PICASO: PERMUTATION-INVARIANT COMPOSITION OF DOCUMENT STATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Generation with Large Language Models (LLMs) is often augmented by incorporating additional information "in-context" through the concatenation of multiple prepended documents. However, this process leads to significant computational costs that scale with the number and size of each document chunk. State Space Models (SSMs) offer a promising solution by allowing a database of documents to be pre-processed as states from which to start the generation. However, it is infeasible to store the state corresponding to all possible combinations of multiple relevant documents. To address this challenge, we present a method called Permutation-Invariant Composition with State Space Models, or PICASO, which can compose states to efficiently approximate the effect of document concatenation. Our method can also enforce invariance to the order in which documents are presented, a desirable property when the temporal ordering of the documents is uninformative. We evaluate PICASO on WikiText and MSMARCO in both zero-shot and fine-tuned settings, and show that PICASO can match the performance of concatenation while enjoying on average $5.4\times$ speedup.

1 INTRODUCTION

Equipping large language models (LLM) with a retrieval mechanism improves their performance on generation and question answering tasks by incorporating information from large external knowledge bases. In this context, prepending retrieved sequences to the user prompt is a strong and frequently used approach for incorporating the retrieved information. This allows the generation to be conditioned on the retrieved knowledge, along with the user prompt.

However, this approach incurs a significant computational cost. Not only must the system process the user query and generate an answer, but it must also process the retrieved context, which in real-world settings can amount to many thousands of tokens. This problem is exacerbated in transformer-based models, as the inference cost of generating output tokens scales quadratically with the length of the extended context (see Figure 1).

In contrast, State Space Models (SSMs) offer a significantly faster approach. By design, SSMs encode information from arbitrary-length input sequences into a fixed-size state vector, which can then be used to condition the generation of new tokens without having to revisit the original input tokens. This suggests a very practical solution: Instead of retrieving from a database containing raw document tokens, we can create a "database of states" by pre-processing the state of each document. At inference time, generation can start directly from the retrieved state, in constant and negligible time with respect to the length of the retrieved document. This effectively eliminates the latency from having to process documents online while greatly reducing the inference time compared to transformer models. We show this in Figure 1.

However, this introduces an additional challenge. While it is possible to condition generated outputs on a single retrieved document state, in practice we often wish to condition on multiple documents at the same time. Concatenating the states of multiple documents is not straightforward, as states cannot be combined trivially. Storing states corresponding to all possible relevant combinations of documents is also not feasible.

To address this, we introduce PICASO, a method that can efficiently retrieve and combine multiple pre-computed states at inference time to condition the generation of high-quality outputs. PICASO

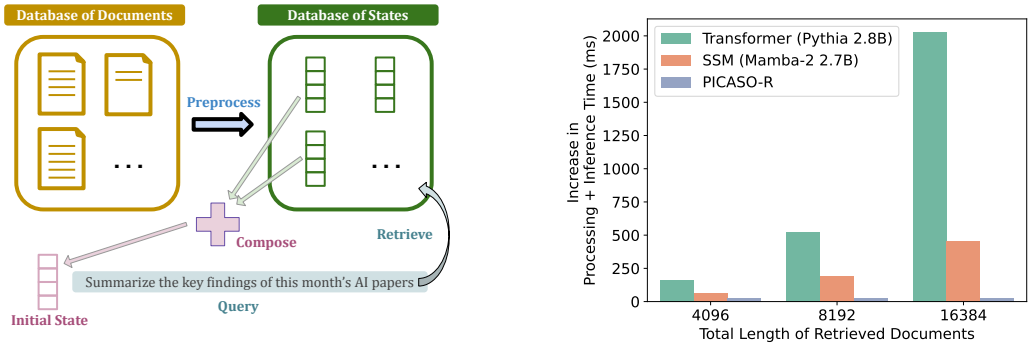

Figure 1: **(Left:)** We propose a "Database of States," where document are stored as pre-processed state vectors. Given a query, relevant states are then retrieved and composed into a single state vector which is used to condition the model's generation. **(Right:)** We plot the increase in total time required to generate an additional 64 tokens, when concatenating a 64-token prompt with retrieved documents. We model the time taken for PICASO-R as the time taken to combine 5 pre-processed document states, which involves only arithmetic operations and notably zero model processing time. As a result, the processing and inference costs for PICASO-R remain constant regardless of the length of retrieved documents. In contrast, the timings for a Transformer model scale quadratically, and for an SSM linearly, with total length when generating from concatenated document tokens. These timings are measured using the official Mamba benchmarking code, which includes optimizations such as quantization and CUDA graphs for SSMs, and flash attention for Transformers.

achieves 91% of the performance gain from combining the raw tokens of multiple documents, while offering a $5.4\times$ speed-up over concatenation.

