# Hierarchical Token Prepending: Enhancing Information Flow in Decoder-based LLM Embeddings

**Xueying Ding**[1*†], **Xingyue Huang**[2*†], **Mingxuan Ju**[3], **Liam Collins**[3], **Yozen Liu**[3], **Leman Akoglu**[1], **Neil Shah**[3], **Tong Zhao**[3]

[1]Carnegie Mellon University [2]University of Oxford [3]Snap Inc.

## Abstract

Large language models produce powerful text embeddings, but their causal attention mechanism restricts the flow of information from later to earlier tokens, degrading representation quality. While recent methods attempt to solve this by prepending a single summary token, they over-compress information, hence harming performance on long documents. We propose *Hierarchical Token Prepending* (HTP) [1], a method that resolves two critical bottlenecks. To mitigate attention-level compression, HTP partitions the input into blocks and prepends block-level summary tokens to subsequent blocks, creating multiple pathways for backward information flow. To address readout-level over-squashing, we replace last-token pooling with mean-pooling, a choice supported by theoretical analysis. HTP achieves consistent performance gains across 11 retrieval datasets and 30 general embedding benchmarks, especially in long-context settings. As a simple, architecture-agnostic method, HTP enhances both zero-shot and finetuned models, offering a scalable route to superior long-document embeddings.

## 1 Introduction

Neural text embeddings are central to tasks such as retrieval, recommendation, and clustering. Large language models (LLMs) have emerged as powerful embedding models, offering strong zero-shot performance and often outperforming traditional sentence encoders (Muennighoff et al., 2023; Thakur et al., 2021).

However, LLMs are optimized for auto-regressive token generation, not for producing sequence-level embeddings. Therefore, LLMs typically rely on causal attention instead of the bidirectional encoding used in traditional encoder-only models (Devlin et al., 2019; Liu et al., 2019). As a result, embedding generation suffers from *restricted backward flow* (Springer et al., 2025; Fu et al., 2024), as earlier tokens cannot access later positions in the sequence. The representations of earlier tokens cannot integrate later information, and hence degrade downstream performances.

Multiple recent works (Springer et al., 2025; Fu et al., 2024; Xu et al., 2024) have attempted to introduce such backward flow with minimal or no changes to model architectures, thereby preserving zero-shot capabilities. Echo embedding (Springer et al., 2025) addresses the issue at the *token level* by repeating the input sequence, allowing early tokens in the second copy can attend to later tokens in the first. While effective, doubling the sequence is computationally infeasible for long documents. In contrast, token prepending (TP) (Fu et al., 2024) is a *document level* solution that dynamically rewire the end-of-sequence (EOS) representation to the beginning of the input with a custom <PST> token before the next layer. This allows all tokens to attend to a compressed summary of the full sequence, yet degrades performance on long-context tasks due to over-compression. We therefore ask: *Can we preserve backward flow while alleviating information compression and maintaining scalability for long-context tasks?*

In this work, we investigate the information over-compression effect in long-context LLM embeddings generations with TP and identify two bottlenecks: (1) an **attention-level** bottleneck, where a single prepended token is forced to summarize the entire document, and (2) a **readout-level** bottleneck, where the final embedding is taken solely from the last token, resulting in an over-squashed representation (Barbero et al., 2024).

We propose *Hierarchical Token Prepending* (HTP), which mitigates over-compressing by replacing a single global <PST> token with a hierarchy of block-level summary tokens. HTP partitions the input into blocks and assigns each block a des-

[*]Equal contribution.

[†]Work done as intern in Snap Inc.

[1]https://github.com/snap-research/HTP



*Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 19047–19066

ignated summary token prepended to subsequent blocks, enabling backward flow through multiple pathways and alleviating the attention-level bottleneck. For the readout-level bottleneck, we replace last-token pooling with mean-pooling and show improved performance in long-context settings.

Our contributions can be summarized as follows:

- We propose *Hierarchical Token Prepending* (HTP), a block-level token prepending method that enables backward information flow with reduced over-compression.
- We show that mean pooling is better suited for long-context retrieval, supported by theoretical and empirical evidence.
- HTP consistently improves performance across 11 retrieval and 30 general embedding tasks in both standard and long-context settings, and also benefits finetuned models (e.g., *NV-Embed-v2* (Lee et al., 2025)).

## 2 Related Work

**Sentence Embeddings.** Modern text embeddings originated from bidirectional encoders such as BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019). While effective for token-level tasks, their raw representations are suboptimal for sentence similarity, motivating specialized training approaches such as supervised finetuning (SBERT (Reimers and Gurevych, 2019), Sentence-T5 (Ni et al., 2022)), contrastive learning (SimCSE (Gao et al., 2021), SNCSE (Chanchani and Huang, 2023)), and prompt-based tuning (PromptBERT (Jiang et al., 2022)). Despite their effectiveness, these methods rely on task-specific training or finetuning, limiting their general applicability.

**Finetuning LLMs for Embeddings.** Recent work has focused on adapting decoder-only LLMs for embedding tasks through architectural modifications and finetuning. LLM2Vec (BehnamGhader et al., 2024) and GRIT (Muennighoff et al., 2025) apply contrastive finetuning. Deelm (Li and Li, 2024) modify a decoder-only model to be bidirectional, resembling traditional encoders. Similarly, NVEmbed (Lee et al., 2025) and RepLLaMA (Ma et al., 2024) have demonstrated strong performance by specifically training LLMs with various tasks. Our proposed method is orthogonal to these approaches and can be applied to enhance the performance of these finetuned models, as in Section 5.4.

**Training-free LLM for Embeddings.** A key appeal of LLMs is their ability to generate powerful embeddings in a zero-shot, training-free manner. Literature proposed strategies such as optimizing prompts to elicit better representations (Jiang et al., 2024; Lei et al., 2024; Zhang et al., 2024; Cheng et al., 2025) and utilizing expert routers in MoE LLMs (Li and Zhou, 2025). However, a core challenge for all training-free methods is the *restricted backward flow* inherent from the causal attention mechanism of autoregressive models.Two recent methods address this limitation. Echo Embedding (Springer et al., 2025) duplicates the input to enable backward attention at the cost of doubled sequence length, while Token Prepending (Fu et al., 2024) introduces a single global summary token, improving efficiency but suffering from an information bottleneck on long documents. HTP overcomes this bottleneck by using hierarchical block-level summary tokens to enable robust backward information flow without over-compression.

## 3 Methodological Preliminaries

We begin with studying two factors affecting embedding quality in decoder-only LLMs: (i) the readout function for token aggregation and (ii) the need for backward dependencies in causal attention.

### 3.1 Mean vs. Last-token Embedding

Two common token-aggregation strategies are mean pooling and last-token pooling. Although last-token pooling is favored due to its alignment with autoregressive decoding and is computationally simple, it introduces a severe information bottleneck, is sensitive to prompt choice (Springer et al., 2025), and underperforms mean pooling in our retrieval experiments (Section 5).

To understand this performance gap, we show that mean-pooling is more robust to the "over-squashing" issue by distributing representational importance across all tokens instead of compressing it into a single position. Following sensitivity analysis adapted for decoder-only Transformers (Barbero et al., 2024; Topping et al., 2022), we quantify how the final embedding changes in response to a perturbation in an input token. Let $\mathbf{v}_i^{(0)} \in \mathbb{R}^d$ be the $i$-th input token embedding, and $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^d$ be the post-normalization representations at the final layer. We study the Jacobian norms $\left\| \frac{\partial \mathbf{y}_n}{\partial \mathbf{v}_i^{(0)}} \right\|$ for last-token readout and $\left\| \frac{\partial \bar{\mathbf{y}}}{\partial \mathbf{v}_i^{(0)}} \right\|$

for mean-token readout[2], where $\bar{\mathbf{y}} := \frac{1}{n}\sum_{j=1}^{n}\mathbf{y}_j$.

**Theorem 3.1** (Mean vs. Last-token Embedding)**.** *In a causal, decoder-only Transformer with $L$ layers, there exists a depth-dependent constant $K_L > 0$ and a nonnegative, lower-triangular, row-stochastic mixing matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ (capturing aggregate attention+residual flow across layers) such that, for every input position $i \in [n]$,*

$$\underbrace{\left\|\frac{\partial \mathbf{y}_n}{\partial \mathbf{v}_i^{(0)}}\right\|}_{\textit{last-token}} \leq K_L\,\mathbf{A}_{n,i}, \tag{1}$$

$$\underbrace{\left\|\frac{\partial \bar{\mathbf{y}}}{\partial \mathbf{v}_i^{(0)}}\right\|}_{\textit{mean-tokens}} \leq \frac{K_L}{n}\sum_{j=1}^{n}\mathbf{A}_{j,i}. \tag{2}$$

**Interpretation.** The last-token bound depends on a *single* entry $\mathbf{A}_{n,i}$, representing the influence of input token $i$ on the final token $n$, which can diminish rapidly with network depth. In contrast, the mean-pooling bound aggregates the *entire column* $\sum_j \mathbf{A}_{j,i}$ and is therefore topologically depth-agnostic up to the scale factor $K_L$, implying more robustness against over-squashing. We present the formal definitions and proof in Appendix A.

**Long-context Sentence Similarity (STS) Task.** We further empirically validate the impact of over-squashing on a modified STS task, where create long-context inputs by concatenating sentences with similar human-annotated scores (details in Appendix B). As shown in Figure 1, the performance of last-token embeddings degrades sharply as input length increases. While methods like PromptEOL (Jiang et al., 2024) offer some mitigation, they still cannot match the stability and superior performance of mean-pooling.

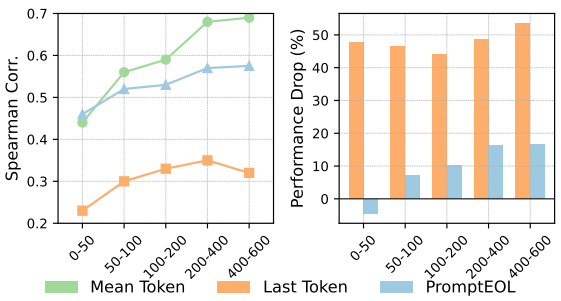
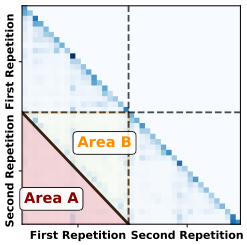



Figure 1: Left: Performance by embedding method across sentence lengths. Right: Performance drop of last-token and PromptEOL vs. mean at longer lengths.

[2] $\|\cdot\|$ denotes the Euclidean norm on vectors and the induced operator norm on Jacobians.

