# Linear-Time and Constant-Memory Text Embeddings Based on Recurrent Language Models

**Tobias Grantner**
Dynatrace Research
tobias.grantner@dynatrace.com

**Emanuel Sallinger**
TU Wien & University of Oxford
emanuel.sallinger@tuwien.ac.at

**Martin Flechl**
Dynatrace Research
martin.flechl@dynatrace.com

## Abstract

Transformer-based embedding models suffer from quadratic computational and linear memory complexity, limiting their utility for long sequences. We propose recurrent architectures as an efficient alternative, introducing a vertically chunked inference strategy that enables fast embedding generation with memory usage that becomes constant in the input length once it exceeds the vertical chunk size. By fine-tuning Mamba2 models, we demonstrate their viability as general-purpose text embedders, achieving competitive performance across a range of benchmarks while maintaining a substantially smaller memory footprint compared to transformer-based counterparts. We empirically validate the applicability of our inference strategy to Mamba2, RWKV, and xLSTM models, confirming consistent runtime-memory trade-offs across architectures and establishing recurrent models as a compelling alternative to transformers for efficient embedding generation.

## 1 Introduction

Text embeddings are central to modern natural language processing, enabling tasks such as information retrieval, clustering, and semantic similarity estimation. While transformer-based embedding models represent the state of the art, their quadratic computational and linear memory complexity in the input length restricts their use for long documents and in resource-constrained environments.

Recent advances in recurrent architectures such as Mamba, RWKV, and xLSTM (Gu and Dao, 2024; Dao and Gu, 2024; Peng et al., 2023, 2025; Beck et al., 2024) achieve competitive language modeling performance while offering linear computational scaling and constant memory usage. Despite extensive work on the efficiency of these architectures for text generation, their use for general-purpose embedding generation remains largely unexplored, with existing work limited to small task-specific models (Zhang et al., 2024; Liang et al., 2025; Pan, 2025; Cao et al., 2025).

We investigate recurrent architectures as efficient alternatives to transformer-based text embedding models by applying the $\text{E5}_{\text{mistral-7b}}$ training procedure (Wang et al., 2024) to pretrained Mamba2-based language models (Dao and Gu, 2024). Our results on the English Massive Text Embedding Benchmark (MTEB), Multilingual MTEB, and the `LongEmbed` benchmark (Muennighoff et al., 2023; Enevoldsen et al., 2025; Zhu et al., 2024) demonstrate competitive performance against strong transformer baselines, validating the viability of recurrent models as general-purpose text embedders.

Beyond architectural adaptation, we leverage the duality of Mamba2 to derive matrix formulations for chunked inference, building on Dao and Gu (2024), that enable parallelization across chunks while maintaining linear computational complexity. We then propose a vertically chunked inference strategy that processes inputs in fixed-size blocks across model layers, balancing parallelization with recurrent processing to keep memory usage constant in the input length. This strategy applies broadly to recurrent architectures with a linear-time recurrence and a complementary parallelizable formulation for intra-chunk processing. We validate it empirically on Mamba2, RWKV, and xLSTM models. Our results show that parallelization benefits saturate at small vertical chunk sizes across all three architectures, yielding favorable memory efficiency over transformers without considerable runtime overhead, particularly for long documents.

We release our fine-tuned Mamba2-based model checkpoints together with our inference implementation, to support reproducibility and encourage further research.[1]

[1] 🤗 `collections/dynatrace-oss/embed-mamba2`



*Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 41459–41481

## 2 Background

### 2.1 Embedding Model Training

The transformer architecture has become the de facto standard for text embedding models. Early approaches adapted masked language models like BERT (Devlin et al., 2019) through multi-stage weak supervision. Following $\text{E5}_{\text{mistral-7b}}$ (Wang et al., 2024), decoder-only models have shifted the paradigm by fine-tuning autoregressive language models on diverse synthetic and annotated data. Despite further advancements driven by improved synthetic data generation, stronger foundation models, and extended training procedures (Zhang et al., 2025; Choi et al., 2024; Vera et al., 2025), $\text{E5}_{\text{mistral-7b}}$ remains competitive among general-purpose open-weight embedding models.

### 2.2 Efficient Embedding Inference

Efficiency gains in transformer inference typically rely on attention optimizations or quantization. FlashAttention (Dao et al., 2022; Dao, 2024; Shah et al., 2024) uses tiling to avoid the materialization of the quadratic attention matrix, reducing the memory footprint of the attention computation from quadratic to linear in the sequence length while improving runtime through increased hardware utilization. Sliding-window attention (Child et al., 2019; Jiang et al., 2023) restricts the context to a fixed local window, achieving linear computational complexity at the cost of expressiveness. Quantization techniques reduce the memory footprint of model weights and activations by representing them in lower-precision formats. While static embedding models (Tulkens and van Dongen, 2024) offer the highest efficiency and constant memory usage, they generally lag behind transformer-based models in embedding quality.

### 2.3 Recurrent Language Models

Recurrent neural networks (RNNs) scale linearly with the input length by maintaining a constant-sized latent state. Traditional RNNs lack the parallelizability required for large-scale training, but modern variants like Mamba2 (Dao and Gu, 2024), RWKV (Peng et al., 2023, 2025), and xLSTM (Beck et al., 2024) introduce structured recurrence mechanisms that admit parallel computation of state updates. Although their text generation capabilities have been extensively studied, their potential as general-purpose text embedders has received comparatively little attention.

## 3 Recurrent Embedding Models

### 3.1 Training

To evaluate the effectiveness of embedding models based on recurrent language model architectures, we adopt the contrastive training recipe established by $\text{E5}_{\text{mistral-7b}}$ (Wang et al., 2024). Given a pre-trained model $\mathcal{M}$, we adapt it for embedding generation by removing the language modeling head to create $\mathcal{M}'$. For each input sequence $\boldsymbol{s}$ consisting of tokens $(t_1, \ldots, t_n) \in \mathcal{T}^n$, we append the model's end-of-sequence (EOS) token $t^{\texttt{EOS}}$. The output of the final model layer at this terminal position $\boldsymbol{e}^{\texttt{EOS}} \in \mathbb{R}^d$ serves as a summary of the preceding context, which we use as the representation $\mathcal{E}(\boldsymbol{s})$ of the input text:

$$\begin{aligned} \boldsymbol{s} &= (t_1, t_2, \ldots, t_n) \\ \boldsymbol{s} \oplus t^{\texttt{EOS}} &= (t_1, t_2, \ldots, t_n, t^{\texttt{EOS}}) \\ \mathcal{M}'(\boldsymbol{s} \oplus t^{\texttt{EOS}}) &= (\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n, \boldsymbol{e}^{\texttt{EOS}}) \\ \mathcal{E}(\boldsymbol{s}) &= \boldsymbol{e}^{\texttt{EOS}} \end{aligned} \tag{1}$$

The resulting model $\mathcal{E} : \mathcal{T}^* \to \mathbb{R}^d$ maps a variable-length text sequence to a $d$-dimensional real-valued output of fixed size. We fine-tune it for embedding generation using the InfoNCE loss, which encourages the model to distinguish a relevant positive pair from a set of negative examples (van den Oord et al., 2019):

$$\mathcal{L} = -\log \frac{e^{\cos(\boldsymbol{e}_q, \boldsymbol{e}_p)/\tau}}{e^{\cos(\boldsymbol{e}_q, \boldsymbol{e}_p)/\tau} + \sum_{i=1}^{N} e^{\cos(\boldsymbol{e}_q, \boldsymbol{e}_{n_i})/\tau}} \tag{2}$$

Here, $\boldsymbol{e}_q$ and $\boldsymbol{e}_p$ represent the embeddings of the query and the positive sample, respectively, $\cos(\cdot, \cdot)$ is the cosine similarity function, and $\tau$ is a temperature hyperparameter. The training data consists of query-positive pairs, along with zero, one, or multiple hard negative examples. The $N$ in-batch negatives $\boldsymbol{e}_{n_i}$ comprise both the hard negatives provided with the query, and all positives and hard negatives associated with the other queries in the batch.

Following Wang et al. (2024), we incorporate instruction tuning to enable the model to adapt to varied downstream tasks without task-specific training. We use the following template for all queries:

`Instruction:␣{prompt}\nQuery:␣`

where `{prompt}` is replaced with the task-specific instruction, and `\n` is a newline character. By

prepending this instruction only to the query, not to the documents, we ensure the system remains efficient for large-scale retrieval, since the document corpus only needs to be embedded once, regardless of the specific task instruction.

### 3.2 Recurrent Embedding Inference

While the inference efficiency of recurrent language models during autoregressive generation is well-documented, their behavior during the initial encoding of input tokens, which constitutes the inference process for embedding generation, remains underexplored.

We investigate recurrent embedding inference using Mamba2 as a representative architecture because its dual nature allows it to be formulated as both a linear recurrence and a structured matrix transformation. We propose strategies for optimizing parallelization levels and memory scheduling. Conceptually, our scheduling strategy requires two properties: i) a linear-time recurrent structure over the input sequence and ii) a parallelizable dual formulation that supports intra-chunk processing in addition to the recurrent mode. Because these properties are shared by a broader family of recurrent and SSM-style architectures, including RWKV and xLSTM (see Appendix B), our strategy is widely applicable. We empirically validate this in Section 4.2 on all three architectures, while focusing on Mamba2 for the formal derivation. For clarity, our discussion omits discretization and focuses on single-dimensional input and output. The concepts generalize independently to multiple dimensions, and we refer to Dao and Gu (2024) for proofs and the extension to discretized, multi-dimensional input.

#### 3.2.1 The Duality of Mamba2

Mamba2 (Dao and Gu, 2024) is a structured state space model (SSM) that maps an input sequence $\boldsymbol{x} \in \mathbb{R}^T$ to an output sequence $\boldsymbol{y} \in \mathbb{R}^T$ of length $T$. The output $\boldsymbol{y}_t$ at each time step $t \in \{1, \dots, T\}$ depends on the current input $\boldsymbol{x}_t$ and a latent state vector $\boldsymbol{h}_{t-1} \in \mathbb{R}^N$ with a state expansion factor $N$. The state is updated iteratively, and the output is generated as follows:

$$\boldsymbol{h}_t = \boldsymbol{A}_t \boldsymbol{h}_{t-1} + \boldsymbol{B}_t \boldsymbol{x}_t \tag{3a}$$

$$\boldsymbol{y}_t = \boldsymbol{C}_t^\top \boldsymbol{h}_t \tag{3b}$$

The time-variant matrices $\boldsymbol{A} \in \mathbb{R}^{T \times N \times N}$, $\boldsymbol{B} \in \mathbb{R}^{T \times N}$ and $\boldsymbol{C} \in \mathbb{R}^{T \times N}$ are functions of the input, typically computed via a combination of learned projections and non-linear transformations. This recurrent formulation enables inference with $\mathcal{O}(T)$ computational complexity and $\mathcal{O}(1)$ memory usage with respect to the sequence length. However, its sequential computation of latent state updates prevents it from fully exploiting the parallelization capabilities of modern hardware accelerators.