Our approach is based on two key insights. First, we derive a simple mathematical relation from SSM dynamics that composes states in a manner that exactly equates to the result of concatenation in a single-layer model. We show that this can also be effectively applied to multi-layer SSMs to yield stronger results. Second, the order in which documents significantly impacts the resulting state, which is undesirable as the relative ordering is often uninformative. We introduce a permutation-invariant expression for composing states that eliminates these biases and significantly regularizes the composed state, improving over gains from the single best ordering by 10%.

PICASO can be applied to an off-the-shelf model without any changes. To further improve performance, we introduce a method for fine-tuning the model to better leverage the composed states for generation. Using a pre-trained Mamba-2 2.7B model, less than a day of fine-tuning on a single A100 GPU leads to the same performance as concatenation while maintaining much faster composition time on the WikiText-V2 dataset.

2 Related Work

State space models. State Space Models (SSMs) are a class of first-order differential equations specified by a set of input, output, and state variables. This ranges from Linear Time Invariant (LTI) systems, to more expressive non-linear Time Varying (Jazwinski, 2007) and Input Varying (Krener, 1975) systems, to hybrid variants such as Jump Linear and Jump-Markov Linear systems (Vidal et al., 2003a;b) which model continuous phenomena that exhibit discontinuous behavior. Among other algorithms, the Kalman Filter and Extended Kalman Filter (Kalman, 1960) are commonly used to estimate unknown state variables of linear and non-linear systems respectively from real-world measurements.

Most recently, SSMs have been used as building blocks of the architectures behind modern Foundation Models. Gu & Dao (2023) proposed Mamba, based on LIV systems, demonstrating comparable performance to Transformers (Vaswani, 2017) on language modeling while enjoying faster inference that scales linearly with input sequence length. Mamba-2 (Dao & Gu, 2024) improved computational time by implementing SSM layers with structured matrix multiplications to better leverage

specialized hardware. Lieber et al. (2024) combined SSM blocks along with global-attention blocks to create a hybrid architecture with Mixture-of-Expert layers for training larger models. To extend the ability of hybrid architectures to arbitrary long contexts, Ren et al. (2024) replaced the global attention with sliding window attention which empowered their model with long context processing ability. Zancato et al. (2024) proposed a general framework for hybrid architectures, leveraging ideas from Stochastic Realization Theory to enable model access to short-term eidetic memory "in-context", permanent structural memory "in-weights," fading memory "in-state," and long-term eidetic memory "in-storage".

Retrieval-Augmented Generation and In-Context Learning. Our work falls within the scope of In-Context Retrieval-Augmented Language Models (Ram et al., 2023), where language models are conditioned on retrieved documents via concatenation. Retrieval Augmented Generation (RAG) allows language models to leverage knowledge stored in external databases, which greatly improves performance on knowledge-intensive and domain-specific tasks (Lewis et al., 2020). In our work, we simply use a pre-trained sentence embedding model for retrieval, and we refer to Gao et al. (2023) for a detailed survey on other mechanisms. Apart from retrieval, processing (multiple) retrieved chunks can also greatly increase generation latency. Izacard et al. (2023) mitigates this by independently processes each retrieved chunk with a LLM encoder, using cross attention over the concatenated encoder outputs. Zhu et al. (2024) similarly encodes retrieved chunks in parallel, and performs decoding in a selective manner by attending only to highly relevant encoder outputs.

In-Context Learning (ICL) (Brown et al., 2020) has also emerged as an effective method to "learn" from structured contexts, or demonstrations, often modeled as a set of (query, answer) labelled pairs (Dong et al., 2022). Similar to RAG, the quality of selected demonstrations have been shown to greatly affect downstream performance (Xu et al., 2024). Several methods have been developed for selecting effective demonstrations, based on sentence embeddings (Liu et al., 2021), mutual information (Sorensen et al., 2022), perplexity (Gonen et al., 2022), and even BM25 (Robertson et al., 2009). Similar to the motivation of our work, several studies have shown that the performance of ICL is heavily dependent on demonstration ordering. Zhao et al. (2021) shows that answers positioned towards the end of the prompt are more likely to be predicted, while Lu et al. (2021) shows that results can vary wildly between random guess and state-of-the-art depending on the order that demonstrations are presented. Outside of ICL, Liu et al. (2024) further shows that language models do not robustly utilize information in long input contexts due to sensitivity to positioning.

