Attamba: Attending To Multi-Token States

Yash Akhauri [1 2] **Safeen Huda** [2] **Mohamed S. Abdelfattah** [1]

Abstract

When predicting the next token in a sequence, vanilla transformers compute attention over all previous tokens, resulting in quadratic scaling of compute with sequence length. State-space models compress the entire sequence of tokens into a fixed-dimensional representation to improve efficiency, while other architectures achieve sub-quadratic complexity via low-rank projections or sparse attention patterns over the sequence. In this paper, we introduce Attamba, a novel architecture that uses state-space models to compress chunks of tokens and applies attention on these compressed key-value representations. We find that replacing key and value projections in a transformer with SSMs can improve model quality and enable flexible token chunking, resulting in 24% improved perplexity with transformer of similar KV-Cache and attention footprint, and $\approx 4\times$ smaller KV-Cache and Attention FLOPs for 5% perplexity trade-off. Attamba can perform attention on chunked-sequences of variable length, enabling a smooth transition between quadratic and linear scaling, offering adaptable efficiency gains. [Logs] [Code]

1. Introduction

Transformers have provided an effective and scalable sequence modeling architecture, leading to major strides in natural language processing. This comes at a high cost when processing long sequences due to the quadratic complexity of attention. Efforts to ameliorate this inefficiency have faced challenges for tasks requiring extended contexts, as dropping tokens that may later need to be referenced can render many token-pruning techniques (Zhang et al., 2023; Xiao et al.) ineffective. To address the inefficiency of standard attention, several approaches have been developed. KV-Cache compression techniques, such as Palu(Chang

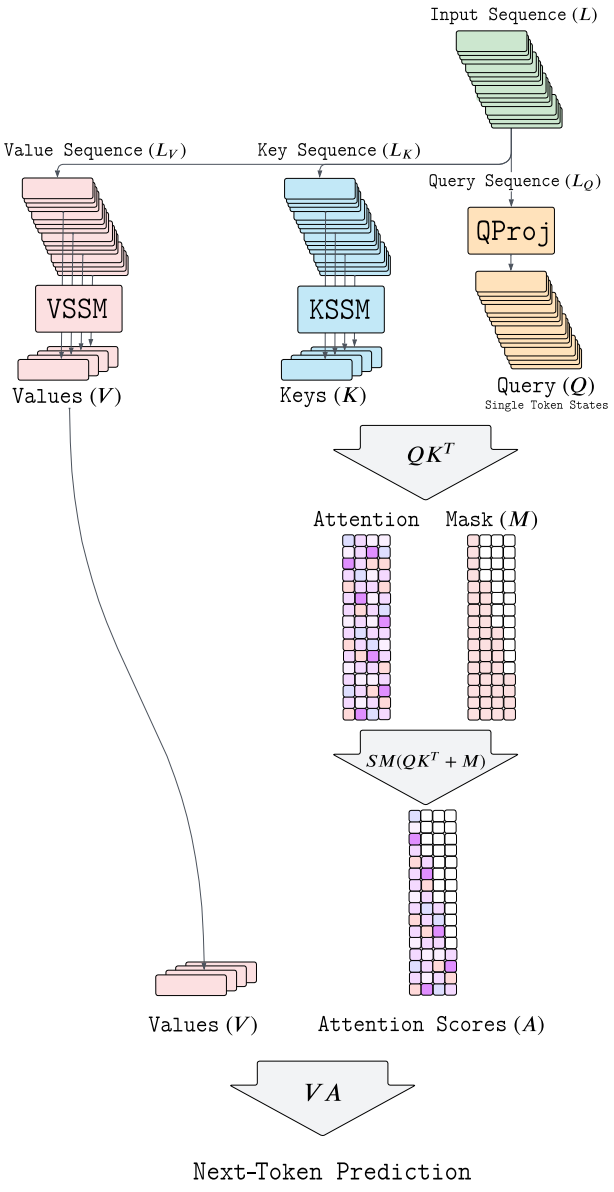

Figure 1. Attamba uses State-Space Models (SSM) to compress key-value sequences into token chunks (e.g., chunks of $P = 4$), reducing the attention map and KV-Cache size by $P\times$ by storing only chunk boundaries.

[1]Cornell University [2]Google. Correspondence to: Yash Akhauri <ya255@cornell.edu>.

et al., 2024) uses low-rank projections to compress hidden dimensions, while ShadowKV (Sun et al., 2024) uses low-rank key caching for long-context inference. Methods such as LinFormer (Wang et al., 2020) and PerFormer (Choromanski et al.) use low-rank approximations or kernel-based projections to reduce complexity of attention. Sparse attention models such as BigBird (Zaheer et al., 2020) adopt fixed attention patterns, but these can fail in settings where static sparsity may not capture necessary interactions.

In contrast, State-Space Models (SSMs) (Gu et al.; 2020) including architectures like Mamba (Gu & Dao, 2023; Dao & Gu) compress entire sequence histories into fixed-dimensional states, offering linear complexity. However, SSMs face challenges in representing arbitrarily long contexts with the same expressivity as the attention mechanism. Stuffed Mamba (Chen et al., 2024) highlights the phenomenon of *state collapse*, which arises when the recurrent state of RNN-based architectures like Mamba fail to generalize to sequences longer than those seen during training (Wang & Li; Merrill et al.). Despite efficient memory use, the fixed-dimensional state of SSMs has an upper bound on information representation, which once exceeded, cannot effectively retain earlier contextual information.

Our key insight is that this limitation can be leveraged: SSMs can be adapted to learn how to compress chunks of tokens into meaningful, single-token states that preserve essential information. By training SSMs to perform variable-length token chunking, SSMs can consolidate sequences of tokens into compact representations, which can then be processed by the standard attention mechanism, reducing the L^2 attention operations by a factor of chunk size. In this paper, we introduce *Attamba*, a novel architecture that combines SSMs and Attention. As shown in Figure 1, Attamba uses SSMs for key-value projections, enabling flexible *chunked attention* by compressing multiple tokens into a single state. As illustrated in Figure 2, Attamba significantly reduces KV-cache memory and attention computation costs by caching only the compressed chunk boundaries, rather than all tokens. Our contributions are:

- **Integration of SSMs into Attention:** By replacing key-value projection matrices with SSM blocks, we demonstrate that it is possible to compress multiple tokens into one representation, while effectively attending to these representations.

- **Efficient Token Chunking**: We introduce cyclic token chunking when compressing multiple tokens to a single SSM state to reduce bias from fixed boundaries. We also demonstrate the feasibility of variable-length token chunking and present over $8\times$ KV-cache and attention savings with minimal perplexity trade-offs.

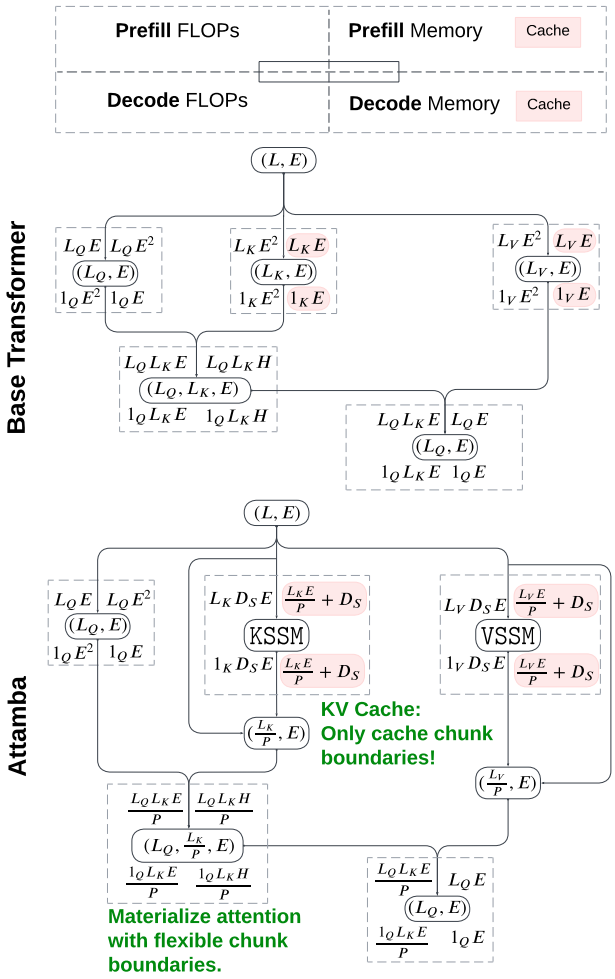

Figure 2. State Space Models (SSMs) efficiently encode multiple tokens into a single representation. By compressing key (K) and value (V) sequences into chunked representations, SSMs maintain essential contextual information, enabling efficient query (Q) interactions. This approach minimizes KV-Cache size by storing only chunk boundaries and reduces the computational cost of attention. Attamba demonstrates robustness to randomized chunk boundaries, indicating the potential for flexible computation-quality trade-offs. Approximate FLOPs/Memory shown, constants ignored. Variables: L (Sequence length), P (Chunk size), D_S (SSM state dimension), E (Model dimension).