To bridge this gap, structured SSMs constrain the transition matrix $\boldsymbol{A}_t$ to allow for efficient computation of state updates. For Mamba2, $\boldsymbol{A}_t = a_t \boldsymbol{I}$, where $a_t$ is a scalar and $\boldsymbol{I}$ corresponds to the identity matrix. This structure allows the recurrence in Equation 3a to be unrolled. Assuming an initial state $\boldsymbol{h}_0 = \boldsymbol{0}$, the state at any time $t$ (and consequently the output $\boldsymbol{y}_t$) can be expressed as a direct function of all preceding inputs:

$$\boldsymbol{A}^{\times}_{i:j} := \begin{cases} \prod_{k=j+1}^{i} a_k \, \boldsymbol{I}, & i > j \\ \boldsymbol{I}, & i = j \\ \boldsymbol{0}, & i < j \end{cases} \tag{4a}$$

$$\begin{aligned} \boldsymbol{y}_t &= \sum_{s=1}^{t} \boldsymbol{C}_t^\top \boldsymbol{A}_t \cdots \boldsymbol{A}_{s+1} \boldsymbol{B}_s \boldsymbol{x}_s \\ &= \sum_{s=1}^{t} \boldsymbol{C}_t^\top \boldsymbol{A}^{\times}_{t:s} \boldsymbol{B}_s \boldsymbol{x}_s \end{aligned} \tag{4b}$$

Here, $\boldsymbol{A}^{\times}_{i:j}$ represents the cumulative transition from step $j$ to $i$, taking advantage of the structure $\boldsymbol{A}_t = a_t \boldsymbol{I}$. The entire sequence transformation can therefore be viewed as a single linear operator $\boldsymbol{M}$:

$$\boldsymbol{M}_{i,j} := \boldsymbol{C}_i^\top \boldsymbol{A}^{\times}_{i:j} \boldsymbol{B}_j \tag{5a}$$

$$\boldsymbol{y} = \boldsymbol{M} \boldsymbol{x} \tag{5b}$$

As a consequence of the recurrence in Equation 3a, $\boldsymbol{M}$ is a lower-triangular matrix, which is encoded by the value of $\boldsymbol{A}^{\times}_{i:j}$ for $i < j$. We can further vectorize the computation of $\boldsymbol{M}$ by defining a causal kernel matrix $\boldsymbol{L} \in \mathbb{R}^{T \times T}$ with:

$$\boldsymbol{L}_{i,j} := \begin{cases} \prod_{k=j+1}^{i} a_k, & i > j \\ 1, & i = j \\ 0, & i < j \end{cases} \tag{6a}$$

$$\boldsymbol{M} = \boldsymbol{L} \circ (\boldsymbol{C} \boldsymbol{B}^\top) \tag{6b}$$

This matrix formulation is key to the training efficiency of Mamba2, as it leverages parallelizable, hardware-accelerated matrix multiplications. However, instantiating the full matrix $\boldsymbol{M}$ incurs $\mathcal{O}(T^2)$ complexity in both computation and memory. The

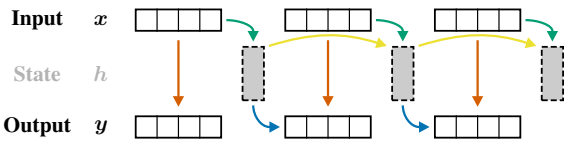



Figure 1: Chunked inference with parallel computation of chunks and recurrent state updates between chunks.

*duality* of the model lies in its ability to switch between these two views, producing the same output: the fully parallelizable quadratic matrix formulation, and the memory-efficient linear recurrent view from Equation 3.

### 3.2.2 Chunked Inference

The duality of Mamba2 enables the combination of the memory efficiency of recurrence with the parallelism of matrix multiplication. This is achieved by partitioning the input sequence into $K = T/Q$ chunks, each of size $Q$, assuming $Q$ divides $T$ for simplicity. Local chunk results are computed in parallel using the matrix formulation from Equation 5, while the global latent state is iteratively updated between chunks as illustrated in Figure 1. Under this strategy, the dominant intra-chunk work is quadratic in the chunk size $Q$, yielding a total cost of $\mathcal{O}(K \cdot Q^2) = \mathcal{O}(T \cdot Q)$ over a sequence of length $T$. By leveraging the semi-separable structure of $\boldsymbol{M}$, i.e., the property that its off-diagonal blocks admit low-rank factorizations through the shared transition matrices $\boldsymbol{A}$, we can additionally parallelize most of the computation across chunks through a block-decomposition matrix multiplication algorithm. A detailed derivation from first principles is provided in Appendix A, while the key steps are outlined in this section.

We define chunked views of our previously introduced matrices and vectors for chunk indices $c, c' \in \{1, \dots, K\}$, within-chunk indices $i, j \in \{1, \dots, Q\}$ and latent state indices $k, l \in \{1, \dots, N\}$:

$$\widehat{\boldsymbol{L}}^{(c,c')}_{i,j} := \boldsymbol{L}_{(c-1)Q+i,(c'-1)Q+j} \tag{7a}$$

$$\widehat{\boldsymbol{A}}^{(c)}_{i,k,l} := \boldsymbol{A}_{(c-1)Q+i,k,l} \tag{7b}$$

$$\widehat{\boldsymbol{A}}^{(c)\times}_{i:j} := \boldsymbol{A}^{\times}_{(c-1)Q+i:(c-1)Q+j} \tag{7c}$$

$$\widehat{\boldsymbol{B}}^{(c)}_{i,k} := \boldsymbol{B}_{(c-1)Q+i,k} \tag{7d}$$

$$\widehat{\boldsymbol{C}}^{(c)}_{i,k} := \boldsymbol{C}_{(c-1)Q+i,k} \tag{7e}$$

$$\widehat{\boldsymbol{x}}^{(c)}_{i} := \boldsymbol{x}_{(c-1)Q+i} \tag{7f}$$

For any block $\widehat{\boldsymbol{L}}^{(c,c')}$, we use $\widehat{\boldsymbol{L}}^{(c,c')}_{i,\cdot} \in \mathbb{R}^{Q}$ to denote row $i$ and $\widehat{\boldsymbol{L}}^{(c,c')}_{\cdot,j} \in \mathbb{R}^{Q}$ to denote column $j$. To distinguish chunk-boundary states from token-level recurrent states, we define the boundary state after chunk $c$ by $\boldsymbol{b}^{(c)} := \boldsymbol{h}_{cQ}$. The computation is split into three stages: i) intra-chunk computation, ii) inter-chunk state propagation, and iii) the final output adjustment.

**Intra-chunk Computation (Parallelizable)**
First, we calculate the intra-chunk contributions to the outputs and the boundary latent states for each chunk independently, assuming a zero incoming chunk-boundary state ($\boldsymbol{b}^{(c-1)} = \boldsymbol{0}$). These operations are chunk-local matrix transformations:

$$\widehat{\boldsymbol{y}}^{(c)}_{\text{intra}} = \big(\widehat{\boldsymbol{L}}^{(c,c)} \circ (\widehat{\boldsymbol{C}}^{(c)}\, \widehat{\boldsymbol{B}}^{(c)\top})\big)\widehat{\boldsymbol{x}}^{(c)} \tag{8}$$

$$\boldsymbol{b}^{(c)}_{\text{intra}} = \widehat{\boldsymbol{B}}^{(c)\top} \operatorname{diag}\big(\widehat{\boldsymbol{L}}^{(c,c)}_{Q,\cdot}\big)\widehat{\boldsymbol{x}}^{(c)} \tag{9}$$

**Inter-chunk State Propagation (Sequential)**
To maintain global context, we must account for the state left by the previous chunk. We recurrently calculate the final chunk-boundary state $\boldsymbol{b}^{(c)}$ by combining the final boundary state of the previous chunk with the current intra-chunk contribution. Given an initial state $\boldsymbol{b}^{(0)} = \boldsymbol{0}$,

$$\boldsymbol{b}^{(c)} = \boldsymbol{A}^{\times}_{(cQ):((c-1)Q)} \boldsymbol{b}^{(c-1)} + \boldsymbol{b}^{(c)}_{\text{intra}} \tag{10}$$

**Final Output Adjustment (Parallelizable)**
Finally, we calculate the correction needed for the output of each chunk based on the updated final state from the previous chunk, add it to our intra-chunk result, and assemble the final output. We define the base case $\widehat{\boldsymbol{y}}^{(1)}_{\text{inter}} = \boldsymbol{0}$, since the first chunk has no preceding context. For $c > 1$,

$$\widehat{\boldsymbol{y}}^{(c)}_{\text{inter}} = \operatorname{diag}\big(\widehat{\boldsymbol{L}}^{(c,c-1)}_{\cdot,Q}\big)\, \widehat{\boldsymbol{C}}^{(c)}\, \boldsymbol{b}^{(c-1)} \tag{11}$$

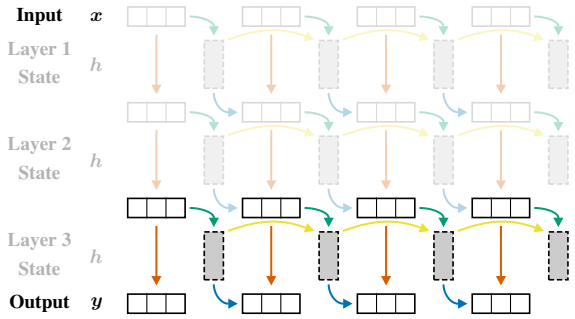
$$\widehat{\boldsymbol{y}}^{(c)} = \widehat{\boldsymbol{y}}^{(c)}_{\text{intra}} + \widehat{\boldsymbol{y}}^{(c)}_{\text{inter}} \tag{12a}$$

$$\boldsymbol{y}_{(c-1)Q+i} = \widehat{\boldsymbol{y}}^{(c)}_{i} \tag{12b}$$

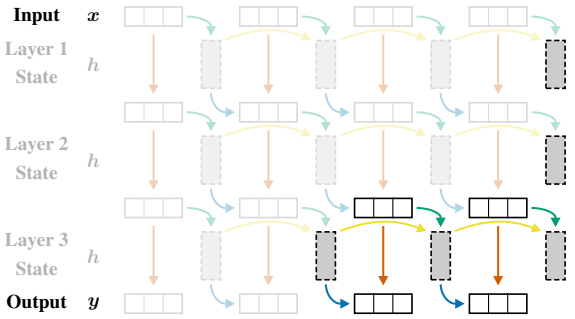
The remaining sequential computations for the state propagation in Equation 10 operate only on $K$ chunk-boundary states and consist of matrix-vector multiplications with low computational overhead. The majority of computations can be performed in parallel. Consequently, the overall computational cost scales as $\mathcal{O}(K \cdot Q^2) = \mathcal{O}(T \cdot Q)$, with the activation memory following the same scaling.

### 3.2.3 Cross-Layer Inference Strategies

Modern embedding models consist of $L$ layers stacked sequentially, where the output of layer $l$




(a) Fully horizontal inference: parallelization spans the full sequence length, but all intermediate activations for the currently processed layer must be retained in memory.




(b) Fully vertical inference: parallelization is limited to a fixed vertical chunk size, but only the activations of the current chunk and one recurrent state per layer need to be stored.

Figure 2: Final forward pass through the last model layer using fully horizontal (left) and vertically chunked (right) inference. Highlighted regions indicate activations retained in memory. A visualization of the full forward pass through all layers is provided in Appendix C.

serves as input to layer $l+1$. The standard execution pattern is *horizontal inference*: the model processes the entire sequence of length $T$ through one layer at a time. Each Mamba2 layer can internally leverage the chunked parallelization described in Section 3.2.2 as illustrated in Figure 2a. While this horizontal inference strategy benefits from intra-layer parallelization, it requires storing full-sequence intermediate activations for the currently processed layer. Consequently, the resulting memory consumption of $\mathcal{O}(K \cdot Q^2) = \mathcal{O}(T \cdot Q)$ scales linearly with the input length $T$.

To overcome this limitation, we propose chunked *vertical inference*, which trades sequence-level parallelization for depth-wise recurrence. Specifically, we partition the input sequence into $K' = T/V$ vertical chunks of size $V$, where $V$ is a multiple of the intra-layer chunk size $Q$, implying $V \geq Q$. We process a single vertical chunk through all $L$ layers before advancing to the next chunk, as illustrated in Figure 2b. For each layer, the recurrent state is preserved across vertical chunks, analogous to the within-layer state recurrence across time steps described in Section 3.2.2. This creates a *cross-layer recurrence* over vertical chunks in addition to the within-layer recurrence over horizontal chunks. The memory footprint of the model is now decoupled from the total sequence length $T$ for $T > V$, since we only need to store the activations of the current vertical chunk of length $V$ and the $L$ recurrent states, one for each layer. For input lengths $T \leq V$, we can fall back to standard horizontal inference, which avoids the need to store latent states across layers and thus eliminates any memory overhead from vertical chunking for short sequences. A detailed step-by-step visualization of both strategies is provided in Appendix C.

The resulting memory consumption is $\mathcal{O}(L + Q^2 \cdot \frac{V}{Q}) = \mathcal{O}(L + QV)$, where the first term accounts for the $L$ per-layer recurrent states and the second for the activations of the current vertical chunk, remaining constant in $T$ for $T > V$. While reducing the number of tokens processed simultaneously from $T$ to $V$ theoretically limits parallelization, modern hardware often reaches its compute-bound ceiling at comparatively small sequence lengths. By choosing a vertical chunk size $V$ large enough to saturate hardware throughput but small enough to fit within memory constraints, we achieve the speed of horizontal inference with the scalability of recurrence. We validate the efficiency of our approach empirically in Section 4.2, and provide practical parameter selection guidelines in Appendix E.1.