Model and State Composition. Our work falls into the category of composing of deep models, representations, and states. Wortsman et al. (2022) proposes Model Soup, which composes multiple non-linearly fine-tuned models via averaging model weights. Liu & Soatto (2023); Liu et al. (2023) leverages model linearization to enforce an equivalence between weight averaging and output ensembling. Perera et al. (2023) independently learns task-specific prompts which can be linearly averaged to yield new prompts for composite tasks. For SSMs, Pióro et al. (2024) investigates averaging of states, along with decay-weighted mixing which is closely related to a baseline version of our method, CASO. However, the equations described in their work differ from CASO, and their evaluations are limited to composition of two equal-length documents. In contrast, our method greatly improves upon CASO by incorporating permutation invariance, which we show is important to achieve performances comparable to that of concatenation.

3 METHOD

3.1 PRELIMINARIES:

A linear input-dependent discrete-time model has the form

$$\begin{cases} x_t = A(u_t)x_{t-1} + B(u_t)u_t \\ y_t = C(u_t)x_t + Du_t. \end{cases} \tag{1}$$

Here $x_t \in \mathbb{R}^m$ is the *state* at time t, while $u_t, y_t \in \mathbb{R}^d$ are the *input* and the *output* respectively. The matrices $A(u_t) \in \mathbb{R}^{m\times m}, B(u_t) \in \mathbb{R}^{m\times d}, C(u_t) \in \mathbb{R}^{d\times m}$ (which are input-dependent) and $D \in \mathbb{R}^{d\times d}$ are learnable parameters.

Unrolling the first equation, we obtain

$$\begin{aligned} x_t &= A(u_t)\cdots A(u_1)x_0 + \sum_{\tau=0}^{t-1} A(u_t)\cdots A(u_{t-\tau+1})B(u_{t-\tau})u_{t-\tau} \\ &= A(\boldsymbol{u})x_0 + x(\boldsymbol{u}), \end{aligned} \tag{2}$$

where $\boldsymbol{u} = (u_1, \ldots, u_t)$ denotes the *sequence* of inputs, $A(\boldsymbol{u}) = A(u_t)\cdots A(u_1)$ is the accumulated decay matrix and $x(\boldsymbol{u}) = \sum_{\tau=0}^{t-1} A(u_t)\cdots A(u_{t-\tau+1})B(u_{t-\tau})u_{t-\tau}$ is the accumulated input signal. Since this coincides with x_t when $x_0 = 0$, we refer to it as the *state* for input sequence $\boldsymbol{u}$.

In the following, we write $\mathcal{V} \subset \mathbb{R}^d$ for a finite set of token embeddings and $\mathcal{V}^* = \bigcup_{n\geq 0} \mathcal{V}^n$ for the set of variable-length sequences of token embeddings. We view a State Space Language Model (SSM) as a map $f_\theta : \mathcal{V}^* \times \mathbb{R}^m \mapsto \mathbb{P}(\mathcal{V})$ with parameters θ which takes in as input an initial state $x \in \mathbb{R}^m$ and token embedding sequence $\boldsymbol{u} \in \mathcal{V}^*$, and returns a distribution over $\mathcal{V}$. Modern SSMs (Gu & Dao, 2023; Zancato et al., 2024) usually contain multiple Selective State Space stacked layers as in equation 1. In a multi-layer setting, we write $x(\boldsymbol{u})$ and $A(\boldsymbol{u})$ for the sequence of states and decay matrices corresponding to all layers.

3.2 Database of States

By the Markov property, the state of an SSM makes the past independent of the future. In other words, $f_\theta(\boldsymbol{u}\cdot\boldsymbol{u}', 0) = f_\theta(\boldsymbol{u}, x(\boldsymbol{u}'))$ for all $\boldsymbol{u}, \boldsymbol{u}' \in \mathcal{V}^*$, where $\cdot$ denotes concatenation. In practice, this means that a SSM model can equivalently be initialized with the state arising from an (variable-length) input sequence, instead of the input sequence itself. This is akin to the KV-cache of Transformer architectures, except that the dimension of the state is fixed regardless of sequence length.

In several real-world use cases such as Retrieval Augmented Generation, document chunks are commonly obtained or retrieved from a database (Borgeaud et al., 2022). Instead of storing them in the database as raw text or tokens, we propose to use a "database of states," where we pre-process each document chunk and store their states. The advantage here is self-evident when conditioning on a single chunk, since we can initialize the SSM with the retrieved pre-processed state instead of having to process the entire chunk online. However this poses a problem when attempting to compose multiple document chunks, since we do not know how to compose their states. We will show how this is tackled with our proposed method.