2. Related Work

Attention:

Transformers are foundational for language modeling but face challenges due to the quadratic complexity of attention, which grows with the square of the sequence length. This makes attention computation both memory-intensive and computationally expensive. Additionally, during autoregressive inference, the key-value cache size grows linearly with the sequence length and embedding dimension, adding

significant memory overhead. These factors limit efficiency and scalability, especially for long-context applications. Efforts to mitigate this inefficiency include *LinFormer* (Wang et al., 2020), which reduces complexity via low-rank factorization (`k`), and *BigBird* (Zaheer et al., 2020), which uses sparse attention patterns (`r,w,g` denoting random, window, global tokens) to handle long sequences more efficiently. *PerFormer* (Choromanski et al.) leverages kernel-based approximations to achieve sub-quadratic complexity. While effective, these methods face limitations in preserving attention expressivity, especially in long-context tasks..

State-Space Models: State-Space Models provide an efficient mechanism for long-sequence processing. *Mamba* (Gu & Dao, 2023) and its successor *Mamba2* (Dao & Gu) are notable implementations that achieve linear complexity by compressing sequence history into fixed-dimensional states. However, these models struggle with information retention over arbitrarily long contexts, as discussed in *Stuffed Mamba* (Chen et al., 2024) which highlights the state collapse issue.

Hybrid Models: Combining the strengths of attention and SSMs, hybrid models have emerged. *Jamba* (Lieber et al., 2024) interleaves Transformer and Mamba layers, using a mixture-of-experts (MoE) approach to manage parameter usage and support long-context modeling efficiently. *Griffin* (De et al., 2024) integrates gated linear recurrences with local attention, achieving efficient scaling and superior performance on extrapolation tasks. Similarly, *Hawk* (De et al., 2024) utilizes recurrent blocks to outperform Mamba on various downstream tasks. Techniques like *Multi-Token Prediction (MTP)* (Gloeckle et al.) optimize efficiency by predicting multiple tokens simultaneously, improving sample efficiency and enabling faster inference. Hybrid approaches like *Samba* (Ren et al., 2024) and *Jamba* explore novel trade-offs between efficiency and expressivity. *Samba* employs sliding-window attention combined with state-space layers. Our approach differentiates itself by directly integrating SSM blocks inside the attention mechanism instead of interleaving SSMs and Transformer blocks.

3. Preliminaries

3.1. Attention

Let $\mathbf{X} \in \mathbb{R}^{n \times e}$ be the input to the attention mechanism, where n is the sequence length and e is the model embedding dimension. The embedding dimension e can be expressed as $e = h \times d$, where h is the number of attention heads, and d is the per-head dimension. The projection matrices $W^Q, W^K, W^V \in \mathbb{R}^{e \times e}$ are used to compute the query, key, and value representations respectively. `SM` denotes the softmax operation, and `Attn` represents the attention computation. S represents the scaled attention scores, and A represents the attention probabilities (normalized weights).

$$\texttt{Attn}(\mathbf{X}) = \underbrace{\texttt{SM}\left(\underbrace{\frac{\mathbf{X}W^Q(\mathbf{X}W^K)^T}{\sqrt{d}}}_{S}\right)}_{A} \cdot \mathbf{X}W^V \quad (1)$$

3.2. SSMs

State Space Models maintain a hidden state $\mathbf{x}(\mathbf{t}) \in \mathbb{R}^{D_S}$, which evolves over time based on the input sequence and state transition matrices. Computing the output sequence from a given input is linear in complexity, requiring $\mathcal{O}(nD_S)$ in time-complexity and $\mathcal{O}(D_S)$ in space. In the Mamba framework (Gu & Dao, 2023), variable-length sequence handling is streamlined using `cu_seqlens` (**c**umulative **u**nique **se**quence **len**gths), which denotes cumulative sequence lengths. This allows efficient indexing of flattened batch sequences, avoiding padding overhead. We leverage `cu_seqlens` for efficient processing of chunked-sequences. In Section A.1, we investigate different schemes for token chunking.

3.3. Auto-regression and Masking

Transformers are trained for next-word prediction, by modelling the probability of each token, given all previous tokens in a sequence. To enforce *causality* (to not attend to future tokens), a causal mask is applied to the attention mechanism during training, when computing $\mathbf{A}$ in Equation 1. Specifically, the causal mask $M \in \mathbb{R}^{n \times n}$ is defined to prevent positions from attending to future tokens as below:

$$M_{i,j} = \begin{cases} 0, & \text{if } j \leq i, \\ -\infty, & \text{if } j > i. \end{cases} \quad (2)$$

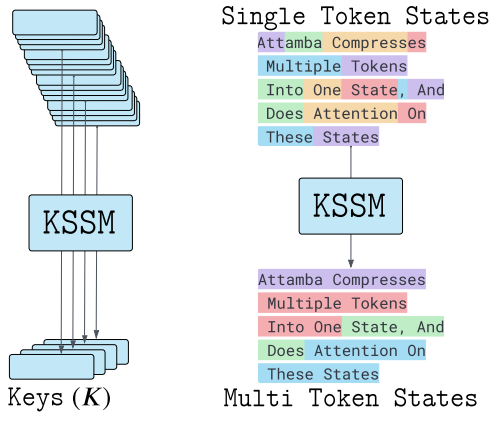

Figure 3. Attamba uses SSM blocks to compress chunks of tokens ($P = 4$ in the example above) into a single token.

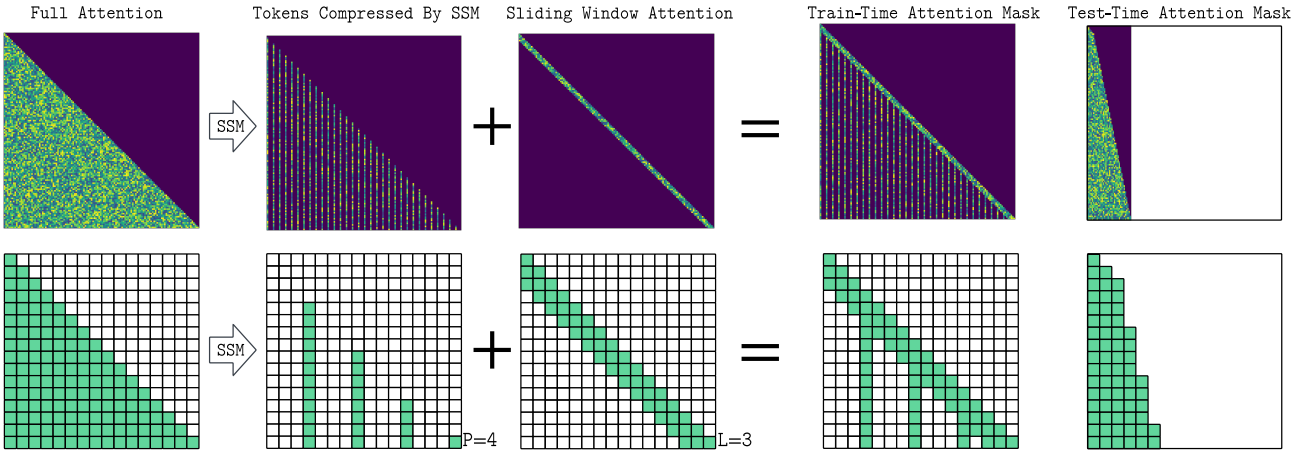

Figure 4. Full-Attention has a purely causal mask, attending to all past tokens. Attamba uses Key-Value SSM blocks to compress chunks of P tokens (e.g. $P = 4$) into one state. Tokens compressed by SSMs are at chunk boundaries. This is incorporated with a sliding-window attention (when $L > 1$). At test-time (inference), only the chunk boundaries and sliding window tokens need to be preserved, reducing KV-Cache and Attention FLOPs.