### 3.2 Backward Dependency

Causal attention in decoder-only LLMs restricts backward information flow. Input repetition methods (Springer et al., 2025; Xu et al., 2024) address this by allowing a "second pass" over the input, where the repeated sequence can attend to the original to form a more complete representation. Figure 2 showcases a masking experiment to isolate the effect of this backward dependency. We find that preventing the second-pass tokens from attending to the first-pass tokens significantly degrades STS performance, whereas masking the forward attention (from the first pass to the second) has a negligible impact. This result confirms that enabling backward information flow is critical for generating high-quality embeddings.




(a) Attention Map.

| Condition | STS Score |
|---|---|
| Echo Unmasked | **68.00** |
| + Mask Area A | 67.89 |
| + Mask Area B | 54.25 |

(b) Average scores across 4 STS tasks on Echo Mean Embeddings, with no-masking applied, masking Area A and masking Area B.

Figure 2: Disabling second-pass backward attention significantly degrades STS performance.

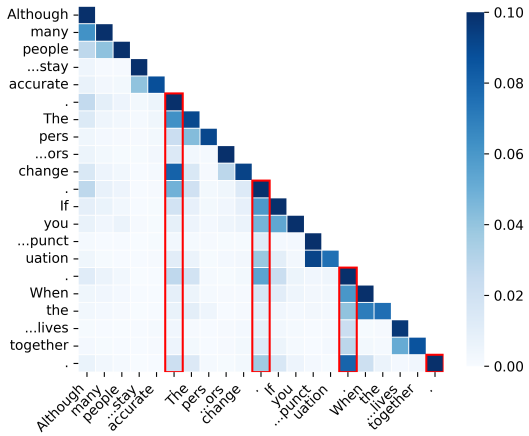



Figure 3: The attention map shows that End-of-Sentence (EOS) tokens are capturing information with more attention lookup to previous tokens.

## 4 Hierarchical Token Prepending

To mitigate the attention and readout bottlenecks in decoder-only models, we propose *Hierarchical Token Prepending* (HTP), a training-free method that establishes multi-level backward information flow. Figure 4 shows the three main stages of HTP:

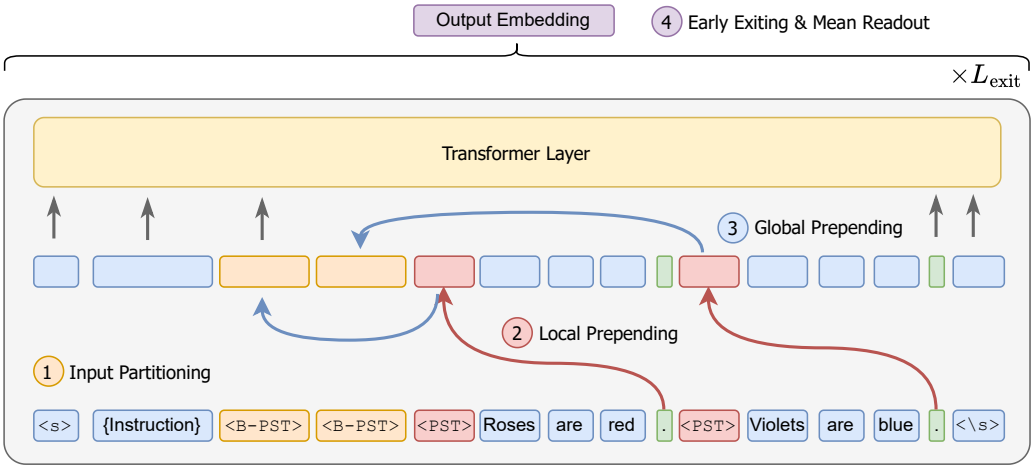



Figure 4: HTP partitions the input into blocks and creates a two-level summary. First, the hidden state of each block's final token is copied to a local summary token (<PST>). These local summaries are then aggregated into a global (<B-PST>) block at the front, making them accessible to all tokens. A mean readout with early exiting produces the final embedding.

1. **Input Partitioning:** The input text is segmented into semantic blocks (e.g., sentences), and placeholder summary <PST> tokens are inserted to create a hierarchical structure.
2. **Local Prepending:** Between Transformer layers, the final hidden state of each block is "rewired" to a corresponding local summary token, creating a block-level summary.
3. **Global Prepending:** These local summaries are then propagated to a global summary <B-PST> block at the beginning of the sequence, making them accessible to all tokens.

After the final layer, a mean-pooling readout is used to produce the output embedding. The following subsections elaborate over each step.

### 4.1 Input Partitioning

HTP begins by restructuring the input sequence to accommodate hierarchical summaries. Unlike methods that use a single summary token for entire text, HTP partitions the text into smaller semantic units to prevent information over-compression.

Given an input sequence of tokens $T = [t_1, \cdots, t_n]$, we first segment it into $M$ subsequences $(S_1, \cdots, S_M)$, $S_m$ with $m \in [M]$ represents a local context. This partition is defined by indices $0 = i_0 < i_1 < \cdots < i_M = n$, s.t. $S_m = (t_{i_{m-1}+1}, t_{i_{m-1}+2}, \ldots, t_{i_m})$ for $m \in [M]$.

We then augment the partitioned sequence with two types of special placeholder tokens to create the input sequence $T'$:

1. A local summary token, <PST>$_m$, is inserted before each subsequence $S_m$.
2. A block of $M$ global summary tokens, (<B-PST>$_1, \ldots,$ <B-PST>$_M$), is prepended to the beginning of the entire sequence.

The augmented input sequence $T'$ is:

$$T' = [\texttt{<B-PST>}_1, \ldots, \texttt{<B-PST>}_M, \texttt{<PST>}_1, S_1, \ldots, \texttt{<PST>}_M, S_M] \quad (3)$$

In practice, we define each subsequence $S_m$ by grouping every $K$ sentences, leveraging their natural semantic boundaries. This choice is motivated by observed attention patterns in decoder-only models, where the EOS tokens effectively aggregate information from preceding words (Figure 3). Thus, the final token's hidden state serves as an effective proxy for a sentence's summary. Unless otherwise specified, we set the hyperparameter $K = 1$.

### 4.2 Local Prepending

After partitioning, the Local Prepending step populates the <PST> placeholders with sentence-level summaries. This operation is performed dynamically between the layers of the Transformer.

Let $\mathbf{v}^{(\ell)} = (\mathbf{v}_1^{(\ell)}, \ldots, \mathbf{v}_{|T'|}^{(\ell)})$ be the sequence of hidden states entering layer $\ell$. Since the placeholder tokens are not in the model's trained vocabulary, their embeddings are randomly initialized for the first layer. For each subsequence layer $\ell > 1$, we apply a "rewiring" function, $f_{\text{local}}$, before the self-attention mechanism. This function copies the hidden state of the final token of each subsequence $S_m$ to the position of its corresponding local summary token, <PST>$_m$.

Formally, let $\mathsf{pos}(\cdot)$ be a function returning a token's index, and $\mathsf{end}(S_m)$ be the final token of subsequence $S_m$, which is typically sentence-ending punctuation (e.g., '.', '!', '?'). Let $\mathcal{P}_{\text{PST}} = \{\mathsf{pos}(\texttt{<PST>}_m)\}_{m=1}^{M}$ be the set of all local summary token positions, and the mapping $\mu : \mathcal{P}_{\text{PST}} \to [M]$ returns the sentence index $m$ for a given token position $i \in \mathcal{P}_{\text{PST}}$. The rewiring function $f_{\text{local}}$ for a hidden state $\mathbf{v}_i^{(\ell)}$ is defined as:

$$(\, f_{\text{local}}(\mathbf{v}^{(\ell)}))_i = \begin{cases} \mathbf{v}^{(\ell)}_{\mathsf{pos}(\mathsf{end}(S_{\mu(i)}))} & \text{if } i \in \mathcal{P}_{\text{PST}}, \\ \mathbf{v}_i^{(\ell)} & \text{otherwise.} \end{cases}$$

This rewired sequence, $\mathbf{v}'^{(\ell)} = f_{\text{local}}(\mathbf{v}^{(\ell)})$, is then fed into the attention block of layer $\ell$. This process ensures that each $\texttt{<PST>}_m$ token carries a summary of subsequence $S_m$, establishing a *sentence-level* backward dependency.

### 4.3 Global Prepending

Building on local summaries, global prepending enables document-level backward flow by propagating sentence summaries from <PST> tokens into a global <B-PST> block at the sequence start, creating multiple backward paths and avoiding a single-token bottleneck. We define a second rewiring function, $f_{\text{global}}$, which operates on the output of the local prepending step, $\mathbf{v}'^{(\ell)}$. $f_{\text{global}}$ copies the hidden state from each local summary token $\texttt{<PST>}_m$ to its corresponding global summary token $\texttt{<B-PST>}_m$. Let $\mathcal{P}_{\text{BPST}} = \{\mathsf{pos}(\texttt{<B-PST>}_m)\}_{m=1}^{M}$ denote the set for the global token positions, and mapping $\nu(i)$ returns the sentence index $m$ for a position $i \in \mathcal{P}_{\text{BPST}}$. The $f_{\text{global}}$ function is then defined as:

$$(\, f_{\text{global}}(\mathbf{v}'^{(\ell)}))_i = \begin{cases} \mathbf{v}'^{(\ell)}_{\mathsf{pos}(\texttt{<PST>}_{\nu(i)})} & \text{if } i \in \mathcal{P}_{\text{BPST}}, \\ \mathbf{v}'^{(\ell)}_i & \text{otherwise.} \end{cases}$$

The final sequence, $\mathbf{v}''^{(\ell)} = f_{\text{global}}(f_{\text{local}}(\mathbf{v}^{(\ell)}))$, is then fed into the attention block of layer $\ell$. This process allows any token to attend to summaries of all subsequent sentences by accessing the <B-PST> block, thereby enabling a comprehensive *document-level* backward flow.

### 4.4 Early Exit & Readout

Consistent with recent findings that intermediate layers often produce richer semantic representations (Fu et al., 2024; Skean et al., 2025; Liu et al., 2024; Jin et al., 2025), we employ an early-exit strategy. We select the output from a predetermined intermediate layer, $L'$. The final document embedding, $\bar{\mathbf{y}}^{(L')}$, is then computed by applying the mean-pooling readout, as justified in Section 3.1, over the fully rewired hidden states $\mathbf{v}''^{(L')}$.

## 5 Experiments

We conduct extensive experiments over HTP to answer following questions:

**Q1.** How does HTP compare with training-free LLM embedding baselines on retrieval task?
**Q2.** Does HTP perform well in general embedding benchmarks?
**Q3.** What is the effect of local prepending scale on retrieval performance?
**Q4.** Does HTP help with finetuned models?

### 5.1 BEIR Retrieval Task

**Setups.** We evaluate on a subset of commonly used BEIR (Thakur et al., 2021) retrieval datasets (**Q1**), including ArguAna (Wachsmuth et al., 2018), SciFact (Wadden et al., 2020), FiQA2018 (Maia et al., 2018), NFCorpus (Boteva et al., 2016), SCIDOCS (Cohan et al., 2020), HotpotQA (Yang et al., 2018), and TREC-COVID (Voorhees et al., 2021). We report NDCG@10, the time, and memory cost for each baseline methods.