## 4 Experiments

### 4.1 Model Comparison

We demonstrate the effectiveness of recurrent embedding models by fine-tuning pretrained language models based on both Mamba2 and transformer architectures using the training procedure outlined in Section 3.1. The resulting embedding models are evaluated on the original English Massive Text Embedding Benchmark `MTEB(eng, v1)`, its updated version `MTEB(eng, v2)`, the Massive Multilingual Text Embedding Benchmark `MTEB(Multilingual, v2)`, and the long-context retrieval benchmark `LongEmbed` (Muennighoff et al., 2023; Enevoldsen et al., 2025; Zhu et al., 2024).

| Model | MTEB(Mult., v2) | | MTEB(eng, v2) | | LongEmbed |
|---|---|---|---|---|---|
| Mean over → | Task | Type | Task | Type | Task |
| Qwen2 1.5B | 56.2 | 48.7 | 65.7 | 62.3 | 41.0 |
| Mamba2 1.3B | 55.2 | 47.9 | 64.3 | 60.9 | 40.8 |
| Mistral 7B `v0.1` | 58.6 | 50.5 | 67.3 | 63.3 | 44.7 |
| Codestral Mamba2 7B | 59.4 | 51.9 | 65.2 | 61.8 | 44.5 |

Table 1: Evaluation of our fine-tuned transformer-based and recurrent embedding models on `MTEB(Multilingual, v2)`, `MTEB(eng, v2)`, and `LongEmbed`. Results are aggregated as the mean over tasks and the mean over task types where applicable. Detailed per-task-type results for all benchmarks are provided in Appendix D.

| Model | MTEB(eng, v1) | |
|---|---|---|
| Mean over → | Task | Type |
| Results from Wang et al. (2024) | | |
| $E5_{\text{mistral-7b (v0.1)}}$ | 66.5 | 64.2 |
| Results from Springer et al. (2025) | | |
| Qwen2 1.5B | 63.3 | 61.3 |
| Mistral 7B `v0.1` | 65.5 | 63.2 |
| Our results | | |
| $E5_{\text{mistral-7b}}$ training recipe with data from Springer et al. (2025) | | |
| Qwen2 1.5B | 63.6 | 61.8 |
| Mistral 7B `v0.1` | 65.7 | 63.5 |

Table 2: Evaluation of our fine-tuned transformer-based models on `MTEB(eng, v1)`, alongside scores reported by Wang et al. (2024) and Springer et al. (2025). Results are aggregated as the mean over tasks and the mean over task types where applicable. Detailed per-task-type results are provided in Appendix D.

The `MTEB(eng, v2)` benchmark includes 41 English datasets covering classification, clustering, pair classification, reranking, retrieval, semantic textual similarity (STS), and summarization tasks. The `MTEB(Multilingual, v2)` benchmark extends this to a multilingual setting encompassing more than 250 languages, and `LongEmbed` evaluates retrieval performance on long documents with sequences of up to 32,768 tokens in our setup.

We compare our fine-tuned transformer and recurrent models (Table 1) after validating our training procedure (Table 2). First, we reproduce the results of $E5_{\text{mistral-7b}}$ (Wang et al., 2024) and the reproduction by Springer et al. (2025) by fine-tuning pretrained checkpoints of Mistral 7B `v0.1`[2] (Jiang et al., 2023), the base model underlying $E5_{\text{mistral-7b}}$, and Qwen2 1.5B (Yang et al., 2024). We train the models for one epoch on a combination of public datasets and the synthetic data generated using Llama 3.1 70B (Grattafiori et al., 2024) and published by Springer et al. (2025), which replicates the synthetic data generation pipeline of $E5_{\text{mistral-7b}}$. Further details on the training data and hyperparameters can be found in Appendix F.

The results are shown in Table 2. Our fine-tuned Qwen2 1.5B improves upon the task mean reported by Springer et al. (2025) by 0.3 percentage points, while our model based on Mistral 7B `v0.1` improves by 0.2 percentage points, reducing the gap to the original $E5_{\text{mistral-7b}}$ scores.

For comparison, we fine-tune pretrained Mamba2-based language models of comparable size following the same training procedure, data, and hyperparameters where applicable. Concretely, we use the Mamba2 1.3B model published by Dao and Gu (2024) and the code generation model Codestral Mamba2 7B[3]. We found that Mamba2 1.3B benefits from a considerably higher learning rate compared to the other models. Details on the Mamba2 base models and the learning rate selection for Mamba2 1.3B are provided in Appendix F.

In Table 1, we contrast our fine-tuned transformer and recurrent models under largely identical fine-tuning conditions across three benchmarks. The performance differences of recurrent and transformer-based models vary between benchmarks. On `MTEB(Multilingual, v2)`, Codestral Mamba2 7B outperforms Mistral 7B `v0.1`, achieving the highest mean task score among our fine-tuned models, while Mamba2 1.3B trails

[2]We additionally fine-tune the more recent version `v0.3` of Mistral 7B, which is used in the inference evaluation in Section 4.2. The differences between the versions and detailed per-task results for both versions are provided in Appendix F and Appendix D, respectively.

[3]`https://mistral.ai/news/codestral-mamba`

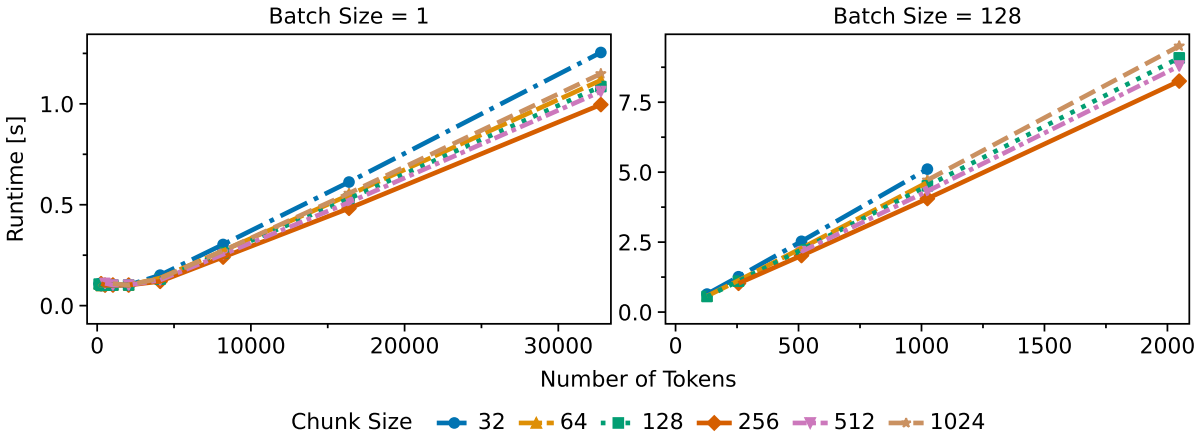



Figure 3: Runtime for Codestral Mamba2 7B using fully horizontal chunked inference with varying chunk sizes for batch sizes 1 and 128.

Qwen2 1.5B by a comparable margin. For `MTEB(eng, v2)`, the Mamba2-based models lag behind their transformer-based counterparts by 1–2 percentage points. On `LongEmbed`, both model families achieve comparable performance, with differences remaining within 0.2 percentage points for models of similar size. These results demonstrate that recurrent models are viable general-purpose text embedders, achieving competitive quality while offering favorable inference efficiency. The remaining quality gap on English tasks may be partially attributable to differences in pretraining scale and data composition between the recurrent and transformer base models, as Mamba2 1.3B was pretrained on significantly less data and Codestral Mamba2 7B was specifically trained for code generation. We anticipate that further improvements are possible with more extensively pretrained base models.

### 4.2 Inference Evaluation

To investigate the efficiency of recurrent embedding inference strategies, we measure runtime and memory consumption during embedding generation for varying input lengths and batch sizes on a single NVIDIA H100 with 80 GB of memory. We evaluate Codestral Mamba2 7B using both the fully horizontal and the vertically chunked inference strategies outlined in Section 3.2, with varying chunk sizes to explore the effect of different parallelization levels. For the model comparison, we additionally evaluate RWKV7 7.2B and xLSTM 7B using their respective optimal vertically chunked inference configurations. For comparison against transformer-based models, we measure the runtime and memory consumption of Mistral 7B `v0.1` and Mistral 7B `v0.3`. We repeat our experiments three times and report mean results to account for system load variability. Corresponding minimum and maximum values are reported in Appendix E.

Figure 3 visualizes the effect of different chunk sizes on the runtime of Codestral Mamba2 7B for single-sequence embedding generation and for batches of 128 sequences. We observe that a chunk size of $Q = 256$ provides the best performance across input lengths and batch sizes. Smaller chunk sizes reduce the parallelization potential, while larger chunk sizes increase the overhead of quadratic intra-chunk computations. We use this chunk size as the basis for further experiments.

With $Q$ fixed at 256, Figure 4 shows that the single-sequence runtime approaches that of fully horizontal inference as the vertical chunk size ($V$) increases, reaching full convergence at $V = 4096$ in our setup. This indicates that parallelization is fully saturated at 4096 tokens, making the performance impact of cross-layer recurrence negligible. For a batch size of 32, the runtime remains close to that of fully horizontal inference across all vertical chunk sizes, showing only a minor increase for a vertical chunk size of $V = 256$. Parallelization saturates at smaller vertical chunk sizes when processing batches.

A cross-architecture comparison of runtime and memory consumption against the transformer-based Mistral 7B `v0.1` and `v0.3` models is shown in Figure 5. For single-sequence embedding generation, Codestral Mamba2 7B outperforms both transformer-based models in terms of runtime and memory usage for input lengths beyond 4096 to-

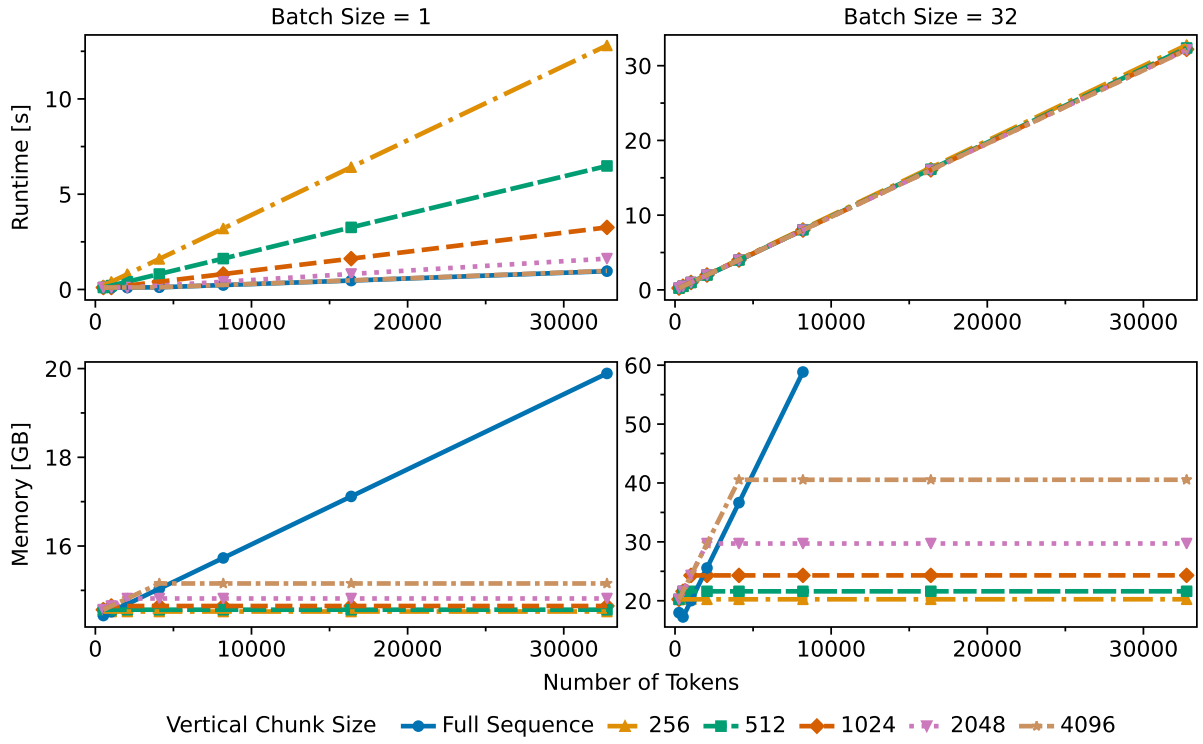



Figure 4: Runtime and memory usage for Codestral Mamba2 7B using vertically chunked inference with a fixed intra-layer chunk size of 256 and varying vertical chunk sizes for batch sizes 1 and 32.