This M mask is applied before the softmax, resulting in $\mathbf{A} = \texttt{SM}(S + M)$. Setting elements of M to $-\infty$, the attention weights become zero after the softmax, which effectively excludes future tokens from the computation. This mechanism can also be used to emulate a token being omitted by appropriately adjusting the mask.

In next-word prediction tasks, the output at position k depends on all previous tokens from positions 0 to $k-1$, building up cumulative information in hidden states, with each position k capturing knowledge of all tokens up to that point. This property is essential for capturing dependencies across the sequence. Leveraging this cumulative information property of auto-regressive models such as SSMs and Transformers, along with the flexibility of mask M controlling token omissions makes it possible to control range and choice of tokens each position attends to.

4. Attamba: Attentive SSMs

Auto-regressive transformers and SSMs enable us to compress information about prior tokens into a singular, final representation. For next-word prediction, this property is used to transform the representation to what the next word should be. Further, we can control what past information transformers attend to with the attention mask, described in Section 3.3. Attamba leverages these properties to **(1)** compress P tokens into a single token using SSM blocks, exhibiting linear complexity, and **(2)** leverage attention mask to attend to only these compressed states for efficient training and inference. Specifically, we preserve the query sequence length, to enable *causally valid* training of all next-word prediction problems in a given input sequence, and replace the key and value projection matrices with SSM blocks.

4.1. Formulation

Attamba integrates State Space Models (SSMs) into the attention mechanism to efficiently handle long sequences. As seen in Figure 2, it replaces key and value projection matrices with SSM blocks and a residual connection that processes chunks of tokens, reducing computational complexity of attention while preserving context of input sequence.

Let P denote the chunk size, i.e., the number of tokens processed by the SSM at a time. Given an input sequence $\mathbf{X} \in \mathbb{R}^{n \times e}$, the query vector is computed as usual to preserve the auto-regressive nature of transformers. However, the keys and values are obtained by processing the input sequence through SSMs. The sequence is divided into non-overlapping chunks of size P (Figure 3), and each chunk is processed auto-regressively by the SSM. For simplicity, we assume n is divisible by the chunk size P in this discussion, though SSMs can seamlessly handle partial chunks, as they do during auto-regressive inference in Attamba.

Let $\mathbf{X}^{(p)} \in \mathbb{R}^{P \times e}$ denote the p-th chunk of the input sequence, where $p = 1, 2, \ldots, \frac{n}{P}$. The SSM processes each chunk to produce compressed key and value representations:

$$\mathbf{K}^{(p)} = \text{SSM}_K\left(\mathbf{X}^{(p)}\right), \ \mathbf{V}^{(p)} = \text{SSM}_V\left(\mathbf{X}^{(p)}\right) \quad (3)$$

where SSM_K and SSM_V denote the SSMs used for keys and values, respectively.

At **train-time**, we need to preserve all SSM outputs, since next-word prediction problems require attending to incomplete (partial) chunks as well. Thus, we keep the SSM outputs for every token, giving us:

$$\begin{bmatrix}\mathbf{K}_{\text{SSM}}\\ \mathbf{V}_{\text{SSM}}\end{bmatrix} = \begin{bmatrix}\mathbf{K}^{(1)} & \mathbf{K}^{(2)} & \cdots & \mathbf{K}^{(n)}\\ \mathbf{V}^{(1)} & \mathbf{V}^{(2)} & \cdots & \mathbf{V}^{(n)}\end{bmatrix} \in \mathbb{R}^{2\times n\times e} \quad (4)$$

To perform attention, the queries $\mathbf{Q}$ attend to compressed keys-values at chunk boundaries and the latest partial chunk (Self-Attention). Thus, the attention mask M_{train} must account for both causality and chunk boundaries:

$$(M_{\text{train}})_{i,j} = \begin{cases} 0, & \text{if } \left(\left\lfloor \frac{j}{P} \right\rfloor = \left\lfloor \frac{i}{P} \right\rfloor \text{ and } j \le i\right) \\ & \quad \text{or } (j \le i \text{ and } j \bmod P = P-1), \\ -\infty, & \text{otherwise.} \end{cases} \quad (5)$$

At **test-time**, the outputs $\mathbf{K}^{(p)}[-1], \mathbf{V}^{(p)}[-1] \in \mathbb{R}^{1\times e}$ are compressed representations of each chunk, as only the final representation is needed. This is obtained by taking $(\mathbf{K}^{(p)}[-1], \mathbf{V}^{(p)}[-1])$ from Equation 3. By concatenating these, we obtain:

$$\begin{bmatrix}\mathbf{K}_{\text{SSM}}\\ \mathbf{V}_{\text{SSM}}\end{bmatrix} = \begin{bmatrix}\mathbf{K}^{(1)}[-1] & \cdots & \mathbf{K}^{(\frac{n}{P})}[-1]\\ \mathbf{V}^{(1)}[-1] & \cdots & \mathbf{V}^{(\frac{n}{P})}[-1]\end{bmatrix} \in \mathbb{R}^{2(\frac{n}{P})\times e} \quad (6)$$

As shown in Figure 4, at test-time, an appropriate attention mask M_{test} can be constructed:

$$(M_{\text{test}})_{i,j} = \begin{cases} 0, & \text{if } j \le \left\lfloor \frac{i}{P} \right\rfloor, \\ -\infty, & \text{otherwise.} \end{cases} \quad (7)$$

By replacing key and value projections with SSMs that compress chunks of tokens, we reduce the computational cost of the attention mechanism from (n^2e) to (n^2e/P). We can also achieve $\mathcal{O}(nPe)$ complexity if we divide the input sequence length into P chunks, irrespective of sequence length, resembling Attamba-Linear in Figure 13. We find that SSMs are robust to even randomized chunk boundaries, which may facilitate this complexity trade-off.

Chunk Sizes and Leading Tokens: In Attamba, the notion of *chunk-size* (C/P) play a critical role in determining how tokens are compressed using SSMs. The chunk-size refers to the number of consecutive tokens that are grouped together and processed as a single unit by the SSM to create a compressed key-value representation. The *leading tokens* (L) specifies the number of recent tokens that should retain *full attention* after the SSM. This is akin to sliding-window attention and has a constant cost as shown in Figure 5. Chunked attention with $L > 1$ would need us to parse the SSM block outputs as described in Equation 8.

$$\begin{bmatrix}\mathbf{K}_{\text{SSM}}\\ \mathbf{V}_{\text{SSM}}\end{bmatrix} = \begin{bmatrix}\mathbf{K}^{(1)}[-1] & \cdots & \mathbf{K}^{(\frac{n}{P})}[-L:-1] & \mathbf{K}^{(\frac{n}{P})}[-1]\\ \mathbf{V}^{(1)}[-1] & \cdots & \mathbf{V}^{(\frac{n}{P})}[-L:-1] & \mathbf{V}^{(\frac{n}{P})}[-1]\end{bmatrix} \quad (8)$$

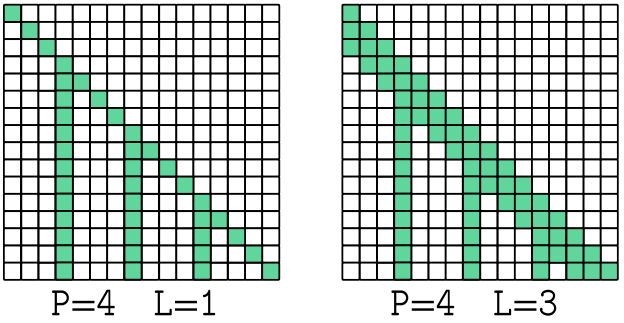

Figure 5. Leading-Tokens (**L**) control how many 'leading' tokens full-attention happens over, preserving full-attention on the newest tokens. This resembles Sliding-Window attention. Chunk-size (**P**) controls how many tokens are chunked by the SSM.