3.3 PICASO: Permutation-Invariant Composition with State Space Models

Given a query and a collection of relevant document chunks, an easy method to compose them is to simply concatenate all chunks with the query into a single sequence to feed into the SSM. Recall that this, however, presents two key limitations. Before even a single token continuation can be generated from the query, the entire sequence of concatenated chunks has to be processed sequentially, which can be computationally intensive when document chunks are long or numerous (Figure 1). Another limitation is having to select the order of document concatenation when prompting the model, for which there might be no natural way of doing so without a powerful scoring mechanism.

To address the first limitation, we propose a first version of our method, **C**omposition with St**a**te **S**pace M**o**dels (CASO), which works by modeling sequence concatenation with state composition based on the dynamics of a single-layer SSM.

Proposition 1 (CASO)**.** *Let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$ be a collection of input sequences and let $\boldsymbol{u} = \boldsymbol{u}_1 \cdots \boldsymbol{u}_n$ be their concatenation. Then, for a SSM layer that evolves based on equation 1, we have*

$$x(\boldsymbol{u}) = x(\boldsymbol{u}_n) + \sum_{i=1}^{n-1} A(\boldsymbol{u}_n)\cdots A(\boldsymbol{u}_{i+1})\cdot x(\boldsymbol{u}_i) \tag{3}$$

We can see this by recursively applying equation 2 on $x(\boldsymbol{u}) = A(\boldsymbol{u}_n)x(\boldsymbol{u}_1 \cdots \boldsymbol{u}_{n-1}) + x(\boldsymbol{u}_n)$.

Given a collection of document chunks $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n$, CASO simply approximates the dynamics of multi-layer SSMs, for which Proposition 1 does not hold exactly, via $x_\theta^{\mathrm{CASO}}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) =$

$x(\boldsymbol{u}_n) + \sum_{i=1}^{n-1} A(\boldsymbol{u}_n) \cdots A(\boldsymbol{u}_{i+1}) \cdot x(\boldsymbol{u}_i)$. We then load $x_\theta^{\mathrm{CASO}}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$ as the initial state of the model to infer continuations from the given query. We note that in Mamba-style models, the matrices $A(\cdot)$ are diagonal. As such, computing CASO requires only simple element-wise arithmetic operations and importantly zero model computation time (*i.e.* zero forward passes required).

However, since each state is weighted by the decay factors of future document chunks, this composition operation is still very much order-dependent. We propose to introduce permutation-variance by considering a group of permutations $G \subseteq S_n$, where S_n denotes the symmetric group of n elements, using which we define our method, PICASO (**P**ermutation-**I**nvariant CASO):

$$x^{\mathrm{PICASO}}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) := \frac{1}{|G|} \sum_{\pi \in G} x^{\mathrm{CASO}}(\boldsymbol{u}_{\pi(1)}, \ldots, \boldsymbol{u}_{\pi(n)}) \tag{4}$$

For any group G, by expansion of the CASO terms and collecting common factors, this can be written as a linear combination of individual document chunk states $x(\boldsymbol{u}_i)$:

$$x^{\mathrm{PICASO}}(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) = \sum_{i=1}^{n} W_i(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) x(\boldsymbol{u}_i)$$

with weights W_i depending on $A(\boldsymbol{u}_1), \ldots, A(\boldsymbol{u}_n)$. In this work we are particularly concerned with two cases: the full symmetric group $G = S_n$, which includes all possible permutations, and the cyclic group $G = C_n$, which consists of rotations of the sequence. We will refer to them as PICASO-S and PICASO-R respectively.

While they appear computationally infeasible at first glance, since PICASO-S and PICASO-R average over $n!$ and n CASO states respectively, each of which is itself a composition of n document states, the following propositions show that they can actually be computed in polynomial and linear time respectively for modern SSM models with diagonal A matrices.

Proposition 2. *Assume $G = S_n$ and that the matrices $A(\boldsymbol{u}_i)$ commute with each other (e.g., are diagonal). Using shorthand notations $A_i := A(\boldsymbol{u}_i)$ and $W_k := W_k(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$ we have*

$$\begin{aligned} W_k &= \frac{1}{n!} \left[(n-1)! + (n-2)! \cdot 1! \cdot \sum_{\substack{1 \le i_1 \le n \\ i_1 \neq k}} A_{i_1} + (n-3)! \cdot 2! \cdot \sum_{\substack{1 \le i_1 < i_2 \le n \\ i_1, i_2 \neq k}} A_{i_1} A_{i_2} + \ldots \right] \\ &= \frac{1}{n} \sum_{m=0}^{n-1} \frac{1}{\binom{n-1}{m}} \cdot e_m(A_1, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n), \end{aligned}$$

where

$$e_m(A_1, \cdots A_{n-1}) := \sum_{1 \le i_1 < i_2 < \cdots < i_m \le n-1} A_{i_1} \cdots A_{i_m}$$

is the m-th elementary symmetric polynomial *(Macdonald, 1998) (in the matrices A_i).*