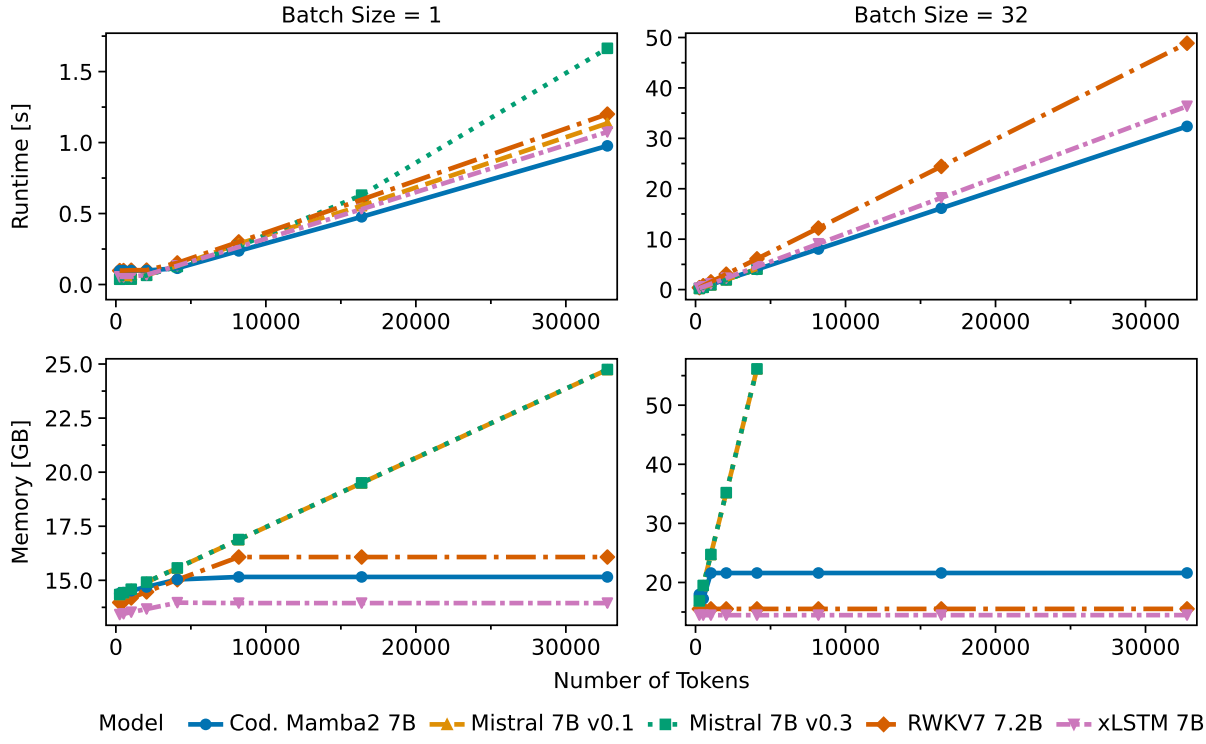



Figure 5: Runtime and memory usage for Codestral Mamba2 7B, RWKV7 7.2B, and xLSTM 7B compared to Mistral 7B `v0.1` and `v0.3`. Codestral Mamba2 7B uses vertically chunked inference with an intra-layer chunk size of 256 and vertical chunk sizes of 4096 (batch size 1) and 512 (batch size 32). Latent states are stored across layers only for input lengths exceeding the vertical chunk size. RWKV7 7.2B and xLSTM 7B use the same strategy with chunk sizes 32 and 64 and vertical chunk sizes 64 and 128, respectively, for batch size 32. Both use a vertical chunk size of 4096 for batch size 1. Transformer-based models use FlashAttention-2.

kens. The advantage over Mistral 7B `v0.3` grows with sequence length due to the quadratic complexity of full attention, while Mistral 7B `v0.1` does not exhibit this quadratic growth since it uses a sliding window of 4096 tokens. The runtime of all models is comparable for batches of 32 sequences, although both Mistral 7B versions were only evaluated for input lengths up to 4096 tokens due to memory requirements exceeding the available accelerator memory. All three recurrent architectures share the same memory-scaling pattern: consumption grows linearly up to the vertical chunk size and then remains practically constant for longer sequences, in contrast to the linear growth of the transformer-based models. As a result, all recurrent models exhibit substantially lower memory requirements than transformers for long sequences.

## 5 Conclusions

We have demonstrated that recurrent language models are a viable and efficient foundation for text embedding generation. By applying standard contrastive fine-tuning, our Mamba2-based models achieve performance competitive with similarly sized transformers across the `MTEB(eng, v2)`, `MTEB(Multilingual, v2)`, and `LongEmbed` benchmarks. Minor performance differences exist on individual benchmarks, with transformers slightly leading on English tasks and recurrent models showing advantages on multilingual tasks, while overall results remain consistently close.

Our primary technical contribution is a vertically chunked inference strategy applicable to recurrent architectures with a linear-time recurrence and a dual, parallelizable formulation. By introducing a cross-layer scheduling approach that processes data in fixed-size vertical blocks, we avoid storing full-sequence activations between layers, reducing memory requirements to a function of model depth rather than sequence length. Empirical validation on Mamba2, RWKV, and xLSTM models confirms consistent behavior across all three architectures: activation memory becomes effectively constant beyond the vertical chunk size, and parallelism saturates at relatively small vertical chunk sizes. In practice, the vertical chunk size can serve as a mechanism to enable inference under strict memory constraints while retaining close to fully parallel runtime, rather than as a hyperparameter requiring careful optimization.

Our comparisons reveal that while transformers and recurrent models perform similarly for short context lengths, the recurrent approach offers favorable runtime-memory trade-offs at moderate scale and becomes particularly advantageous for applications involving long contexts or batch processing, where the memory requirements of transformers become prohibitive. These findings establish recurrent architectures as a scalable and resource-efficient alternative for embedding generation. We anticipate that more extensively pretrained recurrent base models are a promising direction toward closing the remaining quality gap on English tasks, paving the way for a new class of long-context, resource-efficient embedding models.

## Limitations

**Recurrent Architectures.** Our embedding quality evaluation focuses on Mamba2 as a representative recurrent architecture. While we extend the empirical inference efficiency comparison to RWKV7 7.2B and xLSTM 7B to validate the generalizability of vertically chunked inference, a thorough investigation of embedding quality for architectures beyond Mamba2 is left for future work.

**Base Models.** The Mamba2 1.3B model, which serves as a foundation for our experiments, was pretrained on 700 billion tokens, an order of magnitude fewer than the Qwen2 1.5B transformer model, which was trained on 7 trillion tokens. For Codestral Mamba2 7B, information on the exact pretraining data and duration is not publicly available, but as it is an instruction-tuned code generation model, its pretraining likely differs considerably from the transformer-based models evaluated in this work. This discrepancy may influence the embedding quality comparison and could explain the remaining small quality gaps. Fine-tuning more extensively pretrained recurrent base models under comparable pretraining conditions is a promising direction for future work.

**Evaluation Scope.** While our experiments demonstrate the effectiveness of Mamba2-based embedding models across the `MTEB(eng, v2)`, `MTEB(Multilingual, v2)`, and `LongEmbed` benchmarks, covering a wide range of downstream tasks, domains, and languages, other potential applications remain unevaluated. In particular, we do not account for the distribution of input sequence lengths in the benchmarks nor do we systemati-

cally vary input lengths for controlled evaluation. Further investigation is needed to assess the performance of recurrent embedding models across a wider range of input lengths and specialized domains.

**Broader Impact.** Our work contributes to making text embedding generation more efficient, which can facilitate access to NLP technology in resource-constrained settings. However, embedding models can inherit biases present in their pretraining data. Users should be aware that embedding-based similarity judgments may reflect such biases and should supplement model outputs with expert verification where necessary.

## Acknowledgments

We thank the anonymous reviewers for their thoughtful and constructive feedback, which helped us strengthen both the experimental analysis and the presentation of this paper.

Emanuel Sallinger's work on this paper was funded by the Vienna Science and Technology Fund (WWTF), Grant IDs `10.47379/VRG18013`, `10.47379/ICT25032`, `10.47379/NXT22018`, `10.47379/ICT2201`, `10.47379/DCDH001`, and by the Austrian Science Fund (FWF) `10.55776/COE12`.

## References

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. 2024. xLSTM: Extended long short-term memory. In *Advances in Neural Information Processing Systems*, volume 37, pages 107547–107603. Curran Associates, Inc.

Weili Cao, Jianyou Wang, Youze Zheng, Longtian Bao, Qirui Zheng, Taylor Berg-Kirkpatrick, Ramamohan Paturi, and Leon Bergen. 2025. Single-pass document scanning for question answering. In *Proceedings of the Second Conference on Language Modeling*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *Preprint*, arXiv:1904.10509.

Chanyeol Choi, Junseong Kim, Seolhwa Lee, Jihoon Kwon, Sangmo Gu, Yejin Kim, Minkyung Cho, and Jy yong Sohn. 2024. Linq-Embed-Mistral technical report. *Preprint*, arXiv:2412.03223.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pages 35549–35562.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359. Curran Associates, Inc.

Tri Dao and Albert Gu. 2024. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 10041–10071. PMLR.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115. Curran Associates, Inc.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Diganta Misra, Shreeya Dhakal, Jonathan Rystrø m, Roman Solomatin, Ömer Çağatan, and 63 others. 2025. MMTEB: Massive multilingual text embedding benchmark. In *International Conference on Learning Representations*, volume 2025, pages 101715–101771.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. In *Proceedings of the First Conference on Language Modeling*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

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.

Li Liang, Huan Wang, and Kai Wang. 2025. Cognitive-inspired xLSTM for multi-agent information retrieval. *Scientific Reports*, 15(1):36121.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia. Association for Computational Linguistics.

Xinghan Pan. 2025. Exploring RWKV for sentence embeddings: Layer-wise analysis and baseline comparison for semantic similarity. *Preprint*, arXiv:2502.14620.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Gv, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, and 13 others. 2023. RWKV: Reinventing RNNs for the transformer era. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14048–14077, Singapore. Association for Computational Linguistics.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. 2025. RWKV-7 "Goose" with expressive dynamic state evolution. In *Second Conference on Language Modeling*.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. 2024. FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems*, volume 37, pages 68658–68685. Curran Associates, Inc.

Jacob Mitchell Springer, Vaibhav Adlakha, Siva Reddy, Aditi Raghunathan, and Marius Mosbach. 2025. Understanding the influence of synthetic data for text embedders. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22551–22567, Vienna, Austria. Association for Computational Linguistics.

Stephan Tulkens and Thomas van Dongen. 2024. Model2Vec: Fast state-of-the-art static embeddings. Zenodo.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2019. Representation learning with contrastive predictive coding. *Preprint*, arXiv:1807.03748.

Henrique Schechter Vera, Sahil Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, Daniel Cer, Alice Lisak, Min Choi, Lucas Gonzalez, Omar Sanseviero, Glenn Cameron, Ian Ballantyne, Kat Black, Kaifeng Chen, and 70 others. 2025. EmbeddingGemma: Powerful and lightweight text representations. *Preprint*, arXiv:2509.20354.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

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.

Hanqi Zhang, Chong Chen, Lang Mei, Qi Liu, and Jiaxin Mao. 2024. Mamba retriever: Utilizing mamba for effective and efficient dense retrieval. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, pages 4268–4272, New York, NY, USA. Association for Computing Machinery.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *Preprint*, arXiv:2506.05176.

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 *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 802–816, Miami, Florida, USA. Association for Computational Linguistics.