Other Design Considerations: In developing Attamba, several design choices were empirically validated, detailed in Appendix A. First, we found that removing Key-Value projection weights did not significantly impact model quality (1% perplexity difference), simplifying the architecture. Secondly, cyclic chunk boundaries across layers mitigate bias introduced by fixed chunk boundaries on the input sequence (5% improvement). Third, increasing SSM state dimensions beyond $D_s > 32$ yielded diminishing returns on $P = 8$ ($< 1\%$ perplexity difference), allowing us to minimize SSM parameter overhead. Fourth, preserving leading tokens as un-chunked ensured improved model quality by maintaining full attention on recent tokens, emulating sliding window behavior (8.5% improvement with $L = P$). Further, incorporating residual connections in Key-Value SSMs improved model quality, even without K-V projections. Finally, Attamba was robust to even **randomized chunk boundaries** (at both train and test time!). These are explained in more detail in the Appendix A.

5. Experiments

In this section, we present experimental results comparing the WikiText2 test-set perplexity. Training is done on 10% of dclm-baseline-1.0 (Li et al., 2024), with a batch size of 16, sequence length of 1024. We use the Meta Lingua (Videau et al., 2024) framework. Unless otherwise specified, we train on approximately 1B tokens (982M tokens). *Where relevant, we add the final WK2 perplexity in the graph legend.*

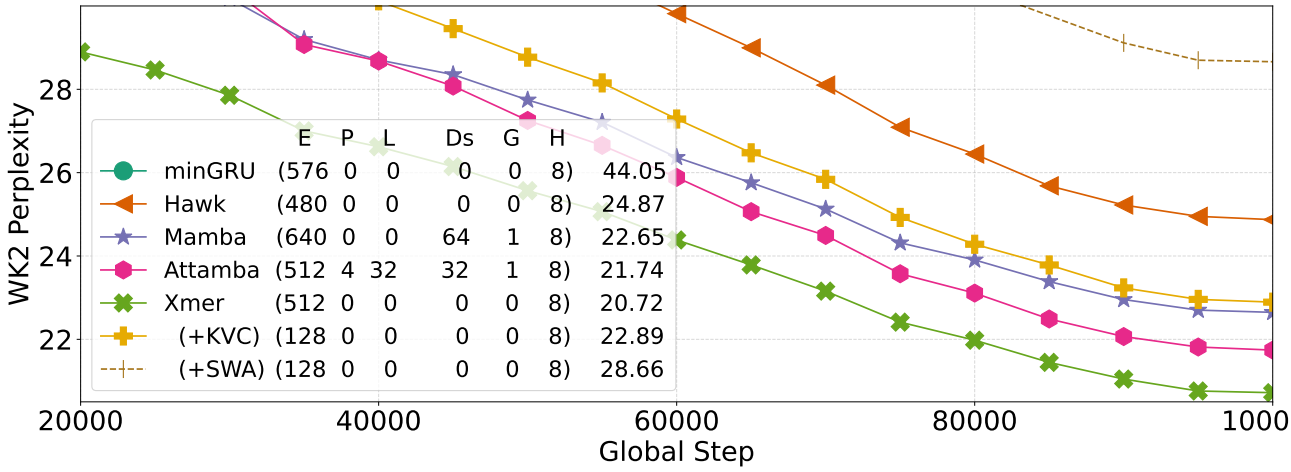

Figure 6. Comparing Attamba with SSMs (Mamba), minGRU, Hawk and Transformers (Xmer) by training on 8 billion tokens. `E, P, L, Ds, G, H` denote Model-Dim, Chunk Size, Leading Tokens, SSM State-Dim, Num. Groups and Num. Heads respectively, 0 when not applicable. Models $\in [60, 64]$M params, with Transformer having $4\times$ larger KV and attention map footprint. (+KV) & (+SWA) transformer variants are 53M params, to match Attamba KV-Cache and Attention Map memory footprint more closely. [Logs]

5.1. On transformer baselines

Attamba compresses the sequence length for keys and values, significantly reducing KV-Cache size and the operational intensity of the L^2 attention map, with the majority of savings occurring in inference-time activations. Comparing Attamba directly with a transformer of similar parameter count is not ideal, as traditional transformers incur much larger KV-Cache and attention map overhead. To provide a fairer context for Attamba's performance, we construct baselines with reduced KV-Cache sizes and attention map dimensions, detailed in Appendix A. Specifically, for transformers, we emulate smaller KV-Cache sizes by reducing the attention model dimension F such that $F = \frac{E}{P}$, and smaller attention maps by employing sliding window attention (SWA) during evaluation.

In Figure 7, we observe that transformers with a $4\times$ to $8\times$ larger KV-Cache and attention map can outperform Attamba, but these comparisons do not reflect equivalent memory or computational constraints. When matched for KV-Cache and attention map size, Attamba consistently outperforms transformer baselines, showcasing its efficiency. Furthermore, as sequence length increases, transformers must reduce their attention dimension F proportionally, resulting in greater trade-offs in quality. Attamba, on the other hand, demonstrates robust and scalable token compression, with a mere 2.2% perplexity increase when transitioning from $P = 4$ to $P = 8$, highlighting its ability to balance efficiency and model quality effectively, particularly for long-context tasks.

5.2. Comparison with Mamba, minGRU, Hawk

We conduct an extended training experiment with Attamba, Transformer (Xmer), minGRU (Feng et al., 2024), Hawk (De et al., 2024), Mamba models within the parameter budget $\in [60, 64]M$ params (roughly *iso-Parameter* configurations), training for 100,000 steps over 8 billion tokens, as shown in Figure 6. We also construct appropriate baselines for transformers, by reducing the model-dimension (specifically in the attention mechanism) to emulate KV-Cache compression (+KVC) and testing sliding window attention (+SWA), similar to Section 5.1.

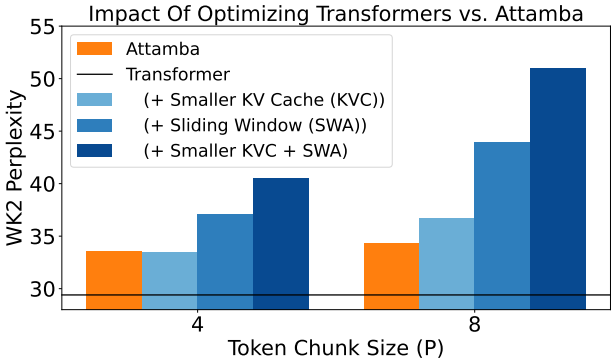

Figure 7. Comparing Attamba with a base transformer with matching parameter counts. Further, we train variants with smaller KV-Cache size to match Attamba. Additionally, to match the attention map size, we evaluate these models in *Sliding Window* attention with window size $= \frac{L}{P}$. Attamba significantly out-performs a similarly sized transformer baseline (Smaller KVC + SWA). [Logs]

We find that Attamba out-performs fair transformer baselines as well as Mamba. Mamba will still show better performance scaling for extremely long context, but model quality may suffer (Wang & Li). Since Attamba uses SSMs to compress fixed-sized chunks of tokens, SSMs will not have to scale beyond their trained chunk-length, but merely attend to compressed token representations. Transformer (+KVC) is 5% worse, but still materializes a L^2 attention map. Transformer (+SWA) is conducted on top of the Transformer (+KVC) variant, with a notable dip in perplexity due to a significantly constrained attention map.