**Baselines.** We evaluate three decoder-only LLMs: *Mistral-Instruct-7B-0.3* (Jiang et al., 2023), *Gemma2-9B* (Team et al., 2024), and *Qwen2-1.5B-Instruct* (Yang et al., 2024). For each model, we benchmark six embedding strategies: Vanilla Mean, Vanilla Last, Echo Mean (Springer et al., 2025), Token Prepending (TP) with PromptEOL (Jiang et al., 2024), TP with mean pooling, and HTP (Section 4). For HTP, we fix $K = 1$ across datasets and use spaCy for paragraph segmentation (Honnibal et al., 2020). All methods use instructions except TP with PromptEOL (see Appendix B).

Table 9 shows the LLM configuration for TP-based and HTP models. We briefly describe the version of LLM used, the starting and ending layers of the TP methods, and the early exit layer. For exit-layer selection, we use the develop sets of MSMARCO (7.44k corpus–query pairs) and HotpotQA (10.4k pairs) from BEIR (Thakur et al., 2021), optimizing for average NDCG@10 across both datasets. Following Fu et al. (2024), we do not tune the starting layer of HTP and fix it at the first layer.

For each LLM, we tune the exit layer independently via a coarse grid search over candidate layers $\{3, 7, 11, 16, 21, \dots\}$. As shown in the ablation study later, HTP is robust to the choice of exit

| Models | Method | ArguAna | SciFact | FiQA2018 | NFCorpus | SCIDOCS | HotpotQA | Trec-COVID | Memory | Time |
|---|---|---|---|---|---|---|---|---|---|---|
| Mistral-7B | Vanilla Mean | 45.66 | 42.10 | 8.02 | 9.03 | 2.80 | 8.80 | 26.02 | 1.00× | 1.00× |
| | Vanilla Last | 10.66 | 0.35 | 0.98 | 2.64 | 0.27 | 0.21 | 1.96 | 1.00× | 1.00× |
| | Echo Mean | 35.99 | 28.93 | 11.60 | 13.07 | 4.67 | 14.41 | **39.13** | 4.00× | 3.45× |
| | TP w. PromptEOL | 4.43 | 8.38 | 8.36 | 7.07 | 3.07 | 3.24 | 20.35 | 1.02× | 1.04× |
| | TP w. Mean | 42.41 | 36.71 | **12.72** | 8.13 | 3.30 | 9.84 | 25.67 | 1.02× | 1.15× |
| | HTP (Ours) | **47.06** | **46.67** | 8.77 | **15.51** | **6.08** | **16.02** | 33.65 | 1.12× | 1.18× |
| Gemma2-9B | Vanilla Mean | 42.70 | 49.13 | 6.27 | 12.97 | 3.80 | 13.36 | 10.69 | 1.00× | 1.00× |
| | Vanilla Last | 11.89 | 17.55 | 0.52 | 5.63 | 0.70 | 0.11 | 3.59 | 1.00× | 1.00× |
| | Echo Mean | 32.56 | 40.16 | 10.50 | 14.99 | 4.49 | 20.28 | 19.42 | 4.00× | 3.16× |
| | TP w. PromptEOL | 36.91 | 43.33 | **18.45** | 17.66 | 15.70 | **26.17** | **33.65** | 1.05× | 1.18× |
| | TP w. Mean | 42.14 | 52.73 | 8.89 | 15.02 | **16.50** | 14.94 | 21.22 | 1.06× | 1.16× |
| | HTP (Ours) | **43.64** | **53.72** | 10.76 | **18.33** | 11.16 | 18.84 | 25.16 | 1.18× | 1.20× |
| Qwen2-1.5B | Vanilla Mean | 34.11 | 27.16 | 3.26 | 3.90 | 4.25 | 3.04 | 11.16 | 1.00× | 1.00× |
| | Vanilla Last | 8.80 | 0.01 | 0.20 | 1.74 | 0.13 | 0.02 | 0.92 | 1.00× | 1.00× |
| | Echo Mean | 33.10 | 22.98 | **8.08** | 5.11 | **6.01** | **5.82** | 15.09 | 4.00× | 2.85× |
| | TP w. PromptEOL | 16.98 | 18.54 | 4.84 | **8.03** | 4.81 | 4.65 | 14.72 | 1.01× | 1.02× |
| | TP w. Mean | 21.66 | 18.31 | 2.52 | 5.89 | 3.91 | 5.23 | **21.78** | 1.01× | 1.03× |
| | HTP (Ours) | **36.01** | **28.29** | 4.06 | 5.43 | 4.85 | 5.53 | 18.35 | 1.10× | 1.08× |

Table 1: NDCG@10 (in percentage) on Retrieval Tasks from MTEB Retrieval Benchmarks. We **bold** the top one and underline the runner-up. We also report the additional memory and running time incurred from Vanilla method.

layer: while performance varies across layers, the differences are generally small, indicating stable behavior even when the exit point shifts.

**Results.** Table 1 summarizes the results. Addressing **Q1**, HTP matches or outperforms other training-free LLM embedding baselines, frequently ranking first or second across datasets. HTP achieves performance comparable to Echo Mean (Springer et al., 2025) with substantially lower memory and latency. It also consistently outperforms Token Prepending (TP) (Fu et al., 2024) with either PromptEOL (Jiang et al., 2024) or mean pooling, due to its local and global lookup tokens (<PST>$_m$, <B-PST>$_m$) that mitigate information squashing. *Gemma2-9B* performs strongest overall, likely due to its larger scale. Finally, last-token embeddings generally underperform mean pooling, except for *Gemma2-9B* with TP and PromptEOL, suggesting that PromptEOL acts as a soft prompt that encourages implicit information flow.

### 5.2 LongEmbed Retrieval Tasks

**Setups.** We evaluate performance of models on four real-world tasks in LongEmbed (Zhu et al., 2024), which features documents of longer length and dispersed target information (**Q1**). The four tasks are QMSum (Zhong et al., 2021), 2WikiMultiHopQA (Ho et al., 2020), SummScreenFD (Chen et al., 2021), NarrativeQA (Kočiskỳ et al., 2018), and a detailed description of the dataset is in Table 17. We evaluate on *Gemma2-9B* and *Mistral-instruct-7B-0.3*, and over the same extraction strategies as in the previous section. We again report NDCG@10 and running time for the evaluation. We primarily use two context lengths: 512, a commonly used length in the embedding model literature (Springer et al., 2025; Lee et al., 2025; BehnamGhader et al., 2024), and an extended length of 8192. We also evaluate models with additional context lengths and report their performance for comparison. Across all models, we fix one single instruction prompt: *"Retrieve relevant document. {text}"* (except for PromptEOL (Jiang et al., 2024) where the prompt is fixed to *"The paragraph {text} means in one word:"*). For our HTP model, we simply use $K = 1$ across all datasets.

**Results.** Table 2 presents results for context lengths of 512 and 8,192, while Figure 5 shows the average NDCG@10 across four datasets for a wider range of context lengths. At both context lengths, HTP achieves strong performance. It notably outperforms other methods at 8,192 and 16,384, while incurring lower runtime than Echo Embedding (Springer et al., 2025) (**Q1**). Furthermore, it shows improved performance with longer context lengths, likely due to local token prepending preserving more local information and reducing information oversquashing. Figure 5 shows that, across both *Mistral* and *Gemma2* models, the top-performing methods (HTP, Vanilla, TP Mean) rely on mean embedding, supporting our earlier claim that it captures more information and offers greater stability for longer contexts.

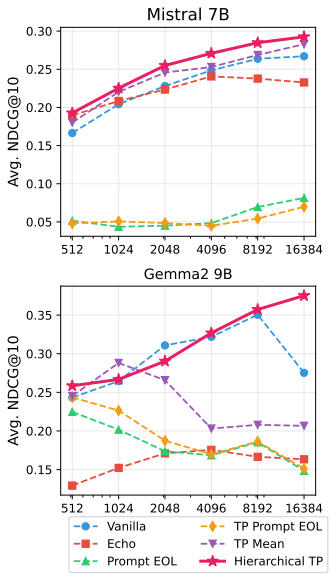



Figure 5: Avg. NDCG@10 across different context lengths.

| Models | Method | CXT Len | QMSum | 2WikiMQA | SumFD | NQA | Avg. Time(s) |
|---|---|---|---|---|---|---|---|
| Mistral-7B | Vanilla Mean | 512 | 11.98 | 15.29 | 36.81 | 2.50 | 73.0 (1.00×) |
| | Echo Mean | 512 | **14.27** | **22.50** | 32.03 | 7.05 | 135.0 (1.84×) |
| | PromptEOL | 512 | 4.57 | 7.16 | 6.07 | 2.80 | 90.5 (1.25×) |
| | TP w. PromptEOL | 512 | 5.44 | 6.51 | 5.52 | 1.72 | 94.5 (1.28×) |
| | TP w. Mean | 512 | 11.97 | 18.62 | **38.50** | 3.08 | 93.7 (1.27×) |
| | HTP (Ours) | 512 | 13.22 | 20.17 | 35.63 | **8.09** | 98.2 (1.34×) |
| Gemma2-9B | Vanilla Mean | 512 | 19.19 | 23.12 | 48.54 | 6.45 | 86.5 (1.00×) |
| | Echo Mean | 512 | 9.73 | 16.30 | 21.54 | 4.14 | 170.5 (2.01×) |
| | PromptEOL | 512 | 13.90 | 27.35 | 32.90 | 15.74 | 132.5 (1.56×) |
| | TP w. PromptEOL | 512 | 14.43 | **30.86** | 36.16 | **15.89** | 118.3 (1.40×) |
| | TP w. Mean | 512 | 20.45 | 21.73 | **48.88** | 7.04 | 103.0 (1.22×) |
| | HTP (Ours) | 512 | **21.89** | 22.90 | 47.78 | 10.84 | 107.7 (1.27×) |
| Mistral-7B | Vanilla Mean | 8,192 | **24.18** | 24.02 | 53.88 | 3.42 | 236.7 (1.00×) |
| | Echo Mean | 8,192 | 17.29 | **29.78** | 39.27 | 8.77 | 818.5 (3.50×) |
| | PromptEOL | 8,192 | 6.36 | 7.49 | 9.86 | 4.08 | 245.5 (1.46×) |
| | TP w. PromptEOL | 8,192 | 4.56 | 6.89 | 6.86 | 3.42 | 256.5 (1.05×) |
| | TP w. Mean | 8,192 | 23.08 | 23.11 | **56.50** | 4.78 | 241.0 (1.02×) |
| | HTP (Ours) | 8,192 | 23.43 | 27.88 | 53.53 | **9.07** | 265.9 (1.50×) |
| Gemma2-9B | Vanilla Mean | 8,192 | 29.85 | 34.62 | 66.69 | 7.02 | 322.5 (1.00×) |
| | Echo Mean | 8,192 | 10.23 | 17.15 | 28.52 | 5.76 | 832.1 (2.58×) |
| | PromptEOL | 8,192 | 9.66 | 21.84 | 30.74 | **11.91** | 322.7 (1.00×) |
| | TP w. PromptEOL | 8,192 | 9.71 | 23.32 | 29.80 | 11.70 | 336.3 (1.03×) |
| | TP w. Mean | 8,192 | 18.29 | 19.20 | 29.22 | 7.21 | 337.5 (1.03×) |
| | HTP (Ours) | 8,192 | **30.22** | **35.19** | **67.06** | 10.42 | 350.4 (1.08×) |

Table 2: NDCG@10 (in percentage) and avg. running time on LongEmbed. we report the context length of 512 and an extended length of 8,192.