Elementary symmetric polynomials satisfy the recursive relation

$$e_m(A_1, \ldots, A_{n-1}) = A_{n-1} e_{m-1}(A_1, \ldots, A_{n-2}) + e_m(A_1, \ldots, A_{n-2}).$$

Using this relation, we can compute all values of e_m, and hence the coefficients W_k, using $O(n^2)$ operations via dynamic programming. We detail the implementation in Algorithm 1 of the Appendix. Consequently, the full state from PICASO-S can be efficiently computed in polynomial $\mathcal{O}(n^3)$ time, which we show in the experiments to still be faster than processing document concatenations even for n as large as 10.

Next, we similarly show that the coefficients for PICASO-R can be efficiently computed by exploiting invertibility of the matrices $A(\boldsymbol{u}_i)$.

Proposition 3. *Assume $G = C_n$ (cyclic permutations). Then writing $A_i := A(\boldsymbol{u}_i)$ and $W_k := W_k(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$ we have*

$$W_k = \frac{1}{n} \left[\mathrm{Id} + \sum_{m=1}^{n-1} A_{[k+m]_n} \cdots A_{[k+1]_n} \right].$$

where Id *is the identity matrix, and* $[i]_n$ *denotes* $i \bmod n$*. Assuming that the matrices* A_i *are invertible, these can be computed efficiently by setting*

$$\bar{A}_i = \begin{cases} A_{[i]_n} \cdots A_1 \cdot A_n \cdots A_1 & i > n \\ A_i \cdots A_1 & i \leq n \end{cases}, \quad \bar{B}_i = \bar{A}_1 + \ldots + \bar{A}_{i-1}, \quad W_k = \frac{1}{n}[\bar{A}_k^{-1}(\bar{B}_{k+n} - \bar{B}_k)],$$

for $i = 1, \ldots, 2n$*, and* $k = 1, \ldots, n$*.*

We detail in Algorithm 3 in the Appendix our efficient implementation for computing PICASO-R in $\mathcal{O}(n)$ time complexity via cumulative sums and products. Evidently, PICASO-R is significantly faster than PICASO-S while trading off exact permutation invariance for invariance only to cyclic permutations of the original order. We will show that the difference in empirical performance between PICASO-S to PICASO-R is negligible, as such PICASO-S can almost always be replaced with its much faster variant PICASO-R.

We remark that the property of permutation-invariance can also be applied to naive concatenation (as opposed to CASO). This is achieved simply by concatenating documents in various different orders, followed by taking an average of their resulting states. While performing this for the symmetric group S_n is computationally infeasible, we can similarly restrict our permutation set to C_n. We term this variant Permutation-Invariant Concatenation (PIConcat-R), where $-R$ denotes invariance to the set of cyclic permutations. We note that the *model* computational costs (forward passes) of this method still scales quadratically with number of documents (compared to linear scaling of regular concatenation), as such we include it only for completeness.

As a final technicality, we note that for Mamba-style SSM models, we additionally require storing the last m_{conv} (usually $m_{conv} = 4$) input tokens of each SSM layer to ensure that the state is sufficient for generating the same distributions over continuations as the input sequence. We perform simple averaging to combine these tokens from different documents which we show to work well empirically; more sophisticated methods could be explored in future work.

4 Why PICASO's average works

While the combination of state expression for CASO is directly motivated by the dynamics of the system, there is no a priori reason why averaging permuted CASO states should perform well. In Figure 3 we show that averaging both independent states and CASO states can perform better than using any individual state. This suggests a emergent/learned algebraic structure on the space of states such that linear combination of states combine their information to some degree.

In our empirical results below, we show that averaging all individual states (which would also be a permtuation invariant solution) performs significantly weaker than avearging CASO states (as PICASO does) We believe that this is because the approximate linear structure of the state space is only valid locally. The combined states are naturally closer together than the independent states, hence able to better exploit the local linearity. We show this in the following proposition:

Proposition 4. *Consider a single-layer SSM parametrized by* θ*, and two input sequences* $\boldsymbol{u}$ *and* $\boldsymbol{u}'$*. Then, the Euclidean distance between the states can be bounded via*

$$\|x^{CASO}(\boldsymbol{u}, \boldsymbol{u}') - x^{CASO}(\boldsymbol{u}', \boldsymbol{u})\|_2^2 \leq \|(I - A(\boldsymbol{u}'))x(\boldsymbol{u})\|_2^2 + \|(I - A(\boldsymbol{u}))x(\boldsymbol{u}')\|_2^2$$