6. Limitations

Currently, most of our training is limited to only 1B tokens on a 60M parameter model on a single A6000 GPU. Further, our test-time evaluation is on WikiText2 (Merity et al., 2022), a task that is highly local. Thus, our variants Attamba with chunk-size 128 performs extremely well (Appendix A). While this variant offers a $128\times$ reduction in KV-Cache size and attention op-intensity over longer contexts, we also maintain full attention on the leading (latest) 128 tokens. This is why, it out-performs even Attamba with chunk size 4. From Figure 17, we can see this in more detail, specifically, our Attamba P128 L1 variant (true $128\times$ KV-Cache reduction, with only 1 leading token (for self-attention)) performs significantly worse than Transformers. However, Attamba with chunk size 8 and 64 uncompressed leading tokens gives us a KV-Cost : $(\frac{L_K+L_V}{8} + 2 \times 64)E$ which offers $\approx 8\times$ KV-compression and $\approx 1/8$ attention computation cost with a 10% perplexity trade-off. However, a transformer with model-dimension $E/8$ on attention performs 7.11% worse, and has no memory savings on the attention map, as each head would still materialize the L^2 tensor. Further, our method leads to no improvements in the FFN, as we preserve the query sequence length to enable auto-regressive training of the transformer. Finding the right transformer design for a fair comparison is key to better understand trade-offs of attention on compressed states. Modifications in chunking strategy, or training on more tokens may alleviate issues with high chunk-sizes, but more thorough evaluation of this methodology on Long-Context evaluation, retrieval and other tasks are key to understand if effective attention can be achieved on compressed states. Finally, every model we have reported has been trained from scratch, exploring fine-tuning strategies and inference-time studies on chunk boundary robustness of models, as well as the impact of leading token L (sliding window attention) need to be tested.

7. Conclusion

Attamba introduces a novel approach to efficiently handle long sequences in transformers by integrating State-Space Models (SSMs) to compress tokens, reducing attention cost and KV-cache memory requirements. Experiments show that cyclic chunking outperforms other strategies, maintaining competitive performance with significant efficiency gains. By replacing conventional key-value projection matrices with SSMs and incorporating variable-length token chunking, Attamba effectively balances computational and memory efficiency, potentially enabling a smooth transition between quadratic and linear scaling if SSMs are flexible to chunk boundary lengths. However, our evaluation was limited to small-scale models and local tasks, such as WikiText2. Consequently, the observed performance improvements may not directly generalize to long-context benchmarks or billion-parameter language models. Future research should focus on extensive evaluation across a wider range of long-context tasks, exploration of token importance-based chunking strategies, and a deeper investigation into the trade-offs between efficiency and information retention in compressed states.

References

Akhauri, Y., AbouElhamayed, A. F., Dotzel, J., Zhang, Z., Rush, A. M., Huda, S., and Abdelfattah, M. S. ShadowLLM: Predictor-based contextual sparsity for large language models. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N. (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 19154–19167, Miami, Florida, USA, November 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-main.1068.

Chang, C.-C., Lin, W.-C., Lin, C.-Y., Chen, C.-Y., Hu, Y.-F., Wang, P.-S., Huang, N.-C., Ceze, L., Abdelfattah, M. S., and Wu, K.-C. Palu: Compressing kv-cache with low-rank projection, 2024. URL https://arxiv.org/abs/2407.21118.

Chen, Y., Zhang, X., Hu, S., Han, X., Liu, Z., and Sun, M. Stuffed mamba: State collapse and state capacity of rnn-based long-context modeling. *arXiv preprint arXiv:2410.07145*, 2024.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. In *International Conference on Learning Representations*.

Dao, T. and Gu, A. Transformers are ssms: Generalized models and efficient algorithms through structured state

space duality. In *Forty-first International Conference on Machine Learning*.

De, S., Smith, S. L., Fernando, A., Botev, A., Cristian-Muraru, G., Gu, A., Haroun, R., Berrada, L., Chen, Y., Srinivasan, S., et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

Feng, L., Tung, F., Ahmed, M. O., Bengio, Y., and Hajimirsadegh, H. Were rnns all we needed?, 2024. URL https://arxiv.org/abs/2410.01201.

Gloeckle, F., Idrissi, B. Y., Roziere, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. In *Forty-first International Conference on Machine Learning*.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Gu, A., Goel, K., and Re, C. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*.

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Li, J., Fang, A., Smyrnis, G., Ivgi, M., Jordan, M., Gadre, S., Bansal, H., Guha, E., Keh, S., Arora, K., Garg, S., Xin, R., Muennighoff, N., Heckel, R., Mercat, J., Chen, M., Gururangan, S., Wortsman, M., Albalak, A., Bitton, Y., Nezhurina, M., Abbas, A., Hsieh, C.-Y., Ghosh, D., Gardner, J., Kilian, M., Zhang, H., Shao, R., Pratt, S., Sanyal, S., Ilharco, G., Daras, G., Marathe, K., Gokaslan, A., Zhang, J., Chandu, K., Nguyen, T., Vasiljevic, I., Kakade, S., Song, S., Sanghavi, S., Faghri, F., Oh, S., Zettlemoyer, L., Lo, K., El-Nouby, A., Pouransari, H., Toshev, A., Wang, S., Groeneveld, D., Soldaini, L., Koh, P. W., Jitsev, J., Kollar, T., Dimakis, A. G., Carmon, Y., Dave, A., Schmidt, L., and Shankar, V. Datacomp-lm: In search of the next generation of training sets for language models, 2024.

Lieber, O., Lenz, B., Bata, H., Cohen, G., Osin, J., Dalmedigos, I., Safahi, E., Meirom, S., Belinkov, Y., Shalev-Shwartz, S., et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations*, 2022.

Merrill, W., Petty, J., and Sabharwal, A. The illusion of state in state-space models. In *Forty-first International Conference on Machine Learning*.

Ren, L., Liu, Y., Lu, Y., Shen, Y., Liang, C., and Chen, W. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*, 2024.

Sun, H., Chang, L.-W., Bao, W., Zheng, S., Zheng, N., Liu, X., Dong, H., Chi, Y., and Chen, B. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference, 2024. URL https://arxiv.org/abs/2410.21465.

Videau, M., Idrissi, B. Y., Haziza, D., Wehrstedt, L., Copet, J., Teytaud, O., and Lopez-Paz, D. Meta lingua: A minimal PyTorch LLM training library, 2024. URL https://github.com/facebookresearch/lingua.

Wang, S. and Li, Q. Stablessm: Alleviating the curse of memory in state-space models through stable reparameterization. In *Forty-first International Conference on Machine Learning*.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Xiao, G., Tian, Y., Chen, B., Han, S., and Lewis, M. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.

Zhang, Z., Sheng, Y., Zhou, T., Chen, T., Zheng, L., Cai, R., Song, Z., Tian, Y., Ré, C., Barrett, C., et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

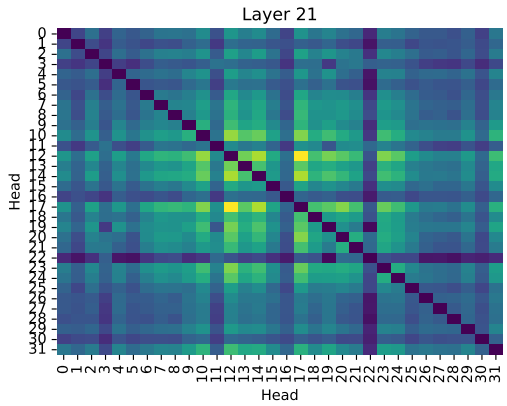

Figure 8. Each head in Llama-2-7B attends to tokens in a manner that is largely uncorrelated (Kendall-Tau $\in [-0.2, 0.8]$) with other heads.

A. Appendix

A.1. On Token Chunking

Processing sequences in fixed-size chunks simplifies implementation, but can limit models flexibility. Prior research (Zhang et al., 2023) has found that certain tokens contribute largely to the perplexity and are contextually important (Akhauri et al., 2024). In this context, having chunk boundaries at *important* tokens for a given query can improve model quality, and maintaining this flexibility for research in token importance prediction can unlock improved efficient language modeling. To enable efficient processing of sequences with arbitrary chunk boundaries in the SSM, we do not reshape or explicitly chunk the sequence. Instead, we utilize the `cu_seqlens` tensor in the Mamba library. This allows us to handle variable-length chunk boundaries without padding overhead. Figure 9 depicts token-chunking strategies we try. Random-Chunking partitions the sequence into P chunks with sizes $\{s_i\}_{i=1}^{P}$, where $s_i \sim \text{Random}(S)$ and $\sum_{i=1}^{P} s_i = n$. From Figure 14, we can see that Random-Chunking works as well as Uniform-Chunking, indicating that SSM based token chunking is flexible.

A.1.1. Cyclic Chunking

Fixed chunk boundaries intorduce biases into the model, as tokens near chunk boundaries may be over-represented due to their position. To aim to mitigate this, we employ a cyclic chunking strategy, with different layers using chunk boundaries with a layer offset. Essentially, the chunk boundary is shifted by the index of the current layer. This ensures different layers process different token groupings, distributing boundary effects across the model.