## A Chunked Inference Derivation

The calculation of $\widehat{\boldsymbol{y}}^{(c)}_{\text{intra}}$ in Equation 8 corresponds to the diagonal block of the full SSM matrix $\boldsymbol{M}$ in the block decomposition of Dao and Gu (2024, Section 6), restricted to within-chunk interactions. Starting from Equation 5a restricted to chunk $c$, we rewrite the expression in chunked notation and factor it via Equation 6:

$$\widehat{\boldsymbol{y}}^{(c)}_{\text{intra}} = \begin{bmatrix} \boldsymbol{C}^{\top}_{(c-1)Q+1}\boldsymbol{A}^{\times}_{((c-1)Q+1):((c-1)Q+1)}\boldsymbol{B}_{(c-1)Q+1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \boldsymbol{C}^{\top}_{cQ}\boldsymbol{A}^{\times}_{(cQ):((c-1)Q+1)}\boldsymbol{B}_{(c-1)Q+1} & \cdots & \boldsymbol{C}^{\top}_{cQ}\boldsymbol{A}^{\times}_{(cQ):(cQ)}\boldsymbol{B}_{cQ} \end{bmatrix} \widehat{\boldsymbol{x}}^{(c)} \tag{13a}$$

$$= \begin{bmatrix} \widehat{\boldsymbol{C}}^{(c)\top}_{1}\widehat{\boldsymbol{A}}^{(c)\times}_{1:1}\widehat{\boldsymbol{B}}^{(c)}_{1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \widehat{\boldsymbol{C}}^{(c)\top}_{Q}\widehat{\boldsymbol{A}}^{(c)\times}_{Q:1}\widehat{\boldsymbol{B}}^{(c)}_{1} & \cdots & \widehat{\boldsymbol{C}}^{(c)\top}_{Q}\widehat{\boldsymbol{A}}^{(c)\times}_{Q:Q}\widehat{\boldsymbol{B}}^{(c)}_{Q} \end{bmatrix} \widehat{\boldsymbol{x}}^{(c)} \tag{13b}$$

$$= \big(\widehat{\boldsymbol{L}}^{(c,c)} \circ (\widehat{\boldsymbol{C}}^{(c)}\,\widehat{\boldsymbol{B}}^{(c)\top})\big)\widehat{\boldsymbol{x}}^{(c)} \tag{13c}$$

The intra-chunk state $\boldsymbol{b}^{(c)}_{\text{intra}}$ in Equation 9 computes the latent state at the end of chunk $c$ from inputs within that chunk alone, disregarding the state from preceding chunks. This corresponds to the right ($\boldsymbol{B}$) block factor of the low-rank off-diagonal blocks in the block decomposition of Dao and Gu (2024, Section 6), accumulating input contributions weighted by the row vector $\widehat{\boldsymbol{L}}^{(c,c)}_{Q,\cdot}$:

$$\boldsymbol{b}^{(c)}_{\text{intra}} = \begin{bmatrix} \boldsymbol{B}^{\top}_{(c-1)Q+1}\boldsymbol{A}^{\times}_{(cQ):((c-1)Q+1)} \\ \cdots \\ \boldsymbol{B}^{\top}_{cQ}\boldsymbol{A}^{\times}_{(cQ):(cQ)} \end{bmatrix}^{\top} \widehat{\boldsymbol{x}}^{(c)} \tag{14a}$$

$$= \underbrace{\begin{bmatrix} \widehat{\boldsymbol{B}}^{(c)\top}_{1}\widehat{\boldsymbol{A}}^{(c)\times}_{Q:1} \\ \cdots \\ \widehat{\boldsymbol{B}}^{(c)\top}_{Q}\widehat{\boldsymbol{A}}^{(c)\times}_{Q:Q} \end{bmatrix}^{\top}}_{\left(\widehat{\boldsymbol{B}}^{(c)\top}\operatorname{diag}(\widehat{\boldsymbol{L}}^{(c,c)}_{Q,\cdot})\right)^{\top}} \widehat{\boldsymbol{x}}^{(c)} \tag{14b}$$

$$= \widehat{\boldsymbol{B}}^{(c)\top}\operatorname{diag}\big(\widehat{\boldsymbol{L}}^{(c,c)}_{Q,\cdot}\big)\widehat{\boldsymbol{x}}^{(c)} \tag{14c}$$

The final chunk state $\boldsymbol{b}^{(c)}$ in Equation 10 accumulates contributions from all chunks up to and including chunk $c$. This can be expressed as a recurrence over chunk states, where the center ($\boldsymbol{A}$) block factor propagates the global state of the previous chunk $\boldsymbol{b}^{(c-1)}$ forward through the transition matrices:

$$\boldsymbol{b}^{(c)} = \sum_{k=1}^{c} \boldsymbol{A}^{\times}_{(cQ):(kQ)} \begin{bmatrix} \boldsymbol{B}^{\top}_{(k-1)Q+1}\boldsymbol{A}^{\times}_{(kQ):((k-1)Q+1)} \\ \cdots \\ \boldsymbol{B}^{\top}_{kQ}\boldsymbol{A}^{\times}_{(kQ):(kQ)} \end{bmatrix}^{\top} \widehat{\boldsymbol{x}}^{(k)} \tag{15a}$$

$$= \sum_{k=1}^{c} \boldsymbol{A}^{\times}_{(cQ):(kQ)} \underbrace{\begin{bmatrix} \widehat{\boldsymbol{B}}^{(k)\top}_{1}\widehat{\boldsymbol{A}}^{(k)\times}_{Q:1} \\ \cdots \\ \widehat{\boldsymbol{B}}^{(k)\top}_{Q}\widehat{\boldsymbol{A}}^{(k)\times}_{Q:Q} \end{bmatrix}^{\top} \widehat{\boldsymbol{x}}^{(k)}}_{\boldsymbol{b}^{(k)}_{\text{intra}}} \tag{15b}$$

$$= \sum_{k=1}^{c} \boldsymbol{A}^{\times}_{(cQ):(kQ)}\boldsymbol{b}^{(k)}_{\text{intra}} \tag{15c}$$

$$= \underbrace{\boldsymbol{A}^{\times}_{(cQ):(cQ)}}_{\boldsymbol{I}}\boldsymbol{b}^{(c)}_{\text{intra}} + \sum_{k=1}^{c-1} \boldsymbol{A}^{\times}_{(cQ):(kQ)}\boldsymbol{b}^{(k)}_{\text{intra}} \tag{15d}$$

$$= \boldsymbol{b}^{(c)}_{\text{intra}} + \boldsymbol{A}^{\times}_{(cQ):((c-1)Q)} \underbrace{\sum_{k=1}^{c-1} \boldsymbol{A}^{\times}_{((c-1)Q):(kQ)} \boldsymbol{b}^{(k)}_{\text{intra}}}_{\boldsymbol{b}^{(c-1)}} \tag{15e}$$

$$= \boldsymbol{b}^{(c)}_{\text{intra}} + \boldsymbol{A}^{\times}_{(cQ):((c-1)Q)} \boldsymbol{b}^{(c-1)} \tag{15f}$$

The inter-chunk output correction $\widehat{\boldsymbol{y}}^{(c)}_{\text{inter}}$ in Equation 11 captures the contribution of all preceding chunks to the output of chunk $c$. We start from the off-diagonal blocks of $\boldsymbol{M}$ (Equation 5b). These factor into left ($\boldsymbol{C}$), center ($\boldsymbol{A}$), and right ($\boldsymbol{B}$) block factors in the block decomposition of Dao and Gu (2024, Section 6). The derivation then collapses the right factor and input into the accumulated global state $\boldsymbol{b}^{(c-1)}$, leaving the left factor applied to $\boldsymbol{b}^{(c-1)}$:

$$\widehat{\boldsymbol{y}}^{(c)}_{\text{inter}} = \sum_{k=1}^{c-1} \begin{bmatrix} \boldsymbol{C}^{\top}_{(c-1)Q+1} \boldsymbol{A}^{\times}_{((c-1)Q+1):((c-1)Q)} \\ \cdots \\ \boldsymbol{C}^{\top}_{cQ} \boldsymbol{A}^{\times}_{(cQ):((c-1)Q)} \end{bmatrix} \boldsymbol{A}^{\times}_{((c-1)Q):(kQ)} \begin{bmatrix} \boldsymbol{B}^{\top}_{(k-1)Q+1} \boldsymbol{A}^{\times}_{(kQ):((k-1)Q+1)} \\ \cdots \\ \boldsymbol{B}^{\top}_{kQ} \boldsymbol{A}^{\times}_{(kQ):(kQ)} \end{bmatrix}^{\top} \widehat{\boldsymbol{x}}^{(k)} \tag{16a}$$

$$= \sum_{k=1}^{c-1} \begin{bmatrix} \widehat{\boldsymbol{C}}^{(c)\top}_{1} \widehat{\boldsymbol{L}}^{(c,c-1)}_{1,Q} \\ \cdots \\ \widehat{\boldsymbol{C}}^{(c)\top}_{Q} \widehat{\boldsymbol{L}}^{(c,c-1)}_{Q,Q} \end{bmatrix} \boldsymbol{A}^{\times}_{((c-1)Q):(kQ)} \begin{bmatrix} \widehat{\boldsymbol{B}}^{(k)\top}_{1} \widehat{\boldsymbol{A}}^{(k)\times}_{Q:1} \\ \cdots \\ \widehat{\boldsymbol{B}}^{(k)\top}_{Q} \widehat{\boldsymbol{A}}^{(k)\times}_{Q:Q} \end{bmatrix}^{\top} \widehat{\boldsymbol{x}}^{(k)} \tag{16b}$$

$$= \underbrace{\begin{bmatrix} \widehat{\boldsymbol{C}}^{(c)\top}_{1} \widehat{\boldsymbol{L}}^{(c,c-1)}_{1,Q} \\ \cdots \\ \widehat{\boldsymbol{C}}^{(c)\top}_{Q} \widehat{\boldsymbol{L}}^{(c,c-1)}_{Q,Q} \end{bmatrix}}_{\left(\widehat{\boldsymbol{C}}^{(c)\top} \operatorname{diag}(\widehat{\boldsymbol{L}}^{(c,c-1)}_{\cdot,Q})\right)^{\top}} \underbrace{\sum_{k=1}^{c-1} \boldsymbol{A}^{\times}_{((c-1)Q):(kQ)} \begin{bmatrix} \widehat{\boldsymbol{B}}^{(k)\top}_{1} \widehat{\boldsymbol{A}}^{(k)\times}_{Q:1} \\ \cdots \\ \widehat{\boldsymbol{B}}^{(k)\top}_{Q} \widehat{\boldsymbol{A}}^{(k)\times}_{Q:Q} \end{bmatrix}^{\top} \widehat{\boldsymbol{x}}^{(k)}}_{\boldsymbol{b}^{(c-1)}} \tag{16c}$$

$$= \left(\widehat{\boldsymbol{C}}^{(c)\top} \operatorname{diag}\left(\widehat{\boldsymbol{L}}^{(c,c-1)}_{\cdot,Q}\right)\right)^{\top} \boldsymbol{b}^{(c-1)} \tag{16d}$$

$$= \operatorname{diag}\left(\widehat{\boldsymbol{L}}^{(c,c-1)}_{\cdot,Q}\right) \widehat{\boldsymbol{C}}^{(c)} \boldsymbol{b}^{(c-1)} \tag{16e}$$

Finally, Equation 12a shows that $\widehat{\boldsymbol{y}}^{(c)}$ decomposes additively into the within-chunk contribution (diagonal block) and the cross-chunk correction (off-diagonal blocks), corresponding to the block decomposition of $\boldsymbol{M}$ in Dao and Gu (2024, Section 6). Starting from $\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x}$ (Equation 5b) and restricting to chunk $c$:

$$\widehat{\boldsymbol{y}}^{(c)} = \underbrace{\begin{bmatrix} \boldsymbol{C}^{\top}_{(c-1)Q+1} \boldsymbol{A}^{\times}_{((c-1)Q+1):((c-1)Q+1)} \boldsymbol{B}_{(c-1)Q+1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \boldsymbol{C}^{\top}_{cQ} \boldsymbol{A}^{\times}_{(cQ):((c-1)Q+1)} \boldsymbol{B}_{(c-1)Q+1} & \cdots & \boldsymbol{C}^{\top}_{cQ} \boldsymbol{A}^{\times}_{(cQ):(cQ)} \boldsymbol{B}_{cQ} \end{bmatrix} \widehat{\boldsymbol{x}}^{(c)}}_{\widehat{\boldsymbol{y}}^{(c)}_{\text{intra}}}$$
$$+ \underbrace{\sum_{k=1}^{c-1} \begin{bmatrix} \boldsymbol{C}^{\top}_{(c-1)Q+1} \boldsymbol{A}^{\times}_{((c-1)Q+1):((c-1)Q)} \\ \cdots \\ \boldsymbol{C}^{\top}_{cQ} \boldsymbol{A}^{\times}_{(cQ):((c-1)Q)} \end{bmatrix} \boldsymbol{A}^{\times}_{((c-1)Q):(kQ)} \begin{bmatrix} \boldsymbol{B}^{\top}_{(k-1)Q+1} \boldsymbol{A}^{\times}_{(kQ):((k-1)Q+1)} \\ \cdots \\ \boldsymbol{B}^{\top}_{kQ} \boldsymbol{A}^{\times}_{(kQ):(kQ)} \end{bmatrix}^{\top} \widehat{\boldsymbol{x}}^{(k)}}_{\widehat{\boldsymbol{y}}^{(c)}_{\text{inter}}} \tag{17a}$$

$$= \widehat{\boldsymbol{y}}^{(c)}_{\text{intra}} + \widehat{\boldsymbol{y}}^{(c)}_{\text{inter}} \tag{17b}$$

The diagonal block equals the within-chunk SSM output with zero initial state (Equation 8). The off-diagonal blocks factor into the $\boldsymbol{C}$ block factor applied to the accumulated global state (Equation 11).