### 5.3 General Embedding Tasks

**Setups.** Beyond retrieval tasks, we assess the embeddings' quality on a wider range of downstream tasks from the Massive Text Embedding Benchmark (MTEB) (Muennighoff et al., 2023). We evaluate across six task categories: Classification (11 datasets), Reranking (3 datasets), Clustering (11 datasets), and Semantic Textual Similarity (STS) (5 datasets), and BEIR Retrieval tasks (7 datasets) (**Q2**). We report the average performance for each category using *Mistral-Instruct-7B-0.3*, and compare HTP with Echo Mean, PromptEOL, and TP with PromptEOL, which are described in detail in Section 5.1. For the Echo Mean and HTP methods, we prepend commonly used instruction-style prompts to the texts (except for STS), formatted as *"{instruct} {text}"*. For the other two methods, we use the same PromptEOL (Jiang et al., 2024) prompt: *"The sentence means {text} in one word."* Details are provided in Appendix B.2.

| Task | Echo Mean | Prompt-EOL | TP w. PromptEOL | HTP (Ours) |
|---|---|---|---|---|
| Cls (11 datasets) | 66.12 | 68.12 | **69.52** | 68.44 |
| Rerank (3 datasets) | **43.31** | 40.35 | 40.57 | 40.85 |
| Cluster (11 datasets) | 34.07 | 23.10 | 22.11 | **35.39** |
| STS (5 datasets) | 64.46 | 67.89 | **68.46** | 53.64 |
| Retrieval (7 datasets) | 21.11 | 8.15 | 7.84 | **24.82** |

Table 3: Average performance on general embedding tasks. We **bold** the top one and underline the runner-up.

**Results.** We present the results in Table 3 (individual results in Appendix B.2.2). To address **Q2**, we observe that HTP performs well on classification, reranking, and clustering tasks. However, it lags behind other methods on sentence similarity tasks. We attribute this to the fact that these tasks generally do not involve long sequences, as STS tasks focus on intra-sentence similarity. Hence, considering PromptEOL (Jiang et al., 2024) excels by summarizing each sentence with a single representative token, it is much more suited for such fine-grained comparisons.

### 5.4 Ablations over HTP

**Effect of Hyperparameter $K$.** The partition size $K$ (sentences per summary block) trades off detail and coherence. To study HTP's sensitivity to local-prepending scale (**Q3**), we vary $K$ on short-document datasets (NFCorpus, SciFact; max length 512) and long-document datasets (SummScreenFD, 2WikiMultipleQA; max length 16,382). Note that when $K$ is equal to document length, our HTP is TP w. Mean, since we only append one token. As shown in Figure 6, the results reveal distinct trends based on context length. For short documents, performance degrades as $K$ increases, suggesting that coarser summaries may elide critical, fine-grained details necessary for the task. Conversely, for long documents, performance improves with a larger $K$. We attribute this to two factors: (1) larger sentence blocks yield more coherent semantic summaries in lengthy, multi-topic contexts, and (2) a very small $K$ in a long document creates an excessive number of <B-PST> tokens, which can push the model into

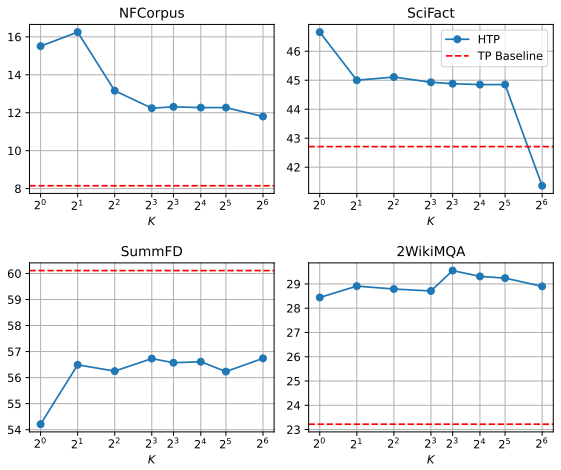



Figure 6: NDCG@10 performance of HTP with various $K$.

| Model | NFCorpus | SciFact | FiQA2018 | ArguAna | SCIDOCS |
|---|---|---|---|---|---|
| HTP (Ours) | **15.51** | **46.67** | 8.77 | **47.06** | **6.08** |
| Local Only | 10.03 | 43.39 | 5.87 | 46.40 | 3.04 |
| Global Only | 8.13 | 36.71 | **12.72** | 42.41 | 3.30 |

Table 4: NDCG@10 for HTP and variants with only global or local prepending on the BEIR subset. Best results are in **bold**.

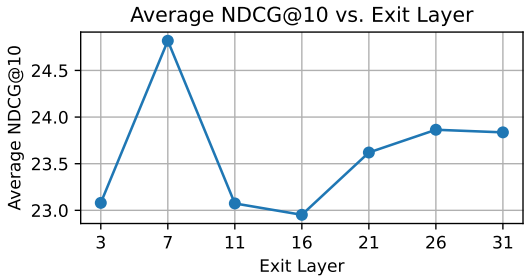



Figure 7: Avg NDCG@10 on across different exit layer.

out-of-distribution behavior. This suggests the optimal partitioning strategy is dependent on document length and task granularity.

**Sentences vs. Every Few Tokens.** We also conduct an ablation to validate our choice of partitioning the input along semantic boundaries (i.e., sentences) rather than at arbitrary fixed-length intervals. We compare our standard approach against a baseline that inserts <PST> tokens every $N$ tokens. To ensure a fair comparison, we dynamically set $N$ for each document to match the average sentence length in tokens, thereby keeping the total number of inserted <PST> tokens identical between the two methods (i.e., $N = \left\lfloor \frac{\texttt{num\_tokens}}{\texttt{n\_sentence}} \right\rfloor$).

The results, shown in Figure 8 for the NFCorpus and SciFact datasets, demonstrate that sentence-based partitioning consistently outperforms the fixed-interval baseline. This supports our hypothesis that leveraging natural linguistic boundaries allows the final token of a sentence to capture a more meaningful and coherent semantic summary.

**Prepending Only Locally and Globally.** We ablate HTP by removing either global or local prepending: **Global only** removes sentence-level <PST> tokens, while **Local only** removes global <B-PST> tokens. As shown in Table 4, removing either component consistently degrades performance, confirming that both global and local backward lookup are essential for HTP.

**Early Readout.** We perform an additional ablation that disables the Early Readout mechanism. Table 5 shows that HTP still improves over Vanilla Mean/Last, which demonstrates that performance improvement is due to enhanced information flows by token prependings and backward lookup.

| Model | NFCorpus | SciFact | FiQA2018 | ArguAna | SCIDOCS |
|---|---|---|---|---|---|
| HTP (Ours) | **15.51** | **46.67** | **8.77** | **47.06** | **6.08** |
| Mean w/o E.R. | 9.03 | 42.10 | 8.02 | 45.66 | 2.80 |
| Mean w. E.R. | 10.31 | 43.49 | 7.88 | 41.39 | 5.18 |
| Last w. E.R. | 3.29 | 1.20 | 0.60 | 4.45 | 0.17 |

Table 5: NDCG@10 comparison between HTP and vanilla baselines with Early Readout (E.R.). Best results are in **bold**.

**Start and Exit Layer.** Figure 7 evaluates different exit layers with the starting layer fixed at the first LLM layer. Exiting too early (e.g., layer 3) degrades performance relative to our default setting (layer 7), while exiting much later or disabling early exit also hurts performance on most datasets. The overall impact is modest, indicating that HTP is robust to the choice of start and exit layers, aligning with Fu et al. (2024). (See full results and ablation on starting layers in Tables 6 and 7).

### 5.5 HTP on Finetuned Embedding Models

Finally, we answer **(Q4)** by modifying over *NV-Embed-v2* (Lee et al., 2025), a finetuned model for embedding tasks, and show that HTP can yield more performance. *NV-Embed-v2* is a general-purpose embedding model based on *Mistral-7B*, enhanced with bi-directional attention and a novel attention aggregation mechanism in the final layer. As a result, we do not enable early-layer extraction or mean pooling as in the previous HTP setup (see Appendix B.1.1 for HP details). Table 8 presents the results of *NV-Embed-v2* with existing training-free methods. HTP consistently boosts perfor-

| Start / Exit Layer | NFCorpus | SciFact | FiQA2018 | ArguAna | SCIDOCS |
|---|---|---|---|---|---|
| Start 1, Exit 3 | 15.53 | 44.06 | 7.37 | 43.30 | 5.14 |
| Start 1, Exit 7 (Current HTP) | 15.51 | **46.67** | **8.77** | **47.06** | **6.08** |
| Start 1, Exit 11 | 14.84 | 44.61 | 7.58 | 43.51 | 4.83 |
| Start 1, Exit 16 | 14.49 | 44.16 | 7.56 | 43.49 | 5.06 |
| Start 1, Exit 21 | 17.50 | 43.91 | 8.06 | 43.38 | 5.25 |
| Start 1, Exit 26 | **18.50** | 44.01 | 8.12 | 43.39 | 5.30 |
| Start 1, Exit 31 (No early exit) | 18.49 | 43.90 | 8.11 | 43.33 | 5.35 |

Table 6: NDCG@10 for HTP with different exit layers.

| Start / Exit Layer | NFCorpus | SciFact | FiQA2018 | ArguAna | SCIDOCS |
|---|---|---|---|---|---|
| Start 1, Exit 7 (Current HTP) | 15.51 | **46.67** | **8.77** | **47.06** | **6.08** |
| Start 11, Exit 17 | **16.30** | 44.77 | 8.09 | 43.74 | 5.44 |
| Start 25, Exit 31 | 11.70 | 42.71 | 8.01 | 43.60 | 4.42 |

Table 7: NDCG@10 for HTP with different start and exit layers.