By varying chunk boundaries across layers we encourage the SSM to be robust to chunk boundaries. We experiment with more chunk boundary decision strategies detailed in the next subsection, but find that cyclic chunking is a simple and effective strategy.

A.1.2. On chunk boundary selection

In addition to cyclic chunking, we explore alternative strategies for determining chunk boundaries to improve model performance. One such method, referred to as **FAttn**, involves using full attention in the first layer to identify important tokens based on attention magnitudes. Specifically, we compute the attention weights in the first layer using the standard full attention mechanism and select the sequence positions with the highest attention scores as chunk boundaries for subsequent layers. This aims to place chunk boundaries at tokens deemed important by the model, potentially enhancing the quality of the compressed representations.

Another approach, termed **FSSM**, utilizes the attention map of chunks from the first layer with uniform chunk boundaries. We compute the attention scores for each chunk and identify the top k chunks with the highest attention values. These selected chunks are then split into 2 smaller chunks in the subsequent layers, effectively allocating more resources to the most informative parts of the sequence.

While we experiment with an array of chunk boundary selection methods, we found that cyclic chunk boundaries yield the best quality improvements. On the other hand, **FSSM** and **FAttn** do not aid chunk boundary selection too much. This may be attributed to our finding that different heads attend to different tokens, and using the first layer to decide all head-boundaries is worse than randomized/cyclic methods. This effect is visible even within a single layer on the Llama-2-7B model, in Figure 8 we can see that for 1024 token context on WikiText2, each head on layer 21 has low correlation between tokens attended to.

A.2. On Pseudo-Chunking

We find that SSMs can serve as a drop-in replacement for the key-value projection matrices, enabling us to save on KV-Cache and the quadratic attention cost by token chunking. However, we can also *pseudo-chunk* the input. That is, given a parameter budget for model size, we can use the SSM as a replacement for projection matrix, and maintain full-attention. This is more computationally expensive, but also improves model quality. Psuedo-chunking can be thought of as Attamba, where **L** (Leading Tokens in Figure 5) is the same as the sequence length.

B. Experiments

In this section, we present experimental results comparing the WikiText2 test-set perplexities during model training for a 60M parameter transformer model, with 8 layers, 8

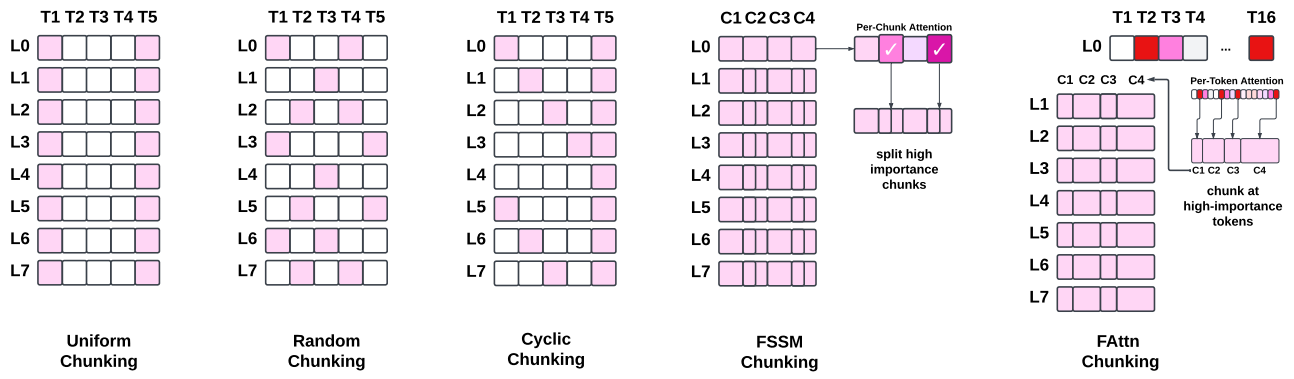

Figure 9. Different token-chunking strategies we investigate. `L`, `T`, `C` represent layer, token and chunk respectively.

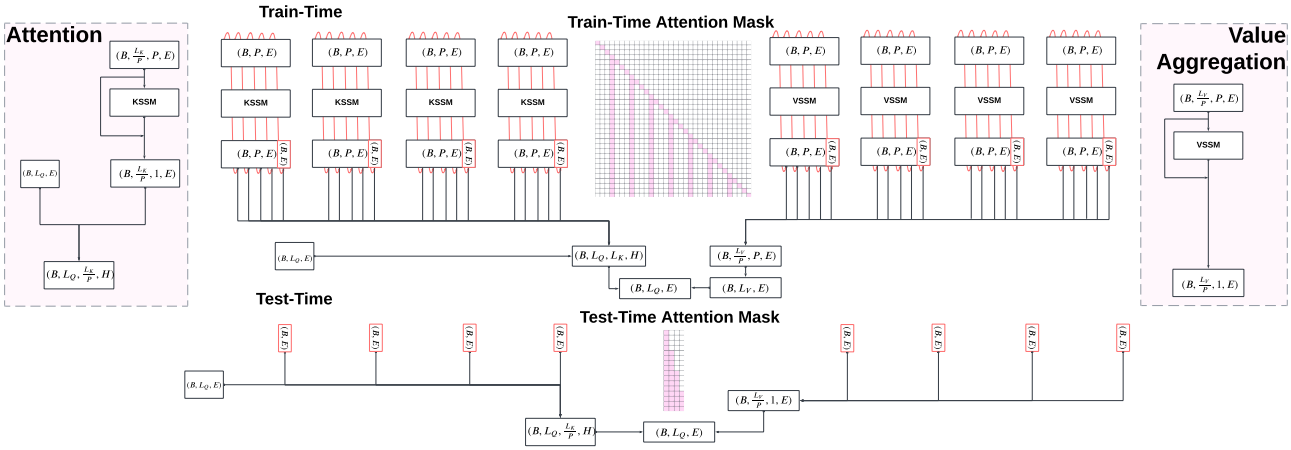

Figure 10. Attamba employs Key and Value State-Space Models (SSMs) to accumulate local information within chunks of tokens. At test time, only the final accumulated activations from each chunk are used in the standard attention mechanism. The red lines denote the auto-regressive SSMs, accumulating *causally valid* local context within chunks. This approach significantly reduces attention complexity by compressing multiple tokens into single representations, while preserving essential contextual information from each chunk.

heads and 512 model-dimension on a single A6000 GPU. Training is done on 10% of dclm-baseline-1.0 (Li et al., 2024), with a batch size of 16, sequence length of 1024. We use the Meta Lingua (Videau et al., 2024) framework. Unless otherwise specified, we train on approximately 1B tokens (982,630,400 tokens). *Where relevant, we add the final WK2 perplexity in the graph legend.*

SSMs For Key-Value Projections: We replace the KV projection matrices with SSMs to enable chunked-attention. In Figure 11, we compare the WikiText2 perplexities. We use uniform 8-token chunking and compare models with and without KV-weight-projections. We find marginal benefits in perplexity by keeping the KV projection matrices before the SSM, and decide to remove it. This also reduces the parameter count and overall FLOPs of the model.

SSM Parameter Count: The SSMs need to do the Key-Value projections, but also compress states for accurate attention, as well as information propagation in the value activations. Thus, the hidden-state of the SSM is important. In Figure 12, we study the impact of varying SSM size, from total approximate parameter-overhead of 2M, 4M and 16M parameters on a 60M parameter model. We see that for a token-chunking size of 8, the SSM does not need to be too large, as the benefit is marginal. For the rest of the experiments, we keep the total SSM parameter overhead 4M, but this can likely be optimized with chunk-size.