To see this, simply apply the triangle inequality on the following obtained via substituting the equations for CASO:

$$\begin{aligned} \|x^{CASO}(\boldsymbol{u}, \boldsymbol{u}') - x^{CASO}(\boldsymbol{u}', \boldsymbol{u})\|_2^2 &= \|A(\boldsymbol{u}')x(\boldsymbol{u}) + x(\boldsymbol{u}') - (A(\boldsymbol{u})x(\boldsymbol{u}') + x(\boldsymbol{u}))\|_2^2 \\ &= \|(A(\boldsymbol{u}') - I)x(\boldsymbol{u}) + (I - A(\boldsymbol{u}))x(\boldsymbol{u}')\|_2^2 \end{aligned}$$

As a special case, we observe that as the decay factor approaches identity, the distance between two CASO states approaches zero. In Figure 2, we visualize naive averaging of the states arising from 3 retrieved document chunks, and averaging of CASO states resulting from each cyclic permutation of these chunks. We use WikiText-v2 as described in the Experiments for these plots. Indeed, we observe that CASO states are much closer to one another in the resulting loss landscape.

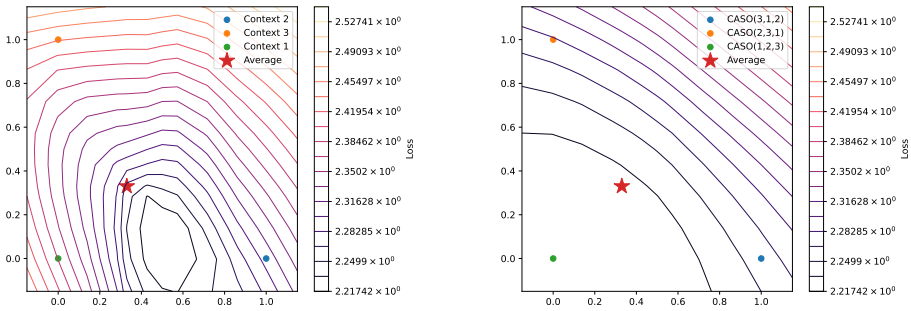

Figure 2: **Left:** Naive averaging ("Soup") of chunk states. **Right:** Averaging CASO states. CASO states are "closer" to one another (see Proposition) and hence can be more meaningfully interpolated. On the other hand, naively averaging states of independent chunks do not possess this property. Both plots are computed over 10 samples of (query, continuation, retrieved chunks).

5 Learning to use composed states

As previously noted, in practice, for SSM models consisting of multiple state-space blocks stacked with temporal convolutions, $x(\boldsymbol{u})$ in equation 3 will not be exactly the state arising from a concatenated list of inputs. In this section, we introduce a fine-tuning objective to enable SSMs to better leverage composed states. Let $\mathcal{D} = \{(\boldsymbol{u}_i, u_i, S_i)\}_{i=1}^{N}$ be a dataset of sequences $\boldsymbol{u}_i$, their next-token continuation u_i, and a collection (in some particular order) of document chunks S_i retrieved from a database using $\boldsymbol{u}_i$. We minimize the prediction loss over the continuation, given a (composed) initial state and the query sequence:

$$\mathcal{L}_{BPTC}(\theta) = \sum_{(\boldsymbol{u}_i, u_i, S_i) \in \mathcal{D}} L_{\text{CE}}(f_\theta(\boldsymbol{u}_i, x^{\text{PICASO}}(S_i)), u_i),$$

where $L_{\text{CE}}(\cdot, \cdot)$ is the cross-entropy loss.

We denote this learning objective Backpropagation Through Composition (BPTC), where gradients are propagated through the state composition process x^{PICASO}. To reduce training time, we also consider an alternative version where we do not backpropagate through the composition step, which we denote Backpropagation To Composition (BP2C):

$$\mathcal{L}_{BP2C}(\theta) = \sum_{(\boldsymbol{u}_i, u_i, S_i) \in \mathcal{D}} L_{\text{CE}}(f_\theta(\boldsymbol{u}_i, \text{sg}\left[x^{\text{PICASO}}(S_i)\right]), u_i),$$

where sg denotes the stop-gradient operator. We will show that when used for fine-tuning, this learning objective greatly improves the model's ability to leverage composed states for generation to the level of the concatenation albeit with much faster speeds, while maintaining performance on standard LLM evaluation tasks.