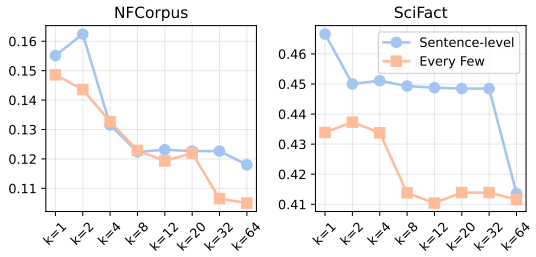



Figure 8: Comparison of Sentence vs. Every Few tokens.

mance over the base *NV-Embed-v2* model across all three datasets. This result suggests that the gains from HTP are orthogonal to those achieved through finetuning. Furthermore, HTP demonstrates competitive performance against the other training-free methods, showing its robustness.

| Model | NFCorpus | FiQA2018 | SciFact |
|---|---|---|---|
| NV-Embed | 45.10 | 62.63 | 80.92 |
| + Echo | 40.88 | 48.35 | 74.08 |
| + TP Mean | <u>46.60</u> | <u>62.68</u> | <u>81.64</u> |
| + HTP | **47.26** | **64.18** | **82.52** |

Table 8: Performance on NV-Embed. We **bold** the top one and <u>underline</u> the runner-up.

## 6 Conclusion

We introduce *Hierarchical Token Prepending* (HTP), a simple, training-free method that alleviates information bottlenecks in LLM embeddings. With hierarchical block-level summary tokens, HTP enables multiple backward information paths and reduces attention compression. Combined with mean pooling, it delivers consistent gains across benchmarks and improves both zero-shot and finetuned models, making it a scalable and versatile text-embedding approach.

## Limitations

While HTP significantly improves zero-shot embeddings from decoder-only LLMs, its performance is not expected to surpass models that are extensively finetuned specifically for retrieval tasks. Our initial experiments showed that HTP can enhance an existing finetuned model (NV-Embed (Lee et al., 2025)), but a broader investigation is needed to understand its interaction with diverse model architectures and training paradigms. A deeper look into the mechanisms of backward dependency and token prepending is also warranted.

## Acknowledgment

We thank Yuya Asano at University of Pittsburgh for helpful discussions and idea development. The authors also used ChatGPT to proofread author-generated text in this paper.

## References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *Preprint*, arXiv:1607.06450.

Federico Barbero, Andrea Banino, Steven Kapturowski, Dharshan Kumaran, João Guilherme Madeira Araújo, Alex Vitvitskyi, Razvan Pascanu, and Petar Veličković. 2024. Transformers need glasses! information over-squashing in language tasks. In *NeurIPS*.

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. LLM2vec: Large language models are secretly powerful text encoders. In *CoLM*.

Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. 2016. A full-text learning to rank dataset for medical information retrieval. In *European Conference on Information Retrieval*.

Sachin Chanchani and Ruihong Huang. 2023. Composition-contrastive learning for sentence embeddings. In *ACL*.

Mingda Chen, Zewei Chu, Sam Wiseman, and Kevin Gimpel. 2021. Summscreen: A dataset for abstractive screenplay summarization. In *ACL*.

Zifeng Cheng, Zhonghui Wang, Yuchen Fu, Zhiwei Jiang, Yafeng Yin, Cong Wang, and Qing Gu. 2025. Contrastive prompting enhances sentence embeddings in llms through inference-time steering. *arXiv preprint arXiv:2505.12831*.

Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel Weld. 2020. SPECTER: Document-level representation learning using citation-informed transformers. In *ACL*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

Yuchen Fu, Zifeng Cheng, Zhiwei Jiang, Zhonghui Wang, Yafeng Yin, Zhengliang Li, and Qing Gu. 2024. Token prepending: A training-free approach for eliciting better sentence embeddings from llms. In *ACL*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *EMNLP*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*.

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spacy: Industrial-strength natural language processing in python.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Ting Jiang, Shaohan Huang, Zhongzhi Luan, Deqing Wang, and Fuzhen Zhuang. 2024. Scaling sentence embeddings with large language models. In *EMNLP*.

Ting Jiang, Jian Jiao, Shaohan Huang, Zihan Zhang, Deqing Wang, Fuzhen Zhuang, Furu Wei, Haizhen Huang, Denvy Deng, and Qi Zhang. 2022. PromptBERT: Improving BERT sentence embeddings with prompts. In *EMNLP*.

Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, Fan Yang, Mengnan Du, and Yongfeng Zhang. 2025. Exploring concept depth: How large language models acquire knowledge and concept at different layers? In *Proceedings of the 31st International Conference on Computational Linguistics*.

Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2018. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*.

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. 2025. NV-embed: Improved techniques for training LLMs as generalist embedding models. In *ICLR*.

Yibin Lei, Di Wu, Tianyi Zhou, Tao Shen, Yu Cao, Chongyang Tao, and Andrew Yates. 2024. Meta-task prompting elicits embeddings from large language models. In *ACL*.

Xianming Li and Jing Li. 2024. BeLLM: Backward dependency enhanced large language model for sentence embeddings. In *NAACL*.

Ziyue Li and Tianyi Zhou. 2025. Your mixture-of-experts LLM is secretly an embedding model for free. In *ICLR*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Zhu Liu, Cunliang Kong, Ying Liu, and Maosong Sun. 2024. Fantastic semantics and where to find them: Investigating which layers of generative LLMs reflect lexical semantics. In *Findings of ACL*.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2024. Fine-tuning llama for multi-stage text retrieval. In *SIGIR*.

Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. 2018. Www'18 open challenge: financial opinion mining and question answering. In *Companion proceedings of the the web conference*.

Niklas Muennighoff, Hongjin SU, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. 2025. Generative representational instruction tuning. In *ICLR*.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. Mteb: Massive text embedding benchmark. In *EACL*.

Jianmo Ni, Gustavo Hernandez Abrego, Noah Constant, Ji Ma, Keith Hall, Daniel Cer, and Yinfei Yang. 2022. Sentence-t5: Scalable sentence encoders from pretrained text-to-text models. In *ACL Findings*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*.

Oscar Skean, Md Rifat Arefin, Dan Zhao, Niket Nikul Patel, Jalal Naghiyev, Yann LeCun, and Ravid Shwartz-Ziv. 2025. Layer by layer: Uncovering hidden representations in language models. In *ICML*.

Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. 2025. Repetition improves language model embeddings. In *ICLR*.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, and 179 others. 2024. Gemma 2: Improving open language models at a practical size. *Preprint*, arXiv:2408.00118.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS Datasets and Benchmarks Track*.

Jake Topping, Francesco Di Giovanni, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. 2022. Understanding over-squashing and bottlenecks on graphs via curvature. In *ICLR*.

Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. 2021. Trec-covid: constructing a pandemic information retrieval test collection. In *ACM SIGIR Forum*.

Henning Wachsmuth, Shahbaz Syed, and Benno Stein. 2018. Retrieval of the best counterargument without prior topic knowledge. In *ACL*.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. 2020. Fact or fiction: Verifying scientific claims. In *EMNLP*.

Xiaohan Xu, Chongyang Tao, Tao Shen, Can Xu, Hongbo Xu, Guodong Long, Jian-Guang Lou, and Shuai Ma. 2024. Re-reading improves reasoning in large language models. In *EMNLP*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, and 43 others. 2024. Qwen2 technical report. *Preprint*, arXiv:2407.10671.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *EMNLP*.

Bowen Zhang, Kehua Chang, and Chunping Li. 2024. Simple techniques for enhancing sentence embeddings in generative language models. In *ICIC*.

Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan Awadallah, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, and Dragomir Radev. 2021. QMSum: A New Benchmark for Query-based Multi-domain Meeting Summarization. In *NAACL*.

Dawei Zhu, Liang Wang, Nan Yang, Yifan Song, Wenhao Wu, Furu Wei, and Sujian Li. 2024. LongEmbed: Extending embedding models for long context retrieval. In *EMNLP*.

# A Proofs

We follow Barbero et al. (2024) and study the over-squashing effect of the readout. In particular, we conduct *sensitivity analysis* original introduced in the context of graph learning (Topping et al., 2022), which aim to study the quantity $\partial \mathbf{y}_n / \partial \mathbf{v}_i^{(0)}$, i.e., the partial derivative of final output embedding (last token) with respect to the $i$-th token. We first define the studied decoder-only transformer architectures, which encompass the majority of the LLMs currently in use.

## A.1 Transformer Architectures

**Notation.** We write $[n] = \{1, \dots, n\}$. We use the Euclidean norm $\|\cdot\|$ for vectors and the induced operator (spectral) norm for Jacobians. $\delta_{ij}$ represents *Kronecker delta*: $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. We denote the $i$-th standard basis vector by $\mathbf{e}_i$ and the all-ones vector by $\mathbf{1}$.

**Setup.** We follow the model specification used by Barbero et al. (2024), and study decoder-only, causal Transformers of $d$ dimension on a length-$n$ sequence of token states $\mathbf{v}^{(0)} = (\mathbf{v}_1^{(0)}, \dots, \mathbf{v}_n^{(0)})$ with $\mathbf{v}_i^{(0)} \in \mathbb{R}^d$. We let $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$ be the query, key, and value matrices for a sequence of $n$ tokens with $d$-dimensional embeddings. We denote the $i$-th token's query, key, and value vectors by $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i \in \mathbb{R}^d$. Let $\mathbf{p}_{ij} \in \mathbb{R}^{2e}$ be the $2e$ dimensional vectors encoding positional information between positions $i$ and $j$. We assume $\sup_{i,j} \|\mathbf{p}_{ij}\| \le P_{\max} < \infty$ (bounded positional encodings), which holds for most positional schemes used in practice, such as Rotational positional encoding (RoPE) (Su et al., 2024).

**Single-head Pre-LN block.** For layer $\ell \in \{0, \dots, L-1\}$ and position $i \in [n]$, define

$$\mathbf{z}_i^{(\ell)} = \mathbf{v}_i^{(\ell)} + \sum_{j \le i} \alpha_{ij}^{(\ell)} \, \mathrm{norm}_1^{(\ell)}\big(\mathbf{v}_j^{(\ell)}\big), \tag{Attn}$$

$$\alpha_{ij}^{(\ell)} = \frac{\exp\Big(k\big(\mathbf{q}_i^{(\ell)}, \mathbf{k}_j^{(\ell)}, \mathbf{p}_{ij}\big)\Big)}{\sum_{w \le i} \exp\Big(k\big(\mathbf{q}_i^{(\ell)}, \mathbf{k}_w^{(\ell)}, \mathbf{p}_{iw}\big)\Big)}, \quad j \le i, \qquad \alpha_{ij}^{(\ell)} = 0 \text{ if } j > i, \tag{Softmax}$$

$$\mathbf{v}_i^{(\ell+1)} = \mathbf{z}_i^{(\ell)} + \psi^{(\ell)}\Big(\mathrm{norm}_2^{(\ell)}\big(\mathbf{z}_i^{(\ell)}\big)\Big), \tag{MLP}$$

where $k : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^{2e} \to \mathbb{R}$ is a scoring function (e.g. a bilinear form with a positional bias), $\psi^{(\ell)} : \mathbb{R}^d \to \mathbb{R}^d$ is the MLP at layer $\ell$, and $\mathrm{norm}_1^{(\ell)}, \mathrm{norm}_2^{(\ell)} : \mathbb{R}^d \to \mathbb{R}^d$ are the (pre-activation) normalization maps (typically LayerNorm (Ba et al., 2016)). Causality is enforced by the mask $j \le i$ in (Attn)–(Softmax), hence $\mathbf{v}_j^{(\ell)}$ depends only on $\{\mathbf{v}_i^{(\ell-1)} : i \le j\}$.