## B Recurrent Architectures beyond Mamba2

Beyond structured state space models like Mamba2, a variety of recurrent architectures exhibit a dual formulation with both parallel and recurrent computation modes. The RWKV model family (Peng et al., 2023, 2025) replaces self-attention with a time-decayed weighted key-value (WKV) operator which computes outputs either via a parallel scan over all time steps or through a lightweight per-step recurrence. Similarly, xLSTM (Beck et al., 2024) introduces extended Long Short-Term Memory (LSTM) blocks with exponential gating and structured memory updates designed to be associative over time, enabling implementation both as parallel prefix operations across the sequence and as standard recurrent updates. These properties enable the combination of parallel intra-chunk computations with inter-chunk recurrence, analogous to the chunked inference strategy we present for Mamba2 in Section 3.2.2.

## C Cross-Layer Inference

In Section 3.2.3, we introduced two cross-layer inference strategies: fully horizontal inference and vertically chunked inference. Whereas the main text highlights the final step of these strategies to illustrate their memory footprint, this section details how each strategy processes data layer by layer. Figures 2a and 2b show the final forward pass through the last model layer, while Figures 6 and 7 illustrate the complete progression through all layers. They visualize how horizontal inference processes the entire sequence through each layer sequentially, accumulating intermediate activations for all tokens. In contrast, our vertical approach processes fixed-size chunks through all layers before advancing, maintaining only the current chunk's activations and per-layer recurrent states.

## D Evaluation Details

### D.1 Embedding Quality

For our results in Section 4, we evaluate all models on the `MTEB(eng, v2)`, `MTEB(Multilingual, v2)`, and `LongEmbed` benchmarks with `bfloat16` precision. For `MTEB(eng, v2)` and `MTEB(Multilingual, v2)`, we limit the maximum sequence length to 512 following Wang et al. (2024) and Springer et al. (2025) to ensure comparability. For `LongEmbed`, we evaluate all tasks with sequence lengths of up to 32,768 tokens, corresponding to the maximum context length of the models based on Mistral 7B. The evaluation instructions are provided in Appendix G.2.

| Task Type | Abbreviation |
|---|---|
| Bitext Mining | Bit.M. |
| Classification | Cls. |
| Clustering | Clust. |
| Instruction Reranking | I.Rera. |
| Multilabel Classification | M.Cls. |
| Pair Classification | P.Cls. |
| Reranking | Rera. |
| Retrieval | Retr. |
| STS | STS |
| Summarization | Sum. |

Table 3: MTEB task type abbreviations.

Table 3 defines the abbreviations used in Tables 4, 5, 6, and 7, presenting the detailed per-task and per-task-type results for the `MTEB(Multilingual, v2)`, `MTEB(eng, v2)`, `LongEmbed`, and `MTEB(eng, v1)` benchmarks, respectively. These results provide a comprehensive breakdown of model performance, showcasing task-type-specific strengths and weaknesses.

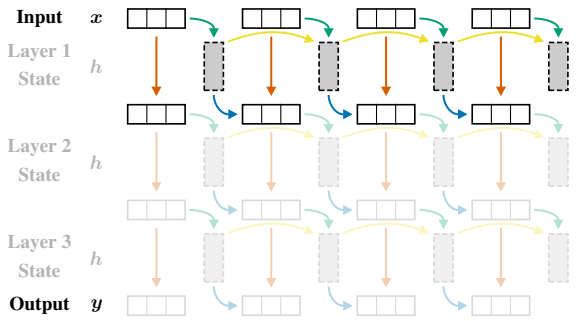
### D.2 Analysis by Task Type

The per-task-type results for Codestral Mamba2 7B and Mistral 7B `v0.1` are distinct for the English and multilingual benchmarks.

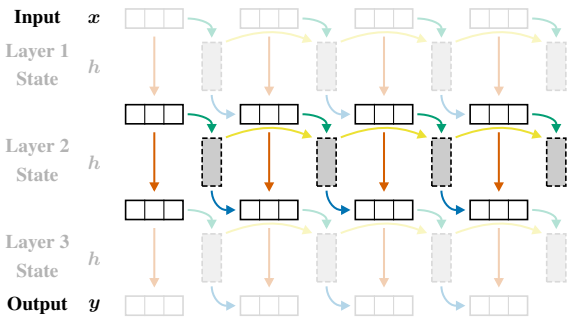
- `MTEB(Multilingual, v2)`, Table 4: Codestral Mamba2 7B leads in bitext mining by 2.0 points, reranking by 4.8 points, and retrieval by 1.9 points, closely matching or exceeding the transformer baseline across all task types.

- `MTEB(eng, v2)`, Table 5: The pattern reverses for the English benchmark. Codestral Mamba2 7B trails most notably on retrieval by 6.5 points, which accounts for the majority of the overall score difference. The remaining English task types are close, with gaps staying approximately within 1–2 points on classification, pair classification, clustering, and STS.

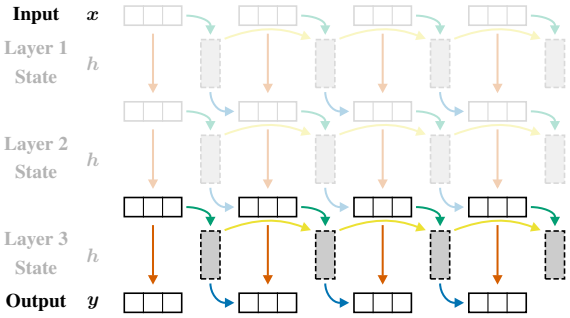
- `LongEmbed`, Table 6: Performance is similar across all tasks, with Codestral Mamba2 7B




(a) Layer 1




(b) Layer 2




(c) Layer 3

Figure 6: Step-by-step visualization of fully horizontal inference. Each diagram shows the state after processing one layer. All chunks are computed in parallel within each layer before advancing to the next, resulting in the accumulation of activations over the entire sequence.

leading on one half of the tasks and Mistral 7B `v0.1` leading on the other half.

In summary, the recurrent model outperforms or matches transformers on multilingual tasks, while English retrieval is the primary area where a notable quality gap remains. All other task types show near parity across all three benchmarks.

# E Parameter Robustness and Portability

To verify that the inference behavior of recurrent models generalizes across hardware, we repeat the key experiments on three accelerators: NVIDIA L40S, A100, and H100. The results, shown in Figures 8, 9, and 10, confirm that the findings from Section 4.2 hold consistently across platforms:

- The default chunk size configuration of $Q = 256$ remains optimal for Codestral Mamba2 7B.
- The parallelization potential saturates at a vertical chunk size of at most $V = 4096$ for single-sequence inference and at $V$ close to the horizontal chunk size $Q$ for larger batches.
- The runtime and memory advantages of recurrent models over transformers for longer sequences are preserved regardless of the hardware choice.

## E.1 Practical Parameter Selection

In our experiments, we found that the default intra-layer chunk size $Q$ works well without modification for all evaluated models. For the vertical chunk size $V$, the appropriate choice depends primarily on the batch size. For the batch sizes larger than one, parallelization nearly saturates at $V = Q$ and fully saturates at $V = 2Q$. This yields runtimes equivalent to full-sequence parallelization, all while maintaining constant activation memory beyond $V$ tokens. Single-sequence inference (batch size 1) requires larger vertical chunk sizes of $V \approx 4096$ to fully saturate the parallelization potential, with the exact saturation point depending on the target hardware.

In practice, we recommend using the model's native chunk size for $Q$ and setting $V = 2Q$ for batched inference or $V = Q$ if memory is constrained. For single-sequence inference, setting $V$ to the largest value that fits in memory yields the


(a) Vertical Chunk 1, Layer 1

(b) Vertical Chunk 1, Layer 2

(c) Vertical Chunk 1, Layer 3

(d) Vertical Chunk 2, Layer 1

(e) Vertical Chunk 2, Layer 2

(f) Vertical Chunk 2, Layer 3

Figure 7: Step-by-step visualization of vertically chunked inference. Each diagram shows one computation step, progressing through all layers for the first vertical chunk before advancing to the second vertical chunk. Only the activations of the current vertical chunk and one state per layer must be retained in memory.

best performance for the available memory budget. These settings generalize across the three accelerators (L40S, A100, H100) and three architectures (Mamba2, RWKV, xLSTM) we evaluated.

# F Training Details

## F.1 Training Data

To compare recurrent and transformer-based embedding models in a controlled setting, we follow the training procedure of $E5_{\text{mistral-7b}}$ (Wang et al., 2024) as closely as possible. We fine-tuned all models on a combination of the public datasets used to train $E5_{\text{mistral-7b}}$ and a replication of the synthetically generated training data from Springer et al. (2025). We found that the synthetic data contains almost 16,000 triples where the query, the positive, or the negative example is empty (after removing leading and trailing whitespace), as summarized in Table 8. To avoid issues during training, we filter out these triples. The instructions used during fine-tuning are provided in Appendix G.1.

## F.2 Training Configuration

We trained all models for one epoch using a batch size of 2048, a 100-step learning rate warm-up with linear decay, weight decay of 0.1, and a loss temperature $\tau = 0.02$. Following Wang et al. (2024) and

| Model | Bit.M. | Cls. | Clust. | I.Rera. | M.Cls. | P.Cls. | Rera. | Retr. | STS | Mean | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # of datasets → | 13 | 43 | 16 | 3 | 5 | 11 | 6 | 18 | 16 | Task | Type |
| Results from Wang et al. (2024) | | | | | | | | | | | |
| $E5_{\text{mistral-7b (v0.1)}}$ | 70.6 | 60.3 | 50.6 | -0.6 | 22.2 | 81.1 | 63.8 | 55.8 | 74.0 | 60.2 | 53.1 |
| Fine-tuned Transformers (ours) | | | | | | | | | | | |
| Qwen2 1.5B | 61.8 | 57.6 | 46.9 | -2.2 | 20.0 | 78.2 | 53.7 | 50.7 | 71.4 | 56.2 | 48.7 |
| Mistral 7B v0.1 | 68.6 | 60.2 | 46.9 | -4.7 | 21.2 | 79.9 | 55.3 | 54.4 | 72.4 | 58.6 | 50.5 |
| Mistral 7B v0.3 | 68.7 | 60.2 | 48.0 | -4.4 | 21.7 | 80.1 | 55.5 | 55.1 | 72.7 | 58.9 | 50.8 |
| Fine-tuned Recurrent Models (ours) | | | | | | | | | | | |
| Mamba2 1.3B | 61.1 | 55.7 | 45.2 | -5.6 | 19.5 | 77.6 | 55.4 | 50.9 | 70.9 | 55.2 | 47.9 |
| Cod. Mamba2 7B | 70.6 | 60.0 | 47.0 | -3.2 | 21.8 | 81.0 | 60.1 | 56.3 | 73.3 | 59.4 | 51.9 |

Table 4: Detailed results of the evaluated models on the `MTEB(Multilingual, v2)` benchmark. Abbreviations for task types are listed in Table 3.

| Model | Cls. | Clust. | P.Cls. | Rera. | Retr. | STS | Sum. | Mean | |
|---|---|---|---|---|---|---|---|---|---|
| # of datasets → | 8 | 8 | 3 | 2 | 10 | 9 | 1 | Task | Type |
| Results from Wang et al. (2024) | | | | | | | | | |
| $E5_{\text{mistral-7b (v0.1)}}$ | 79.9 | 51.4 | 88.4 | 49.8 | 57.6 | 84.3 | 36.6 | 68.0 | 64.0 |
| Fine-tuned Transformers (ours) | | | | | | | | | |
| Qwen2 1.5B | 78.1 | 49.7 | 85.2 | 48.4 | 54.1 | 82.2 | 38.3 | 65.7 | 62.3 |
| Mistral 7B v0.1 | 80.7 | 50.9 | 87.2 | 49.3 | 56.8 | 82.5 | 35.8 | 67.3 | 63.3 |
| Mistral 7B v0.3 | 79.9 | 52.0 | 87.1 | 49.3 | 56.4 | 82.4 | 37.1 | 67.3 | 63.5 |
| Fine-tuned Recurrent Models (ours) | | | | | | | | | |
| Mamba2 1.3B | 76.8 | 48.6 | 85.0 | 48.3 | 50.7 | 81.9 | 35.0 | 64.3 | 60.9 |
| Cod. Mamba2 7B | 79.5 | 50.5 | 86.2 | 49.3 | 50.3 | 81.7 | 35.2 | 65.2 | 61.8 |

Table 5: Detailed results of the evaluated models on the `MTEB(eng, v2)` benchmark. Abbreviations for task types are listed in Table 3.