Chunking Methodology: Chunking can significantly impact model quality. To test it, we try different chunking methodologies *Uniform*, *Random*, *Cyclic*, *FAttn* and *FSSM*. From Figure 14, we can see that cyclic performs the best. However, it is important to note that *Random* chunking performs similarly to *Uniform* chunking, indicating that Attamba is robust to chunking boundaries, and can signifi-

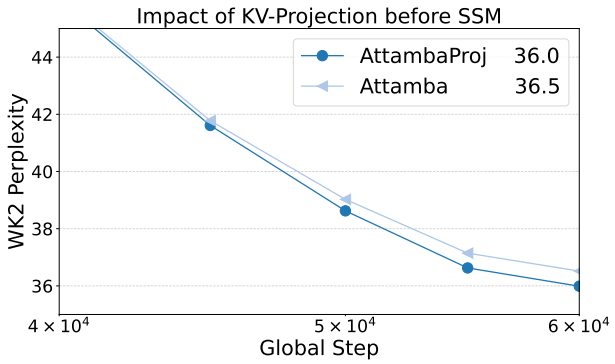

Figure 11. Removing the Key-Value projection matrices when using K-V SSMs does not impact WikiText2 test-perplexity significantly.

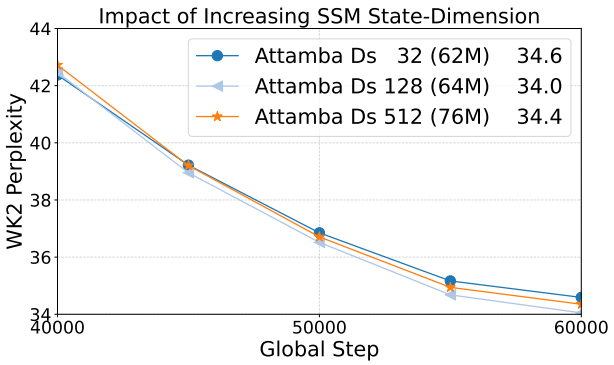

Figure 12. Increasing the state dimension (D_s) of Key-Value SSMs does not improve perplexity when processing chunks of 8 tokens.

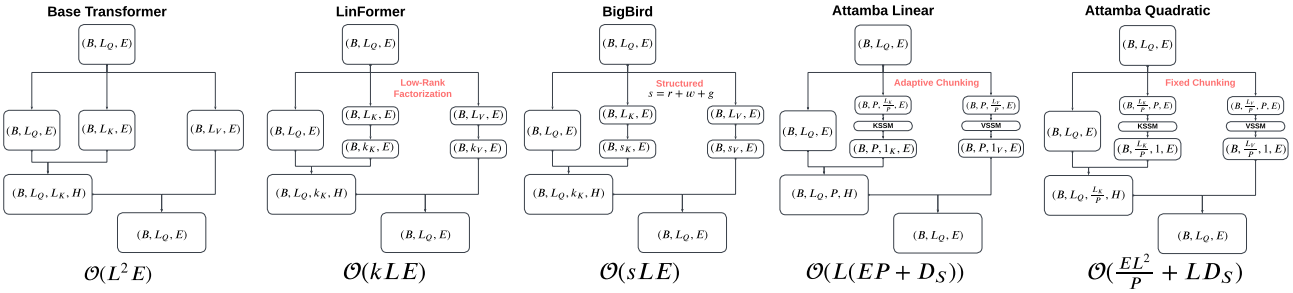

Figure 13. Attamba-Linear maintains linear complexity, by having a fixed-size attention, and dividing the sequence length (L) into chunks. Attamba-Quadratic has quadratic complexity (albeit lower FLOPs/Memory than standard transformer) as the SSM only processes P tokens. `w, r, g, k, E` denote window, random, global, low-rank dimension and model dimension respectively.

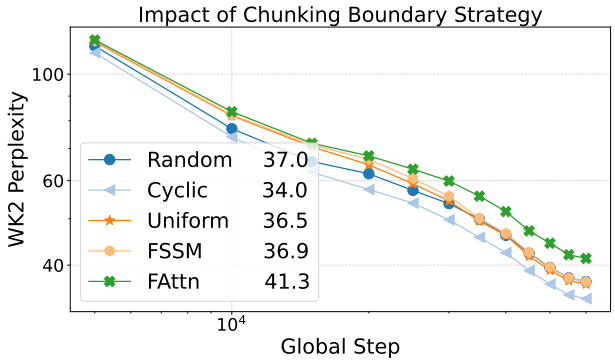

Figure 14. A simple cyclic chunk boundary performs better than other strategies. Notably, randomized chunk boundaries work as well as uniform chunking, indicating potential for flexibility in test-time token chunking.

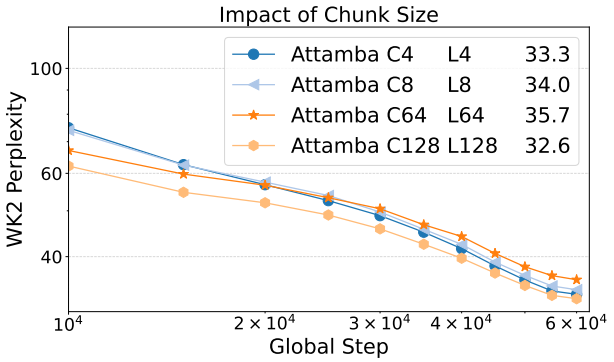

Figure 15. Chunk size of 128 implies a 128× smaller KV-Cache. It outperforms Chunk 4/8/64 because we do full-attention on partial-chunks, giving significant advantage as chunk-size increases on local evaluation tasks like WikiText2.

cantly benefit from research in token importance prediction.

Token Chunking Size: As shown in Figure 5, our chunking methods keeps full attention on the final chunk by default (leading tokens smaller than the chunk size are preserved). This means that as we increase token chunking size, latest `chunk_size` tokens get full attention. This is not compulsory, but we aim to emulate the local sliding window attention with this, as the computational over-head is constant. In Figure 15, we compare different chunk sizes. We

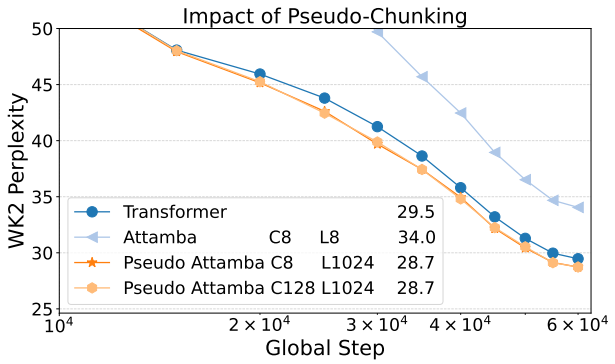

Figure 16. Pseudo-Chunking (replacing Key-Value projection matrices with SSMs, but attending to all tokens) can marginally improve transformer perplexity. (C: Chunk Size)

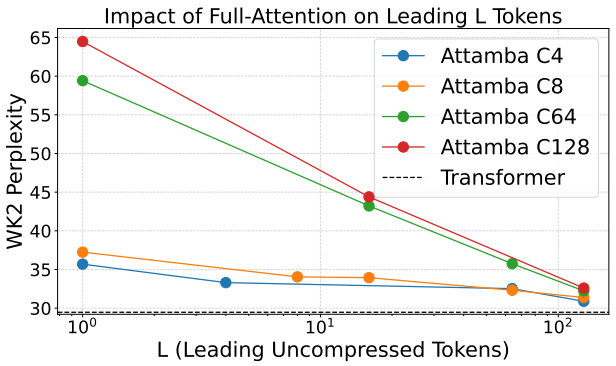

Figure 17. Leading tokens improve test-time perplexity, a proper chunk-size to leading token trade-off is important. This may also indicate limitations in Attamba's ability to compress tokens.

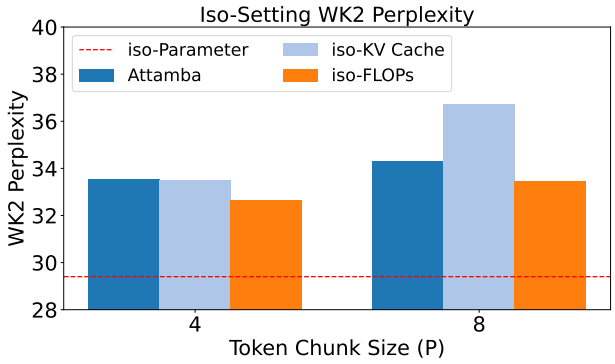

Figure 18. iso-Parameter and iso-FLOPs still has higher memory overhead and does not address the L^2 attention and KV-Cache overhead.

