$ a sequence lengths *bptt* of 128 and 256. For these runs, the chunk size for the low-rank attention entanglement was set to 16 and 32, respectively to investigates if there are increments on memory usage. Table 6 presents the peak GPU memory usage during training and inference as we increase the sequence length. When doubling the context length from 128 to 256, the peak memory usage for all models increases by a factor of approximately $1.5 - 1.7$. This observed scaling is substantially less than the quadratic increase expected from standard self-attention mechanisms and is consistent with the theoretically linear ($O(L)$) in our hybrid architecture for both M-TTLM-A and M-TTLM-H and the M-TTLM-L. The trend observed between $bptt = 128$ and $bptt = 256$ provides strong empirical evidence supporting the linear memory complexity claim for our proposed models.

Model	Params	Train (MB)	Infer (MB)
PTB (rank = 60)			
M-TTLM-L	49.3M	1047.7	281.7
M-TTLM-A	49.3M	1048.2	281.7
M-TTLM-H	49.3M	1048.0	281.7
Transformer	51.2M	1423.5	888.9
WikiText-2 (rank = 60)			
M-TTLM-L	117.4M	2600.4	696.7
M-TTLM-A	117.4M	2600.9	696.7
M-TTLM-H	117.4M	2600.7	696.7
Transformer	118.0M	2414.9	2051.8

Table 5: GPU memory usage for Modular TTLM-Large and its entanglement variants at TT-rank 60.

Model	Params	PPL	Train (MB)	Infer (MB)
WikiText-2 (bptt=128)				
M-TTLM-L	117.3M	67.20	2887.9	1353.4
M-TTLM-A	117.3M	66.42	2905.6	1353.3
M-TTLM-H	117.3M	66.58	2897.3	1353.4
WikiText-2 (bptt=256)				
M-TTLM-L	117.5M	66.49	4429.6	2258.3
M-TTLM-A	117.5M	65.38	4591.3	2258.9
M-TTLM-H	117.5M	65.95	4448.7	2258.9

Table 6: WikiText-2 perplexity, number of parameters, and GPU memory usage as a function of context length.

7 Conclusion

Our preliminary results indicate that our Hybrid approach for language modeling improves the expressiveness capabilities of TTLM

based architectures improving perplexity on the Penn Treebank and WikiText-2 Dataset. These gains come with little to no increase in GPU memory or wall time, consistent with our theoretical complexity analysis. This suggest that TTLMs could incorporate non-local context while retaining their quasi-linear memory scaling advantages over non recurrent models.

For future work, we plan to extend our evaluation to larger and diverse datasets. Additionally, we aim to explore a different range of entropy-based modules to study the implications of entanglement principles in modeling language structure.

References

Katsuya O Akamatsu, Kenji Harada, Tsuyoshi Okubo, and Naoki Kawashima. 2025. Plastic tensor networks for interpretable generative modeling. *arXiv preprint arXiv:2504.06722*.

John C Baez. 2024. What is entropy? *arXiv preprint arXiv:2409.09232*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3(Feb):1137–1155.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Jens Eisert. 2013. Entanglement and tensor network states. *arXiv preprint arXiv:1308.3318*.

William Falcon and The PyTorch Lightning team. 2019. PyTorch Lightning.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Xueyun Ma, Peng Zhang, Sheng Zhang, Nan Duan, Yue Hou, Ming Zhou, and Daxin Song. 2019. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems*, volume 32.

Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Ivan V Oseledets. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Zhenning Su, Yunlong Zhou, Fengwei Mo, and Jakob G. Simonsen. 2024. Language modeling using tensor trains. *arXiv preprint arXiv:2405.04590*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2022. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28.

Andrei Tomut, Seyed Saeed Jahromi, Abhinaba Sarkar, Ugur Kurt, Suraj Singh, Faraz Ishtiaq, Carlos Muñoz, Prateek S. Bajaj, Ahmed Elborady, Alberto Del Bimbo, et al. 2024. Compactifai: extreme compression of large language models using quantum-inspired tensor networks. *arXiv preprint arXiv:2401.14109*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Mengyuan Xu, Yulan Lin Xu, and Danilo P. Mandic. 2023. Tensorgpt: Efficient compression of large language models based on tensor-train decomposition. *arXiv preprint arXiv:2307.00526*.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297.

Shuai Zhang, Peng Zhang, Xindian Ma, Junqiu Wei, Ningning Wang, and Qun Liu. 2020. Tensorcoder: Dimension-wise attention via tensor representation for natural language modeling. *arXiv preprint arXiv:2008.01547*.