Springer et al. (2025), we limited the maximum sequence length to 512 tokens during training. Lacking explicit sampling strategy details, we construct batches from a single source dataset, with a sampling probability proportional to dataset size. With the exception of Mamba2 1.3B, we use the learning rate of $4 \cdot 10^{-4}$ reported by Springer et al. (2025) for all models. We reduced memory consumption during training by using QLoRA (Dettmers et al., 2023) with 4-bit quantization, `bfloat16` activations, and LoRA (Hu et al., 2022) adapters ($r = 16$, $\alpha = 16$) applied to all linear model layers.

For the inference evaluation in Section 4.2, we additionally fine-tune Mistral 7B `v0.3` alongside `v0.1`. Relative to `v0.1`, version `v0.2` increased the context window from 8,192 to 32,768 tokens, adjusted positional encoding parameters, and replaced sliding-window attention with full attention. Version `v0.3` inherits these changes and additionally introduces an extended vocabulary.

### F.3 Learning Rate Optimization

Mamba2 1.3B required a considerably higher learning rate to achieve competitive performance. To determine a suitable value, we fine-tuned Mamba2 1.3B on $\frac{1}{16}$ of the total training data for one epoch. We reduced the batch size by the same factor to a value of 128, ensuring an equal number of gradient updates as in the full training setup. We evaluated a range of learning rates and report the mean score over all tasks of the `MTEB(eng, v2)` benchmark for the quantized model in Table 9. Performance peaks at a learning rate of $2.4 \cdot 10^{-3}$, which we use for the final training run.

| Model | NQA | Needle | Passkey | QMSum | SSFD | WikimQA | Mean |
|---|---|---|---|---|---|---|---|
| Results from Wang et al. (2024) | | | | | | | |
| $E5_{\text{mistral-7b (v0.1)}}$ | 37.2 | 31.5 | 30.8 | 28.6 | 75.1 | 58.7 | 43.7 |
| Fine-tuned Transformers (ours) | | | | | | | |
| Qwen2 1.5B | 27.4 | 30.5 | 34.0 | 30.8 | 75.4 | 48.2 | 41.0 |
| Mistral 7B v0.1 | 40.8 | 28.5 | 37.8 | 31.9 | 79.8 | 49.6 | 44.7 |
| Mistral 7B v0.3 | 43.2 | 27.8 | 36.5 | 32.0 | 79.8 | 49.7 | 44.8 |
| Fine-tuned Recurrent Models (ours) | | | | | | | |
| Mamba2 1.3B | 24.2 | 32.2 | 37.8 | 29.9 | 72.9 | 47.9 | 40.8 |
| Cod. Mamba2 7B | 35.7 | 31.0 | 38.8 | 30.2 | 75.0 | 56.5 | 44.5 |

Table 6: Detailed results of the evaluated models on the LongEmbed benchmark.

| Model | Cls. | Clust. | P.Cls. | Rera. | Retr. | STS | Sum. | Mean | |
|---|---|---|---|---|---|---|---|---|---|
| # of datasets → | 12 | 11 | 3 | 4 | 15 | 10 | 1 | Task | Type |
| Results from Wang et al. (2024) | | | | | | | | | |
| $E5_{\text{mistral-7b (v0.1)}}$ | 77.4 | 50.3 | 88.4 | 60.2 | 57.1 | 84.7 | 31.5 | 66.5 | 64.2 |
| Results from Springer et al. (2025) | | | | | | | | | |
| Qwen2 1.5B | 76.5 | 47.2 | 86.9 | 55.0 | 54.4 | 79.3 | 29.7 | 63.3 | 61.3 |
| Mistral 7B v0.1 | 77.9 | 49.1 | 87.4 | 57.6 | 57.1 | 82.9 | 30.1 | 65.5 | 63.2 |
| Mistral 7B v0.2 | 78.3 | 50.5 | 88.2 | 60.0 | 58.2 | 85.7 | 31.3 | 66.9 | 64.6 |
| Fine-tuned Transformers (ours) | | | | | | | | | |
| Qwen2 1.5B | 75.5 | 46.9 | 85.2 | 58.2 | 52.8 | 82.7 | 31.5 | 63.6 | 61.8 |
| Mistral 7B v0.1 | 77.6 | 49.5 | 87.2 | 59.9 | 56.3 | 82.9 | 31.0 | 65.7 | 63.5 |
| Mistral 7B v0.3 | 76.9 | 49.4 | 87.1 | 60.0 | 55.8 | 83.0 | 30.9 | 65.5 | 63.3 |
| Fine-tuned Recurrent Models (ours) | | | | | | | | | |
| Mamba2 1.3B | 73.8 | 46.2 | 85.0 | 58.0 | 49.0 | 82.4 | 30.6 | 61.9 | 60.7 |
| Cod. Mamba2 7B | 76.9 | 47.3 | 86.2 | 60.0 | 49.5 | 82.2 | 29.8 | 63.2 | 61.7 |

Table 7: Detailed results of the evaluated models on the MTEB(eng, v1) benchmark. Abbreviations for task types are listed in Table 3.

# G Instructions

## G.1 Training Instructions

Table 10 provides statistics for datasets with multiple instructions, whereas Table 11 lists the fine-tuning instructions for datasets with one or two distinct instructions.

## G.2 Evaluation Instructions

We use task-type-specific instructions (Table 12) and task-specific instructions (Table 13). For retrieval and reranking tasks, we prepend the instruction to the query only; for all other tasks, we prepend it to all sequences.

| Dataset | Total | Empty Text (Count) | Empty Text (Percentage) |
|---|---|---|---|
| Short Short | 19932 | 1995 | 10.01 % |
| STS | 99791 | 3819 | 3.83 % |
| Bitext | 89611 | 3047 | 3.40 % |
| Short Long | 153934 | 4635 | 3.01 % |
| Long Short | 108487 | 2111 | 1.95 % |
| Long Long | 19236 | 119 | 0.62 % |
| Public Datasets (11) | 1248378 | 0 | 0.00 % |
| | 1739369 | 15726 | 0.90 % |

Table 8: Distribution of triples with at least one empty text in the query, positive, or negative example.

| Learning Rate | Mean (Task) |
|---|---|
| $1.6 \cdot 10^{-3}$ | 62.89 |
| $2.2 \cdot 10^{-3}$ | 63.19 |
| $\mathbf{2.4 \cdot 10^{-3}}$ | **63.33** |
| $2.6 \cdot 10^{-3}$ | 62.77 |
| $2.8 \cdot 10^{-3}$ | 62.53 |

Table 9: Learning rate tuning results for Mamba2 1.3B, evaluated on `MTEB(eng, v2)` after fine-tuning on a subset of the full training data for one epoch.

| Dataset | Samples | Unique Instructions |
|---|---|---|
| Short-Short | 19932 | 12165 |
| Short-Long | 153934 | 96872 |
| Long-Short | 108487 | 67217 |
| Long-Long | 19236 | 11724 |

Table 10: For datasets used during fine-tuning that contain multiple distinct instructions, the number of samples and unique instructions per dataset are shown.

| Datasets | Instruction |
|---|---|
| ELI5 | Provided a user question, retrieve the highest voted answers on Reddit ELI5 forum |
| HotpotQA | Given a multi-hop question, retrieve documents that can help answer the question |
| MIRACL | Given a question, retrieve Wikipedia passages that answer the question |
| MS MARCO | Given a web search query, retrieve relevant documents that answer the query<br>Given a web search query, retrieve relevant passages that answer the query |
| MrTyDi | Given a question, retrieve Wikipedia passages that answer the question |
| NLI | Given a premise, retrieve a hypothesis that is entailed by the premise<br>Retrieve semantically similar text |
| NQ | Given a question, retrieve Wikipedia passages that answer the question |
| Quora Duplicates | Find questions that have the same meaning as the input question<br>Given a question, retrieve questions that are semantically equivalent to the given question |
| SQuAD | Retrieve Wikipedia passages that answer the question |
| Synthetic Bitext | Retrieve parallel sentences. |
| Synthetic STS | Retrieve semantically similar text. |
| T2Ranking | Given a Chinese search query, retrieve web passages that answer the question |

Table 11: Instructions used during fine-tuning on datasets with one or two distinct instructions.

| Task Type | Instruction |
|---|---|
| STS | Retrieve semantically similar text. |
| Summarization | Given a news summary, retrieve other semantically similar summaries |
| BitextMining | Retrieve parallel sentences. |

Table 12: Instructions used for the MTEB evaluation per task type.

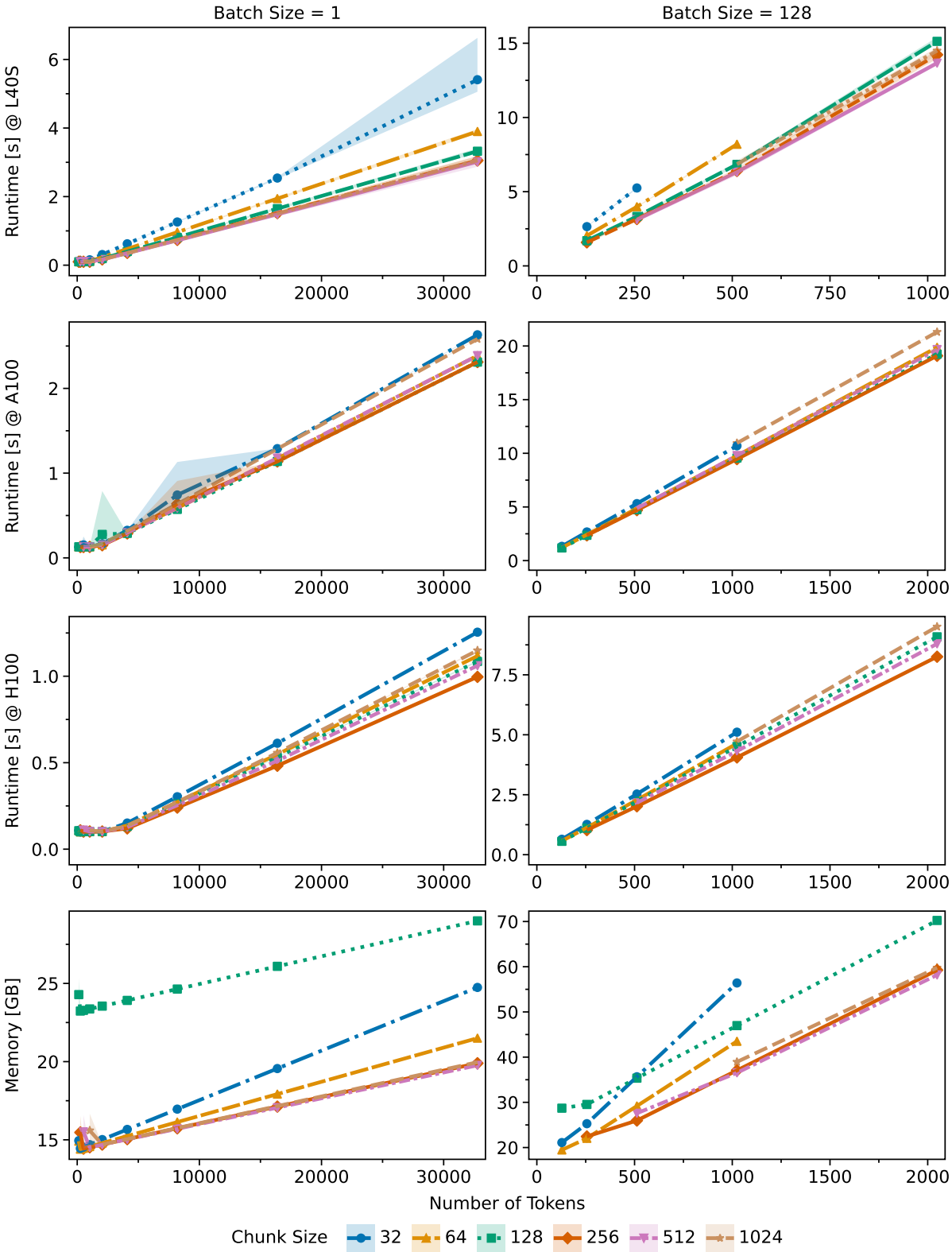



Figure 8: Runtime of Codestral Mamba2 7B using fully horizontal chunked inference with varying chunk sizes for batch sizes 1 and 128, measured on NVIDIA L40S, A100, and H100. Memory usage is consistent across devices and is therefore shown once. Values are reported as mean, minimum, and maximum over three runs.