6 Experiments

6.1 Implementation Details

We run our main experiments on the largest available SSM on Huggingface - Mamba-2 2.7B (Dao & Gu, 2024). We evaluate our method on two large-scale datasets - WikiText-V2 (Merity et al., 2016) and MSMARCO (Nguyen et al., 2016). We use the training splits as our fine-tuning data, and the testing/validation splits respectively for evaluation. To pre-process WikiText-V2 for our use case, we split each passage in the dataset into two equal chunks, with the goal of predicting the second (continuation) from the first (query). The retrieval database comprises all remaining chunks, from which we retrieve via an external sentence embedding model, All-MiniLM-L6-v2[1]. In most experiments, we retrieve up to 10 chunks, since improvements appears to saturate beyond

[1]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

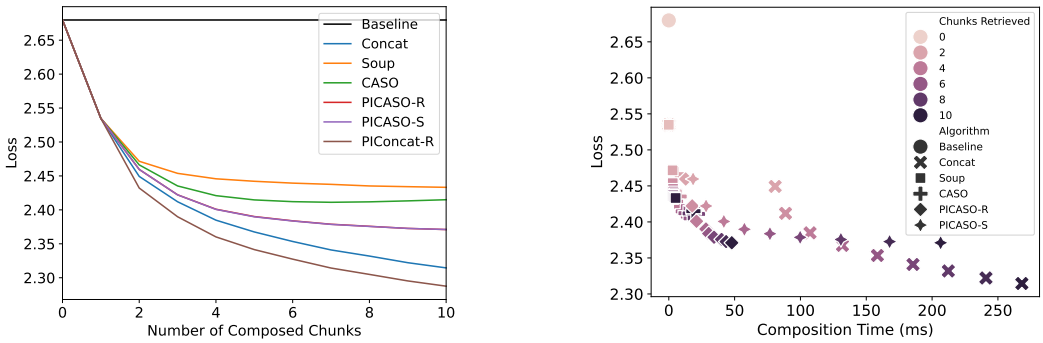

Figure 3: Zero-shot evaluation of PICASO using Mamba-2 compared to other composition methods on WikiText. While the performance of PICASO lags slightly behind that of concatenation (left), PICASO-R is on average $5.4\times$ faster (right). PICASO-S and PICASO-R perform similarly and yield overlapping curves (hence not visible in the left plot). Incorporating permutation invariance for concatenation via PIConcat-R gives the best results. However, it incurs magnitudes higher computational costs despite being performed within a single batched forward pass, hence we omit from the right plot to prevent it from disrupting the scale of the x-axis and focus comparisons on PICASO.

that, and loss from concatenation blows up as a result of exceeding training context length (Figure 6, Appendix). We pre-process MSMARCO by filtering only entries with well-formed answers and discarding those without relevant passages.

We used the official benchmark[2] with an A100 GPU for our timing experiments in Figure 1 to ensure fairest comparisons. For the rest of the experiments, we run the model in full-precision, and evaluate performance of the model starting from a custom initial state, a feature not supported by the official benchmark at the time of writing, as such timings differ.

For fine-tuning experiments using BPTC and BP2C, we base our implementation on the official HuggingFace [3] trainer with default hyperparameters, and retrieve the k most relevant chunks for each query sample for composition. For WikiText, we select $k \in \{0, \ldots, 10\}$ uniformly at random for each batch. For MSMARCO, we use all the available passages (both relevant and irrelevant) associated with each training example. For both datasets, we fine-tune for only 1 epoch. In all fine-tuning experiments, we ensure the training set (both the examples and the chunk database) are disjoint from the validation set to ensure fair evaluation.

6.2 Comparison Models

We compare inference accuracy (measured by log-perplexity) and processing latency of PICASO with its order-dependent version, CASO, in addition to the following methods:

Baseline: Loss of the model on the test sample without using any contextual information.

Concatenation (Ram et al., 2023): We concatenate individual chunks based on a specific ordering. For WikiText-V2 experiments, we consider the "best-case ordering" as determined by the sentence embedding model where more relevant documents are closer to the query (at the end). We initialize the model with the state of the earliest document in the concatenation, which we assume to be available via pre-processing, and recompute the composed state from only the remaining ones.

Soup (Pióro et al., 2024): Simple averaging of states obtained from each document.

6.3 Main Results

In this section, we evaluate both the zero-shot and fine-tuned performance of PICASO in Section 6.3.1 and Section 6.3.2 respectively, and show in Section 6.3.3 that the fine-tuned model does

[2]https://github.com/state-spaces/mamba

[3]https://github.com/huggingface/transformers

not overfit to the composition task. We also include additional experiments showing that LLM capabilities are not impacted by fine-tuning in Appendix B.4, and show that PICASO can also be used for data attribution in Appendix C

6.3.1 ZERO-SHOT PERFORMANCE

We demonstrate in Figure 3 that applying PICASO-R in a zero-shot manner on WikiText-V2 greatly improves performance over the baseline by an average of 10.1% across 1-10 retrieved document chunks. This greatly improves over Soup (8.5%) and CASO (9.2%). Compared to concatenation (11.1%), PICASO-R performs slightly worse but benefits from magnitudes improvement in processing time on an average of $5.4\times$. In this task, PICASO-R achieves almost exactly the same performance as PICASO-S, but with a much faster composition time. As a sanity check for motivation for our method, we show that PIConcat achieves the best performance (12.0%) overall, but at the cost of significantly greater computational time despite our batched-inference implementation.