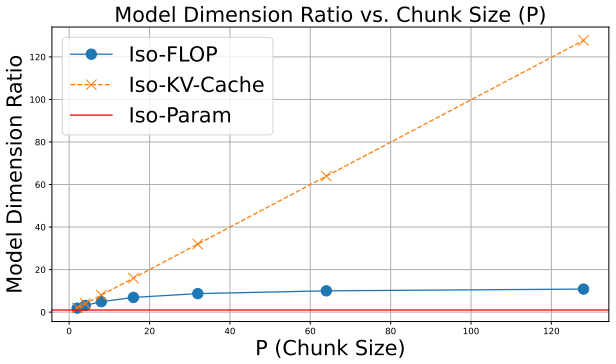

Figure 19. The ratio of Attamba Model Dimension with Transformer Attention Model Dimension (E) required for varying iso-setting baselines as we scale chunk size.

observe a trend where smaller chunk sizes yield better performance, with Chunk 4 outperforming Chunk 8, which outperforms Chunk 64. However, Chunk 128 performs the best, this is simply because WikiText2 is a highly local task, and keeping the latest 128 tokens un-chunked improves perplexity. More rigorous long-context evaluation is required to determine how well token-information is preserved.

Pseudo-Chunking: We replace the KV projection weights with SSMs, and enforce chunk boundaries in the attention mask to emulate KV-Cache optimizations. However, it is also possible to use the SSM so that each token has more information about prior local tokens, without optimizing the transformer for performance. This can be achieved by simply keeping a purely causal mask on Attamba, with no chunk-boundaries. In Figure 16, we find that pseudo-chunking can actually improve transformer performance, even in iso-parameter count settings.

Estimating FLOPs, KVCache and Activation Overhead

Attamba compresses states differently from existing methods of controlling transformer architectures via model dimensions. Comparing Attamba solely with iso-parameter count baselines is inappropriate because transformers produce significantly larger intermediate activations, such as attention maps. To find appropriate transformer baselines, we use a simplified approach to calculate iso-KV-cache size, iso-memory, and iso-FLOPs settings for the *Transformer Block*. These calculations exclude scaling, normalization, and softmax considerations, focusing on high sequence lengths.

We define the following parameters: Transformer attention-only model dimension (F), Attamba model dimension (E), number of heads (H, assumed to be 1 unless otherwise stated), chunk size (P), sequence length (L), SSM dimension (D_S), and batch size (B). To find the right F, we solve simply by substituting the default Attamba configurations, and use this F dimension in the attention mechanism of the base-transformer.

Iso-KV Settings: For iso-KV settings, the appropriate F

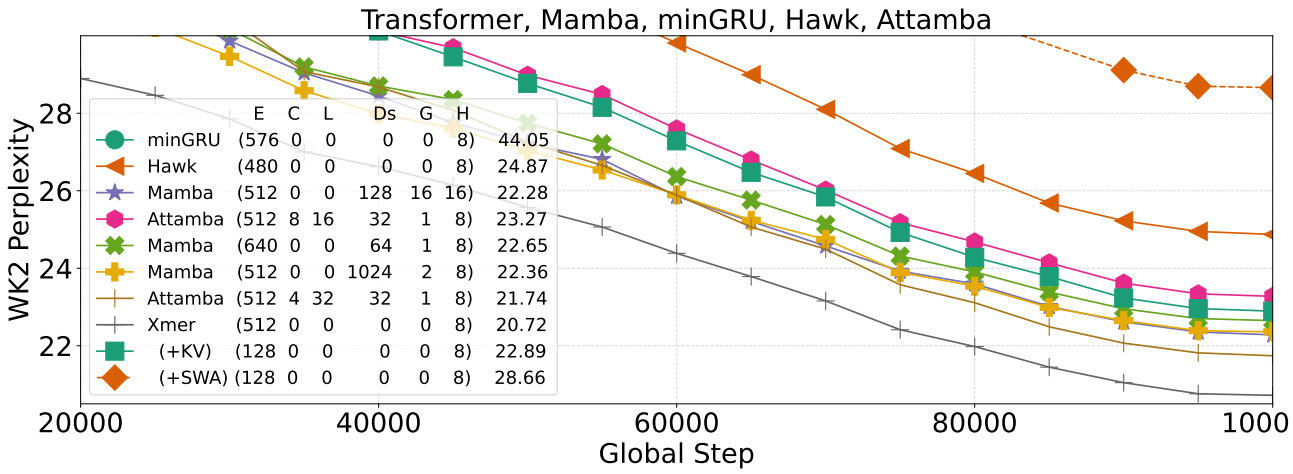

Figure 20. Comparing Attamba with SSMs (Mamba), minGRU, Hawk and Transformers (Xmer) by training on 8 billion tokens. `E, C, L, Ds, G, H` denote Model-Dim, Chunk-Size, Leading-Tokens, SSM State-Dim, Num. Groups and Num. Heads respectively, 0 when not applicable. Models $\in [60, 64]$M params, with Transformer having significantly larger KV-footprint [Logs]

is solved for as follows:

$$2BLF = \frac{2BLE}{P} + 2BD_S \tag{9}$$

Iso-FLOPs Settings: For iso-FLOPs settings, the appropriate F is solved for as follows:

$$\begin{aligned} 6BLF^2 + 4BL^2F =& 2BLE^2 \\ &+ 2BL\left(\frac{E}{H}(5HD_S + D_S) + 21D_S\right) \\ &+ \frac{4BL^2E}{P} \end{aligned} \tag{10}$$

This derivation is more verbose than Figure 2, which included simplified equations for brevity. These formulations enable comparisons across iso-KV-cache, iso-memory, and iso-FLOPs scenarios.

Iso-Activation Settings: For iso-activation settings, the appropriate F is solved for as follows:

$$4BLF = 2BLE\left(1 + \frac{1}{P}\right) + 2BD_S + BL^2H\left(\frac{1}{P} - 1\right) \tag{11}$$

Due to the $\frac{1-P}{P}$ term always being negative, and the quadratic L^2 scaling on high sequence lengths, we are unable to find an appropriate iso-activation transformer design in our budget. This is largely because Attamba significantly optimizes the L^2 attention mechanism, which reduces the activation footprint.

P	Attamba	IsoParam	IsoFLOP	IsoKV
4	512	512	160	128
8	512	512	104	64

Table 1. Setting for Transformer Baseline (Model Dimension) for IsoFLOP and IsoKV at Fixed Attamba Dimension (E=512). Calculated for Sequence Length 4096.

B.1. Baselines

Iso-KV Baseline: For iso-KV settings, the transformer model dimension is adjusted to equate the total KV-cache footprint with that of Attamba. This comparison highlights the memory savings achieved by Attamba's reduced KV-cache size, however **this baseline does not account for the L^2 attention matrix that is materialized.** In this sense, Attamba will still be significantly more efficient for long-context. For instance, at $P = 4$, Attamba achieves the same KV-Cache size, but materializes a $4\times$ smaller attention map per-head.

Iso-FLOPs Baseline: Iso-FLOPs baselines align the computational cost of the transformer with Attamba by scaling down the transformer model dimension (F) to match FLOP counts as estimated by us in Appendix A. As demonstrated in Figure 19 and Table 1, this compares the efficiency of Attamba in scenarios where computational budgets are fixed. However, this also fails to account for the KV-Cache overhead and larger attention map.

Iso-Parameter Baseline: Here, transformer baselines are chosen such that their parameter count approximately matches Attamba. This comparison does not factor in differences in KV-cache size and attention computation but offers

a straightforward view of the representational capacity of the models.

Inference efficiency strongly favors Attamba due to reduced memory bandwidth requirements, a major bottleneck in transformers. Iso-KV baselines ignore the quadratic scaling of attention maps, Iso-FLOPs and Iso-Parameter baselines do not optimize for KV-cache or activation footprint.

As shown in Figure 7, Attamba consistently outperforms Iso-FLOPs models due to its ability to compress and operate on compressed tokens effectively. It performs similarly to Iso-KV models but achieves additional gains by reducing attention map operations, which scale quadratically with sequence length. This gap widens at higher sequence lengths (e.g., $L \geq 4096$), where Iso-KV models require progressively smaller attention dimensions to match Attamba's efficiency. As is expected, we perform worse than iso-Parameter models but are significantly better on FLOPs, KV-cache size, and attention map efficiency.