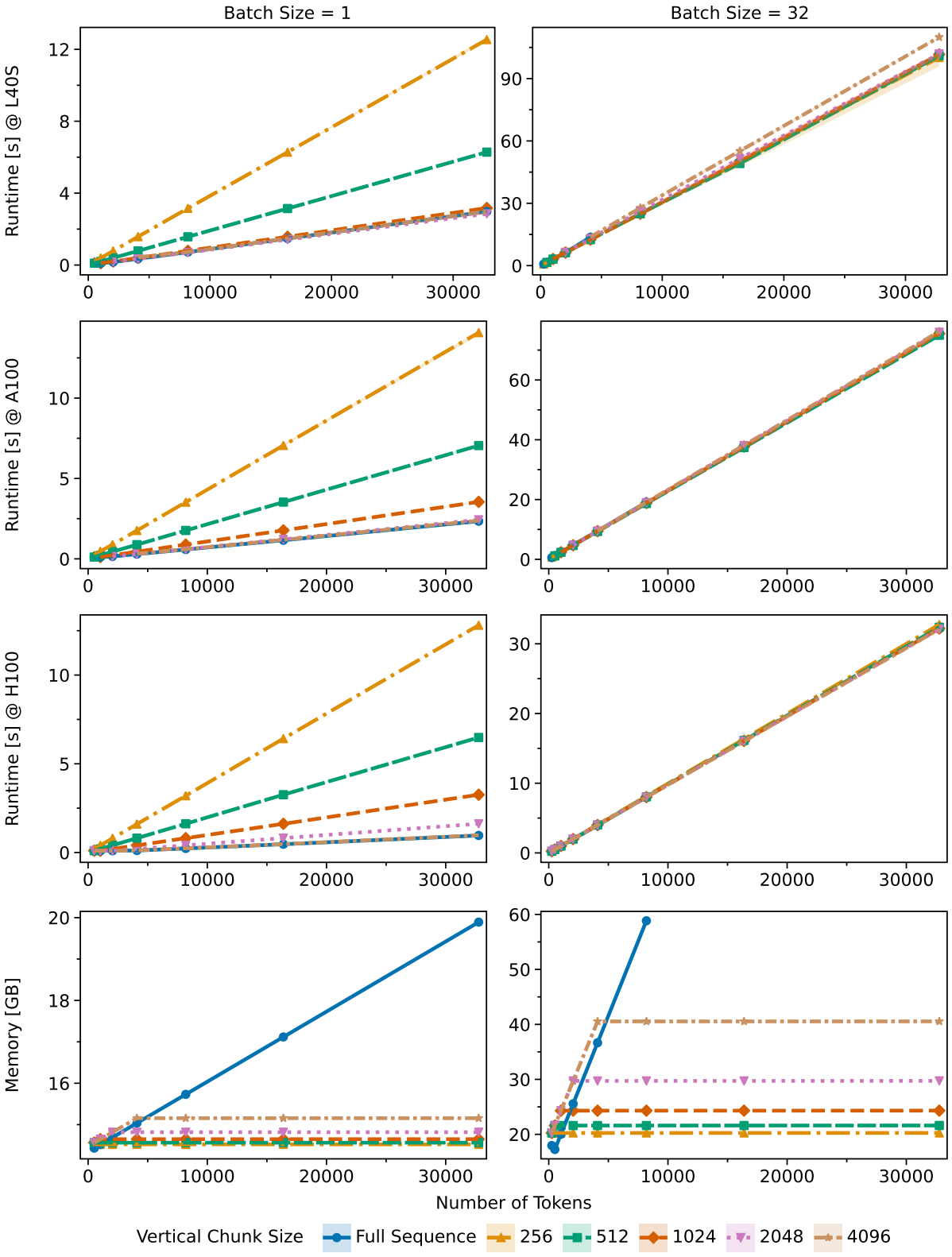



Figure 9: Runtime and memory usage of Codestral Mamba2 7B using vertically chunked inference with a fixed intra-layer chunk size of 256 and varying vertical chunk sizes for batch sizes 1 and 32, measured on NVIDIA L40S, A100, and H100. Memory usage is consistent across devices and is therefore shown once. Values are reported as mean, minimum, and maximum over three runs.

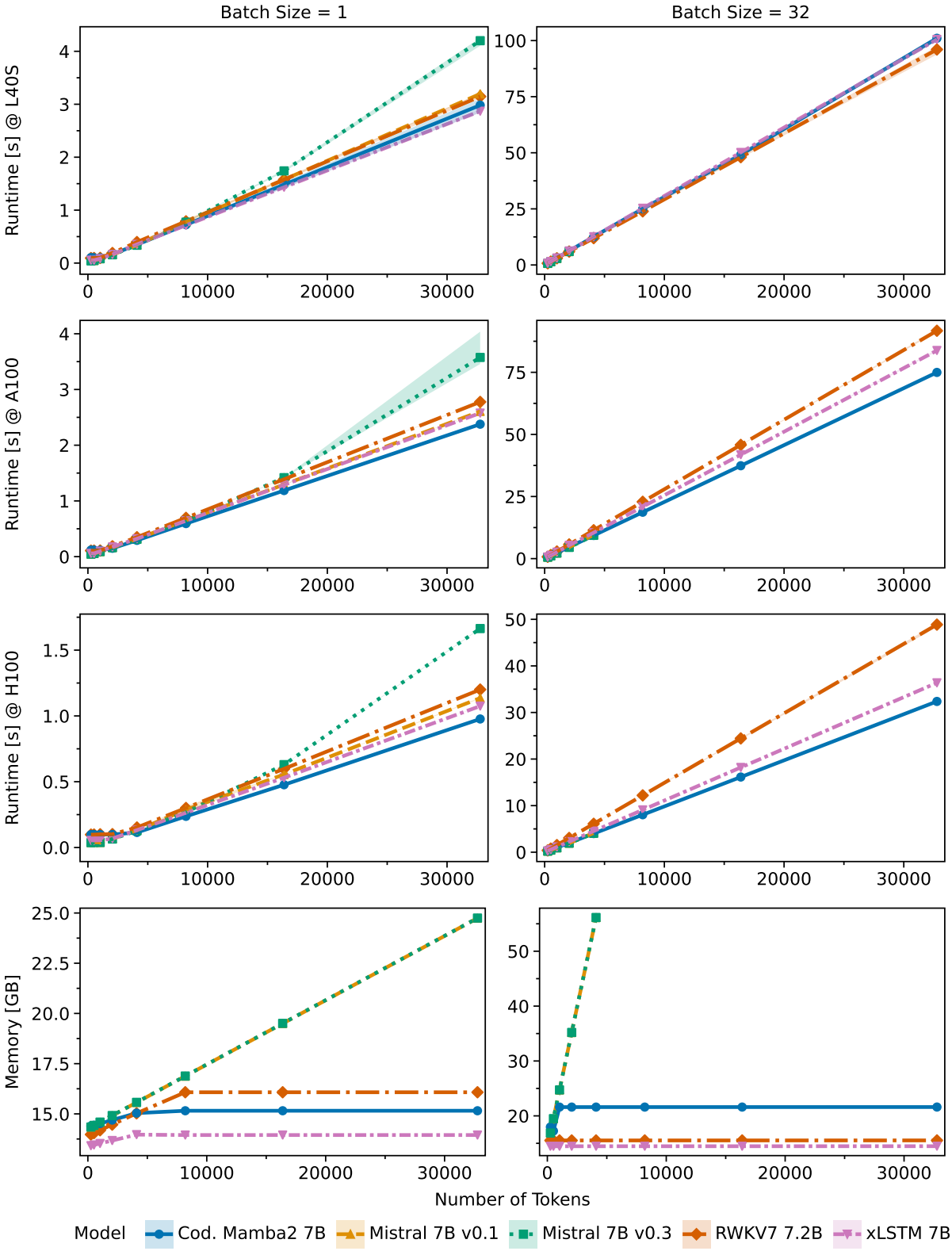



Figure 10: Runtime and memory usage of Codestral Mamba2 7B, RWKV7 7.2B, and xLSTM 7B compared to Mistral 7B `v0.1` and `v0.3`, measured on NVIDIA L40S, A100, and H100. Recurrent models use vertically chunked inference: Codestral Mamba2 7B with intra-layer chunk size 256 and vertical chunk sizes 512 (batch 32) and 4096 (batch 1), RWKV7 7.2B with chunk size 32 and vertical chunk sizes 64 (batch 32) and 4096 (batch 1), and xLSTM 7B with chunk size 64 and vertical chunk sizes 128 (batch 32) and 4096 (batch 1). Transformer-based models use FlashAttention-2. Latent states are stored across layers only for input lengths exceeding the vertical chunk size. Memory usage is consistent across devices and is therefore shown once. Values are reported as mean, minimum, and maximum over three runs.

| Task | Instruction |
|---|---|
| AmazonCounterfactualClassification | Classify a given Amazon customer review text as either counterfactual or not-counterfactual |
| AmazonPolarityClassification | Classify Amazon reviews into positive or negative sentiment |
| AmazonReviewsClassification | Classify the given Amazon review into its appropriate rating category |
| ArguAna | Given a claim, find documents that refute the claim |
| Arxiv(Hierarchical)ClusteringP2P | Identify the main and secondary category of Arxiv papers based on the titles and abstracts |
| Arxiv(Hierarchical)ClusteringS2S | Identify the main and secondary category of Arxiv papers based on the titles |
| AskUbuntuDupQuestions | Retrieve duplicate questions from AskUbuntu forum |
| Banking77Classification | Given an online banking query, find the corresponding intents |
| BiorxivClusteringP2P(.v2) | 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 |
| CQADupstackGamingRetrieval | Given a question, retrieve detailed question descriptions from Stackexchange that are duplicates to the given question |
| CQADupstackUnixRetrieval | |
| ClimateFEVER(HardNegatives) | Given a claim about climate change, retrieve documents that support or refute the claim |
| CovidRetrieval | Given a question on COVID-19, retrieve news articles that answer the question |
| DBPedia | Given a query, retrieve relevant entity descriptions from DBPedia |
| EmotionClassification | Classify the emotion expressed in the given Twitter message into one of the six emotions: anger, fear, joy, love, sadness, and surprise |
| FEVER(HardNegatives) | 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(HardNegatives) | Given a multi-hop question, retrieve documents that can help answer the question |
| ImdbClassification | Classify the sentiment expressed in the given movie review text from the IMDB dataset |
| MIRACLRetrievalHardNegatives | Given a question, retrieve Wikipedia passages that answer the question |
| NQ | |
| MSMARCO | Given a web search query, retrieve relevant passages that answer the query |
| 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 |
| MassiveIntentClassification | Given a user utterance as a query, find the user intents |
| MassiveScenarioClassification | Given a user utterance as a query, find the user scenarios |
| MedrxivClusteringP2P(.v2) | Identify the main category of Medrxiv papers based on the titles and abstracts |
| MedrxivClusteringS2S(.v2) | Identify the main category of Medrxiv papers based on the titles |
| MindSmallReranking | Retrieve relevant news articles based on user browsing history |
| NFCorpus | Given a question, retrieve relevant documents that best answer the question |
| QuoraRetrieval | Given a question, retrieve questions that are semantically equivalent to the given question |
| RedditClustering | Identify the topic or theme of Reddit posts based on the titles |
| RedditClusteringP2P | Identify the topic or theme of Reddit posts based on the titles and posts |
| SCIDOCS | Given a scientific paper title, retrieve paper abstracts that are cited by the given paper |
| SciDocsRR | Given a title of a scientific paper, retrieve the titles of other relevant papers |
| SciFact | Given a scientific claim, retrieve documents that support or refute the claim |
| SprintDuplicateQuestions | Retrieve duplicate questions from Sprint forum |
| StackExchangeClustering(.v2) | Identify the topic or theme of StackExchange posts based on the titles |
| StackExchangeClusteringP2P(.v2) | Identify the topic or theme of StackExchange posts based on the given paragraphs |
| StackOverflowDupQuestions | Retrieve duplicate questions from StackOverflow forum |
| T2Reranking | Given a Chinese search query, retrieve web passages that answer the question |
| TRECCOVID | Given a query on COVID-19, retrieve documents that answer the query |
| Touche2020(Retrieval.v3) | Given a question, retrieve detailed and persuasive arguments that answer the question |
| 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 |
| TwentyNewsgroupsClustering(.v2) | Identify the topic or theme of the given news articles |
| TwitterSemEval2015 | Retrieve tweets that are semantically similar to the given tweet |
| TwitterURLCorpus | |

Table 13: Instructions used for the MTEB evaluation per task.