**Attention Matrix.** Let $\mathbf{\Lambda}^{(\ell)} \in \mathbb{R}^{n \times n}$ collect attention weights with $(\mathbf{\Lambda}^{(\ell)})_{ij} = \alpha_{ij}^{(\ell)}$. Each $\mathbf{\Lambda}^{(\ell)}$ is *row–stochastic* ($\mathbf{\Lambda}^{(\ell)}\mathbf{1} = \mathbf{1}$) and *lower–triangular* ($(\mathbf{\Lambda}^{(\ell)})_{ij} = 0$ if $j > i$). This lower–triangular, row–stochastic structure is preserved under products.See Lemmas B.6–B.7 in Barbero et al. (2024).

**Final Normalization and Readouts.** After layer $L$, we set

$$\mathbf{y}_i = \mathrm{norm}_3\big(\mathbf{v}_i^{(L)}\big), \qquad i \in [n].$$

where $\mathrm{norm}_3 : \mathbb{R}^d \to \mathbb{R}^d$ is a normalization (often LayerNorm). For last-token embedding, the readout usually uses the *last token* $\mathbf{y}_n$; for mean-pooling, we set $\bar{\mathbf{y}} = \frac{1}{n} \sum_{i=1}^n \mathbf{y}_i$.

**Assumptions.** Following Barbero et al. (2024), we adopt the following simplifications for layerwise sensitivity bounds. We write $\mathrm{Lipschitz}(f)$ for the Lipschitz constant of a map $f$, i.e., any $L$ such that

$$\|f(x) - f(y)\| \le L \, \|x - y\|, \qquad \forall x, y.$$

1. During differentiation, the attention weights $\alpha_{ij}^{(\ell)}$ are treated as input–independent constants.

2. Each normalization operator has a Lipschitz bound with known scalings, with

$$\mathrm{Lipschitz}\big(\mathrm{norm}_1^{(\ell)}\big) \;\le\; \frac{1}{\beta_1^{(\ell)}}, \quad \mathrm{Lipschitz}\big(\mathrm{norm}_2^{(\ell)}\big) \;\le\; \frac{1}{\beta_2^{(\ell)}}, \quad \mathrm{Lipschitz}\big(\mathrm{norm}_3\big) \;\le\; \frac{1}{\beta_3},$$

for some $\beta_1^{(\ell)}, \beta_2^{(\ell)}, \beta_3 > 0$.

3. The MLP in each layer $\psi^{(\ell)}$ admits a Lipschitz constant $\sigma_\psi^{(\ell)}$.

**Remark A.1** (Multi-head and projections)**.** *The statements below naturally extend to $H$ heads by stacking the single-head updates in parallel and absorbing the output projection into the constants $\sigma_\psi^{(\ell)}$ and $\beta_2^{(\ell)}$; all proofs go through with the same structure because the causal, residual, and row-stochastic properties are preserved.*

We now proceed to show the layerwise sensitivity bounds. First, we note that we follow a very similar proof idea shown in Barbero et al. (2024): in fact, the layerwise bounds and the last-token embeddings are exactly the ones shown in Theorem 5.1 and Theorem B.5 of Barbero et al. (2024) under mild modification. We include the results nonetheless for the sake of completeness, which aids the derivation of mean-token readouts.

### A.2 Layerwise Bounds and Path Expansion

The residual form (Attn)–(MLP) and our Lipschitz bounds imply, for $j \ge i$,

$$\begin{aligned}
\Big\|\frac{\partial \mathbf{v}_j^{(\ell+1)}}{\partial \mathbf{v}_i^{(\ell)}}\Big\| &= \Big\|\frac{\partial}{\partial \mathbf{v}_i^{(\ell)}}\Big[\psi^{(\ell)}\big(\mathrm{norm}_2^{(\ell)}(\mathbf{z}_j^{(\ell)})\big) + \mathbf{z}_j^{(\ell)}\Big]\Big\| && (4)\\
&\le \Big(\tfrac{\sigma_\psi^{(\ell)}}{\beta_2^{(\ell)}} + 1\Big)\Big\|\frac{\partial \mathbf{z}_j^{(\ell)}}{\partial \mathbf{v}_i^{(\ell)}}\Big\| && (5)\\
&= \Big(\tfrac{\sigma_\psi^{(\ell)}}{\beta_2^{(\ell)}} + 1\Big)\Big\|\frac{\partial}{\partial \mathbf{v}_i^{(\ell)}}\Big[\sum_{m\le j}\alpha_{j,m}^{(\ell)}\,\mathrm{norm}_1^{(\ell)}\big(\mathbf{v}_m^{(\ell)}\big) + \mathbf{v}_j^{(\ell)}\Big]\Big\| && (6)\\
&\le \Big(\tfrac{\sigma_\psi^{(\ell)}}{\beta_2^{(\ell)}} + 1\Big)\Big(\tfrac{\alpha_{j,i}^{(\ell)}}{\beta_1^{(\ell)}} + \delta_{j,i}\Big), && (*)
\end{aligned}$$

and $\frac{\partial \mathbf{v}_j^{(\ell+1)}}{\partial \mathbf{v}_i^{(\ell)}} = 0$ if $j < i$.

Define the *residual-augmented attention* $\bar\alpha^{(\ell)} \in \mathbb{R}^{n\times n}$ by

$$\bar\alpha_{j,i}^{(\ell)} := \frac{\alpha_{j,i}^{(\ell)}}{\beta_1^{(\ell)}} + \delta_{j,i} \quad \text{for } i \le j, \qquad \bar\alpha_{j,i}^{(\ell)} = 0 \text{ for } i > j,$$

and let the row-sum be $r_\ell := \sum_{i\le j}\bar\alpha_{j,i}^{(\ell)} = 1 + \frac{1}{\beta_1^{(\ell)}}$ (because $\sum_{i\le j}\alpha_{j,i}^{(\ell)} = 1$). Normalize

$$\mathbf{M}^{(\ell)} := \frac{1}{r_\ell}\,\bar\alpha^{(\ell)} \quad\Longrightarrow\quad \mathbf{M}^{(\ell)}\mathbf{1} = \mathbf{1},$$

i.e., $\mathbf{M}^{(\ell)}$ is lower-triangular and row-stochastic.

**Lemma A.2** (Path–sum equals matrix entry)**.** *Let $\bar\alpha^{(\ell)}$ and $\mathbf{M}^{(\ell)} = \frac{1}{r_\ell}\bar\alpha^{(\ell)}$ be as above, and $\mathbf{A} := \mathbf{M}^{(L-1)}\cdots\mathbf{M}^{(0)}$. For any $i \le j$,*

$$\sum_{k_1\ge i}\sum_{k_2\ge k_1}\cdots\sum_{k_L\ge k_{L-1}}\bar\alpha_{j,k_L}^{(L-1)}\prod_{\ell=2}^{L-1}\bar\alpha_{k_\ell,k_{\ell-1}}^{(\ell-1)}\,\bar\alpha_{k_1,i}^{(0)} \;=\; \Big(\prod_{\ell=0}^{L-1} r_\ell\Big)\,\mathbf{A}_{j,i}.$$

*Proof.* Write the left-hand side as the $(j, i)$ entry of the matrix product $\bar{\alpha}^{(L-1)} \cdots \bar{\alpha}^{(0)}$: by definition of matrix multiplication for lower-triangular matrices,

$$\left(\bar{\alpha}^{(L-1)} \cdots \bar{\alpha}^{(0)}\right)_{j,i} = \sum_{k_L \geq \cdots \geq k_1} \bar{\alpha}^{(L-1)}_{j,k_L} \cdots \bar{\alpha}^{(0)}_{k_1,i},$$

where the index constraints $k_\ell \geq k_{\ell-1}$ (and $j \geq k_L$, $k_1 \geq i$) are exactly those enforced by lower-triangularity. Now factor each layer's row-sum: $\bar{\alpha}^{(\ell)} = r_\ell \mathbf{M}^{(\ell)}$. Hence

$$\bar{\alpha}^{(L-1)} \cdots \bar{\alpha}^{(0)} = \Big(\prod_{\ell=0}^{L-1} r_\ell\Big)\, \mathbf{M}^{(L-1)} \cdots \mathbf{M}^{(0)} = \Big(\prod_{\ell=0}^{L-1} r_\ell\Big)\, \mathbf{A}.$$

Taking the $(j, i)$ entry gives the claim. The nested sums are precisely the $(j, i)$ entry of the product $\bar{\alpha}^{(L-1)} \cdots \bar{\alpha}^{(0)}$ because lower-triangularity imposes the constraints $j \geq k_L \geq \cdots \geq k_1 \geq i$.

□

### A.3 Oversquashing Bounds for Last-token and Mean-token Readouts

We are now ready to present the main results. Note that the following results for the last token directly correlate to Theorem B.5, Barbero et al. (2024).

**Theorem A.3** (Last-token vs mean-token sensitivity)**.** *Under the assumptions above, let*

$$C \;:=\; \frac{1}{\beta_3} \prod_{\ell=0}^{L-1} \Big(\frac{\sigma_\psi^{(\ell)}}{\beta_2^{(\ell)}} + 1\Big), \qquad K_L \;:=\; C \prod_{\ell=0}^{L-1} r_\ell, \qquad \mathbf{A} \;:=\; \mathbf{M}^{(L-1)} \cdots \mathbf{M}^{(0)}.$$

*Then for every input position $i \in [n]$:*

$$\textit{(a) Last token:} \quad \Big\| \frac{\partial \mathbf{y}_n}{\partial \mathbf{v}_i^{(0)}} \Big\| \;\leq\; K_L\, \mathbf{A}_{n,i}.$$

$$\textit{(b) Mean pooling:} \quad \Big\| \frac{\partial \bar{\mathbf{y}}}{\partial \mathbf{v}_i^{(0)}} \Big\| \;\leq\; \frac{K_L}{n} \sum_{j=1}^{n} \mathbf{A}_{j,i}.$$

*Proof.* By the chain rule across layers, $\partial \mathbf{v}_n^{(L)} / \partial \mathbf{v}_i^{(0)}$ equals a sum over causal paths $i \to k_1 \to \cdots \to k_L \to n$ with one Jacobian factor per layer. Following the proof strategies in Theorem B.5, Barbero et al. (2024) and using $(*)$ at each layer and $\|\partial \mathbf{y}_n / \partial \mathbf{v}_n^{(L)}\| \leq 1/\beta_3$ gives

$$\Big\| \frac{\partial \mathbf{y}_n}{\partial \mathbf{v}_i^{(0)}} \Big\| \;\leq\; \frac{1}{\beta_3} \prod_{\ell=0}^{L-1} \Big(\frac{\sigma_\psi^{(\ell)}}{\beta_2^{(\ell)}} + 1\Big) \sum_{k_1 \geq i} \cdots \sum_{k_L \geq k_{L-1}} \bar{\alpha}^{(L-1)}_{n,k_L} \prod_{\ell=2}^{L-1} \bar{\alpha}^{(\ell-1)}_{k_\ell, k_{\ell-1}}\, \bar{\alpha}^{(0)}_{k_1,i}.$$

Lemma A.2 converts the multi-sum to $\big(\prod_\ell r_\ell\big) \mathbf{A}_{n,i}$, yielding (a). For (b), by linearity:

$$\frac{\partial \bar{\mathbf{y}}}{\partial \mathbf{v}_i^{(0)}} = \frac{1}{n} \sum_{j=1}^{n} \frac{\partial \mathbf{y}_j}{\partial \mathbf{v}_i^{(0)}} \quad \Rightarrow \quad \Big\| \frac{\partial \bar{\mathbf{y}}}{\partial \mathbf{v}_i^{(0)}} \Big\| \leq \frac{K_L}{n} \sum_{j=1}^{n} \mathbf{A}_{j,i}.$$

Here, the term $\sum_{j=1}^{n} \mathbf{A}_{j,i}$ is the sum of the $i$-th column of $\mathbf{A}$.

□

**Interpretation and Discussion** Our sensitivity bounds reveal both a depth-dependent growth factor and a structural transport term. The growth factor

$$K_L \;=\; \underbrace{\frac{1}{\beta_3}}_{\text{final norm}} \underbrace{\prod_{\ell=0}^{L-1} \Big(\frac{\sigma_\psi^{(\ell)}}{\beta_2^{(\ell)}} + 1\Big)}_{\text{MLP+residual per layer}} \underbrace{\prod_{\ell=0}^{L-1} \Big(1 + \frac{1}{\beta_1^{(\ell)}}\Big)}_{\text{attn+residual row sums}}$$

typically grows (often exponentially) with depth $L$, scaling the magnitude of the bound. Orthogonal to this, the structural term $\mathbf{A} = \mathbf{M}^{(L-1)} \cdots \mathbf{M}^{(0)}$ governs how signal moves and attenuates across layers, and the choice of readout changes *which* part of $\mathbf{A}$ the bound depends on.

With *last-token readout*, the sensitivity is controlled by a **single entry** $\mathbf{A}_{n,i}$, i.e., the influence that reaches one fixed "sink" $n$ from source $i$. Random-walk intuition suggests that a single entry like $\mathbf{A}_{n,i}$ can shrink rapidly with depth. In the homogeneous left-drifting regime of Barbero et al. (2024), Proposition B.8, one even has $\mathbf{A} = \mathbf{M}^L \to \mathbf{1}e_1^\top$, hence $\mathbf{A}_{n,i} \to 0$ for $i > 1$.

In contrast, *mean-pooling* depends on the **entire column sum** $\sum_j \mathbf{A}_{j,i}$, which aggregates influence delivered from $i$ to *all* outputs. This aggregates total outgoing mass rather than betting on a single path to a single sink, and thus is structurally more robust: it does not suffer the same guaranteed decay as an individual matrix entry. Consequently, while $K_L$ sets the overall scale of the bound, the readout choice determines whether the structural term induces decay (last-token) or preserves signal (mean-pooling), explaining why mean-pooling provably mitigates over-squashing of early tokens.

| Model | Configuration Details |
|---|---|
| `mistral-instruct` | Mistral-7B-Instruct-v0.3<br>TP plan applied from layer 1 to 7<br>Uses output from third to last layer |
| `gemma-2-9b` | Gemma-2-9b<br>TP plan applied from layer 1 to 6<br>Uses output from second to last layer |
| `qwen2-instruct` | Qwen2.5-1.5B-Instruct<br>TP plan applied from layer 1 to 7<br>Uses output from second to last layer |

Table 9: Summary of TP-based and HTP model configurations.

# B Experiment Details

## B.1 Retrieval Tasks

The detailed description of the statistics of the BEIR evaluation dataset can be found in Table 18 and also in Thakur et al. (2021). We show the characteristics of the datasets of the real-world subsets of the LongEmbed (Zhu et al., 2024) datasets in Table 17.

### B.1.1 LLM Architecture Setup and HTP Setup

Table 9 shows the LLM configuration for TP-based and HTP models.

### B.1.2 Instruction For Retrievals

Table 10 shows the instructions used in retrieval tasks for both BEIR (Thakur et al., 2021) and LongEmbed (Zhu et al., 2024).

## B.2 General Embedding Tasks

To evaluate the generalization capabilities of HTP and baseline methods, we benchmark on a diverse set of 30 public datasets from the Massive Text Embedding Benchmark (MTEB) (Muennighoff et al., 2023). We report the average performance across four categories of tasks: Classification, Reranking, Clustering, and Semantic Textual Similarity (STS). We show their dataset statistics in Table 19.

The specific datasets used in our evaluation are as follows:

- **Classification** (11 datasets): We use accuracy as the metric for AmazonCounterfactual, AmazonReview, Banking77, Emotion, Imdb, MassiveIntent, MassiveScenario, MTOPDomain, MTOPIntent, ToxicConversations, and TweetSentiment.
- **Reranking** (3 datasets): We use Mean Average Precision (MAP) for AskUbuntuDupQuestions, MindSmallReranking, and StackOverflowDupQuestions.
- **Clustering** (11 datasets): We use the V-measure score for ArxivClusteringP2P, ArxivClusteringS2S, BiorxivClusteringP2P, BiorxivClusteringS2S, MedrxivClusteringP2P, MedrxivClusteringS2S, RedditClustering, RedditClusteringP2P, StackExchangeClustering, StackExchangeClusteringP2P and TwentyNewsgroupsClustering.
- **STS** (5 datasets): We report Spearman correlation for the standard STS12 through STS16 benchmarks.

### B.2.1 Experiment Setup

Table 11 shows the instructions (prompts) used in acquiring the general embeddings.

### B.2.2 General Embedding Result

We show the general embedding performance for Classification tasks in Table 12, Reranking tasks in Table 13, Clustering tasks in Table 14, and STS tasks in Table 15.

## B.3 Ablations

Table 16 shows the detailed NDCG@10 for ablation study of various $K$ sizes. Note that when we only insert one token at the end of the last sentence, HTP is equivalant to TP Mean. Table 6 reports the results for different exiting layers while keeping the starting layer fixed at the first layer. Exiting at a much earlier layer (e.g., layer 3) leads to worse performance compared with our current configuration (layer 7). Exiting at much later layers (or not using early exit at all) also results in performance degradation, except on the NFCorpus dataset. In addition, we study different start HTP layer, as in Table 7. The overall degradation, however, remains relatively small, indicating that HTP is robust to the choice of exiting layer. Our findings are also consistent with (Fu et al., 2024), which shows that token prepending tends to work better in earlier layers of decoder-based LLMs.

| Task Name | Instruction Template |
|---|---|
| ArguAna | Given a claim, retrieve documents that support or refute the claim |
| FiQA2018 | Given a financial question, retrieve user replies that best answer the question |
| HotpotQA | Given a multi-hop question, retrieve documents that can help answer the question |
| NFCorpus | Given a question, retrieve relevant documents that answer the question |
| SCIDOCS | Given a scientific paper title, retrieve paper abstracts that are cited by the given paper |
| SciFact | Given a scientific claim, retrieve documents that support or refute the claim |
| TREC-COVID | Given a query on COVID-19, retrieve documents that answer the query |
| NarrativeQA | Retrieve the relevant document |
| QMSum | Retrieve the relevant document |
| 2WikiMultihopQA | Retrieve the relevant document |
| SummScreenFD | Retrieve the relevant document |

Table 10: Instructions used for evaluation on the BEIR benchmark and LongEmbed benchmark.

## C Examples

**Example on ArguAna**

<s> Given a claim, retrieve documents that support or refute the claim Text: <B-PST><B-PST><B-PST><B-PST><B-PST><B-PST><B-PST><B-PST><B-PST><PST> Ending poverty through entrepreneurialism Introducing finance provides communities with access to startup capital. <PST> Access to financial capital is vital in several respects for initiating capitalism. <PST> Firstly, access to capital enables entrepreneurialism. <PST> The poor have business ideas that would benefit both themselves and their community they just require access to capital to invest in such ideas. <PST> The Initiative 'Lend with Care' is providing access to capital to empower entrepreneurs. <PST> [1] . <PST> Secondly, the cumulative effect of small-scale savings and borrowing, enabled through microfinance enables individuals, families and communities, to enter markets - of land and property. <PST> Being able to buy property and land can enable personal security, dignity, and increasing returns. <PST> [1] See further readings: Lend with Care, 2013. </s>

| Task Name | Instruction Template |
|---|---|
| AmazonCounterfactualClassification | Classify a given Amazon customer review text as either counterfactual or non-counterfactual |
| AmazonPolarityClassification | Classify Amazon reviews into positive or negative sentiment |
| AmazonReviewsClassification | Classify the given Amazon review into its appropriate rating category |
| Banking77Classification | Given an online banking query, find the corresponding intents |
| EmotionClassification | Classify the emotion expressed in the given Twitter message into one of the six emotions: anger, fear, joy, love, sadness, and surprise |
| ImdbClassification | Classify the sentiment expressed in the given movie review text from the IMDB dataset |
| MassiveIntentClassification | Given a user utterance as query, find the user intents |
| MassiveScenarioClassification | Given a user utterance as query, find the user scenarios |
| MTOPDomainClassification | Classify the intent domain of the given utterance in task-oriented conversation |
| MTOPIntentClassification | Classify the intent of the given utterance in task-oriented conversation |
| ToxicConversationsClassification | Classify the given comments as either toxic or not toxic |
| TweetSentimentExtractionClassification | Classify the sentiment of a given tweet as either positive, negative, or neutral |
| ArxivClusteringP2P | Identify the main and secondary category of Arxiv papers based on the titles and abstracts |
| ArxivClusteringS2S | Identify the main and secondary category of Arxiv papers based on the titles |
| BiorxivClusteringP2P | Identify the main category of Biorxiv papers based on the titles and abstracts |
| BiorxivClusteringS2S | Identify the main category of Biorxiv papers based on the titles |
| MedrxivClusteringP2P | Identify the main category of Medrxiv papers based on the titles and abstracts |
| MedrxivClusteringS2S | Identify the main category of Medrxiv papers based on the titles |
| RedditClustering | Identify the topic of the given Reddit posts based on the titles |
| RedditClusteringP2P | Identify the topic of the Reddit posts based on the titles and posts |
| StackExchangeClustering | Identify the topic or theme of StackExchange posts based on the titles |
| StackExchangeClusteringP2P | Identify the topic or theme of the StackExchange posts based on the given paragraphs |
| TwentyNewsgroupsClustering | Identify the topic or theme of the given news articles |
| AskUbuntuDupQuestions | Retrieve duplicate questions from AskUbuntu forum |
| MindSmallReranking | Retrieve relevant news articles based on user browsing history |
| StackOverflowDupQuestions | Retrieve duplicate questions from StackOverflow forum |

Table 11: Prompts used for evaluation on the general embedding tasks.

| Task | Echo Mean | Prompt-EOL | TP w. PromptEOL | HTP (Ours) |
|---|---|---|---|---|
| AmazonCounterfactual | 70.36 | 73.88 | 74.09 | 78.64 |
| Banking77 | 77.50 | 66.46 | 66.46 | 68.88 |
| Emotion | 38.90 | 46.44 | 46.43 | 47.51 |
| IMDb | 74.17 | 80.20 | 90.20 | 78.35 |
| MassiveIntent | 70.61 | 71.71 | 67.25 | 75.20 |
| MassiveScenario | 75.42 | 72.88 | 69.78 | 76.58 |
| MTOPDomain | 88.77 | 87.19 | 87.19 | 85.39 |
| MTOPIntent | 76.28 | 82.43 | 82.43 | 78.05 |
| ToxicConversations | 65.49 | 73.13 | 73.13 | 73.14 |
| TweetSentiment | 50.04 | 48.44 | 61.21 | 49.26 |
| AmazonReview | 39.76 | 46.56 | 46.56 | 41.84 |
| **Average** | 66.12 | 68.12 | **69.52** | <u>68.44</u> |

Table 12: Performance on classification tasks (%). We **bold** the top one and <u>underline</u> the runner-up in the average row.

| Task | Echo Mean | Prompt-EOL | TP w. PromptEOL | HTP (Ours) |
|---|---|---|---|---|
| AskUbuntuDupQuestions | 56.09 | 53.65 | 53.65 | 51.05 |
| MindSmallReranking | 28.60 | 27.29 | 27.84 | 28.96 |
| StackOverflowDupQuestions | 45.23 | 40.11 | 40.21 | 42.53 |
| **Average** | **43.31** | 40.35 | 40.57 | <u>40.85</u> |

Table 13: Performance on reranking tasks (%). We **bold** the top one and <u>underline</u> the runner-up in the average row.

| Task | Echo Mean | Prompt-EOL | TP w. PromptEOL | HTP (Ours) |
|---|---|---|---|---|
| ArxivClusteringP2P | 43.18 | 30.98 | 27.46 | 48.48 |
| ArxivClusteringS2S | 37.13 | 25.01 | 24.70 | 34.50 |
| BiorxivClusteringP2P | 31.93 | 17.91 | 17.53 | 37.74 |
| BiorxivClusteringS2S | 25.28 | 14.12 | 13.86 | 26.23 |
| MedrxivClusteringP2P | 27.10 | 17.50 | 17.04 | 30.36 |
| MedrxivClusteringS2S | 23.86 | 17.93 | 17.28 | 25.65 |
| RedditClustering | 36.05 | 16.55 | 15.48 | 26.78 |
| RedditClusteringP2P | 56.10 | 34.41 | 33.92 | 59.20 |
| StackExchangeClustering | 43.11 | 30.27 | 28.88 | 43.53 |
| StackExchangeClusteringP2P | 36.50 | 26.17 | 25.64 | 35.31 |
| TwentyNewsgroupsClustering | 21.60 | 23.29 | 21.44 | 21.46 |
| **Average** | <u>34.71</u> | 23.10 | 22.11 | **35.39** |

Table 14: Average performance on clustering tasks (%). We **bold** the top one and <u>underline</u> the runner-up in the average row.

| Task | Echo Mean | Prompt-EOL | TP w. PromptEOL | HTP (Ours) |
|---|---|---|---|---|
| STS16 | 76.19 | 70.08 | 70.13 | 57.84 |
| STS15 | 69.09 | 69.25 | 69.93 | 62.58 |
| STS14 | 58.27 | 62.39 | 59.40 | 48.28 |
| STS13 | 71.91 | 74.19 | 75.80 | 54.68 |
| STS12 | 46.85 | 63.56 | 65.22 | 44.80 |
| **Average** | <u>64.46</u> | 67.89 | **68.10** | 53.64 |

Table 15: Performance on STS tasks (%). We **bold** the top one and <u>underline</u> the runner-up in the average row.

| Model | $K$ | NFCorpus | SciFact | SummFD | 2WikiMQA |
|---|---|---|---|---|---|
| TP Mean | - | 8.15 | 42.71 | 60.11 | 23.22 |
| HTP | 1 | 15.51 | 46.66 | 54.21 | 28.44 |
| | 2 | 16.25 | 45.00 | 56.49 | 28.91 |
| | 4 | 13.16 | 45.11 | 56.25 | 28.79 |
| | 8 | 12.24 | 44.93 | 56.73 | 28.71 |
| | 12 | 12.31 | 44.88 | 56.57 | 29.55 |
| | 20 | 12.27 | 44.85 | 56.61 | 29.31 |
| | 32 | 12.27 | 44.85 | 56.23 | 29.24 |
| | 64 | 11.80 | 41.36 | 56.74 | 28.90 |

Table 16: Performance comparison of the TP Mean and HTP methods across different datasets and varying $K$ values.

| Dataset | Domain | # Queries | # Docs | Avg. Query Words | Avg. Doc Words |
|---|---|---|---|---|---|
| NarrativeQA | Literature, Film | 10,449 | 355 | 9 | 50,474 |
| QMSum | Meeting | 1,527 | 197 | 71 | 10,058 |
| 2WikiMultihopQA | Wikipedia | 300 | 300 | 12 | 6,132 |
| SummScreenFD | ScreenWriting | 336 | 336 | 102 | 5,582 |

Table 17: Datasets statistics for LongEmbed. We report train pairs, dev/test query counts, corpus size, average number of relevant documents per query (Avg Doc/Query), and average query/document lengths in words, taken directly from the benchmark.

| Dataset | Domain | #Train Pairs | #Dev Query | #Test Query | #Docs | Avg Doc/Query | Avg Query Words | Avg Doc Words |
|---|---|---|---|---|---|---|---|---|
| ArguAna | Misc. | – | – | 1,406 | 8,674 | 1.0 | 192.98 | 166.80 |
| FiQA-2018 | Finance | 14,166 | 500 | 648 | 57,638 | 2.6 | 10.77 | 132.32 |
| HotpotQA | Wikipedia | 170,000 | 5,447 | 7,405 | 5,233,329 | 2.0 | 17.61 | 46.30 |
| NFCorpus | Bio-Medical | 110,575 | 324 | 323 | 3,633 | 38.2 | 3.30 | 232.26 |
| SCIDOCS | Scientific | – | – | 1,000 | 25,657 | 4.9 | 9.38 | 176.19 |
| SciFact | Scientific | 920 | – | 300 | 5,183 | 1.1 | 12.37 | 213.63 |
| TREC-COVID | Bio-Medical | – | – | 50 | 171,332 | 493.5 | 10.60 | 160.77 |

Table 18: Statistics of the BEIR datasets used in our analysis. We report train pairs, dev/test query counts, corpus size, average number of relevant documents per query (Avg Doc/Query), and average query/document lengths in words, taken directly from the BEIR benchmark.

| Dataset | Task | Type | # Samples | Avg. Length (chars) |
|---|---|---|---|---|
| AmazonCounterfactual | Classification | s2s | 5,023 | 106.1 |
| AmazonReviews | Classification | s2s | 1,260,000 | 160.4 |
| Banking77 | Classification | s2s | 13,083 | 54.2 |
| Emotion | Classification | s2s | 20,000 | 96.6 |
| IMDb | Classification | p2p | 50,000 | 1293.8 |
| MassiveIntent | Classification | s2s | 16,521 | 34.6 |
| MassiveScenario | Classification | s2s | 16,521 | 34.6 |
| MTOPDomain | Classification | s2s | 22,288 | 36.8 |
| MTOPIntent | Classification | s2s | 22,288 | 36.8 |
| ToxicConversations | Classification | s2s | 100,000 | 296.6 |
| TweetSentiment | Classification | s2s | 31,015 | 67.8 |
| AskUbuntuDupQuestions | Reranking | s2s | 2,255 | 52.5 |
| MindSmallReranking | Reranking | s2s | 339,498 | 70.9 |
| StackOverflowDupQuestions | Reranking | s2s | 29,952 | 49.8 |
| ArxivClusteringP2P | Clustering | p2p | 732,723 | 1009.9 |
| ArxivClusteringS2S | Clustering | s2s | 732,723 | 74.0 |
| BiorxivClusteringP2P | Clustering | p2p | 75,000 | 1666.2 |
| BiorxivClusteringS2S | Clustering | s2s | 75,000 | 101.6 |
| MedrxivClusteringP2P | Clustering | p2p | 37,500 | 1981.2 |
| MedrxivClusteringS2S | Clustering | s2s | 37,500 | 114.7 |
| RedditClustering | Clustering | s2s | 840,928 | 64.7 |
| RedditClusteringP2P | Clustering | p2p | 459,399 | 727.7 |
| StackExchangeClustering | Clustering | s2s | 790,910 | 57.0 |
| StackExchangeClusteringP2P | Clustering | p2p | 75,000 | 1090.7 |
| TwentyNewsgroupsClustering | Clustering | s2s | 59,545 | 32.0 |
| STS12 | STS | s2s | 10,684 | 64.7 |
| STS13 | STS | s2s | 3,000 | 54.0 |
| STS14 | STS | s2s | 7,500 | 54.3 |
| STS15 | STS | s2s | 6,000 | 57.7 |
| STS16 | STS | s2s | 2,372 | 65.3 |

Table 19: Statistics of the 30 MTEB datasets used in our evaluation. For classification tasks, #Instances is the number of training examples, and the average length is the mean training sequence length in words. For reranking, clustering and STS tasks, #Instances and average length are taken from the MTEB metadata and refer to the evaluation items (documents, queries or sentence pairs).