In Row 1 of Table 1, we show that applying PICASO-R and PICASO-S in a zero-shot manner on MSMARCO similarly yields considerable improvements (37.2%) over the naive baseline, while achieving performance close to that of concatenation (41.3%).

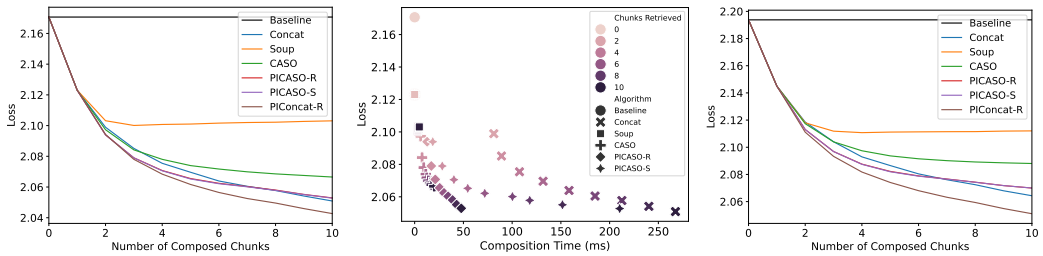

Figure 4: **(Left + Middle:)** Fine-tuning with BPTC on WikiText brings the performance of PICASO to that of concatenation, while retaining its significant speed advantages. **(Right:)** Fine-tuning with BP2C on WikiText improves the effectiveness of PICASO as well, but is much faster in terms of training time since it does not require backpropagating through the composed state. Note that fine-tuning has no impact on the actual composition time when used for inference.

6.3.2 BACKPROPAGATION THROUGH AND TO COMPOSITION

While PICASO demonstrates strong performance in the zero-shot setting, PICASO still lags behind concatenation in terms of prediction accuracy. We attribute this to composed states being "out-of-distribution" for the model, since these states do not arise from any sequence of input tokens. In this section, we test if this can be resolved via fine-tuning with PICASO-R composed states via BPTC and BP2C. Indeed, as we show in Figure 4, BPTC and BP2C greatly improves the performance of PICASO-R and PICASO-S to that similar to concatenation, while maintaining much faster processing timings on WikiText. Similarly, we show in Rows 4-5 of Table 1 that fine-tuning on the MSMARCO training set also levels the performance of PICASO with that of concatenation. We also note that while BP2C is significantly faster in terms of training time, it incurs a small performance trade-off compared to BPTC for both datasets, keeping number of training iterations constant.

6.3.3 EVALUATION OF FINE-TUNED MODEL ON OTHER DIFFERENT TASKS

We showed that models fine-tuned on a specific downstream task (training set) using BPTC/BP2C can perform strongly when composing samples drawn from a similar distribution (test set). We further show in Table 1 that models fine-tuned on one domain (WikiText) can demonstrate small performance gains (or at the very least, no performance loss) when composing samples via PICASO on another domain (MSMARCO). Finally, we show in Appendix B.4 that fine-tuning models with BP2C/BPTC maintain (and occasionally even improve) performance on general LLM evaluation tasks compared against the original model.

Table 1: All models in this table are evaluated on the MSMARCO validation set. We evaluate performance of models fine-tuned via BPTC/BP2C on both the WikiText and MSMARCO training sets. Rows 2-3 show that fine-tuning models to compose WikiText chunks does not harm performance when evaluated on composing document chunks from MSMARCO. When composing chunks from distributions similar to those encountered during training (Rows 4-5), PICASO matches the performance of concatenation while being magnitudes faster.

	Naive	Concat	Soup	CASO	PICASO-R	PICASO-S
Mamba2-2.7B Base	2.42	1.42	2.04	1.56	1.52	1.52
Mamba2-2.7B BP2C-WikiText	2.44	1.44	2.07	1.53	1.50	1.50
Mamba2-2.7B BPTC-WikiText	2.43	1.46	2.08	1.53	1.49	1.49
Mamba2-2.7B BP2C-MSMARCO	1.85	0.68	1.27	0.72	0.69	0.69
Mamba2-2.7B BPTC-MSMARCO	1.79	0.65	1.20	0.68	0.65	0.65

7 Limitations and Discussion

We have proposed a method, PICASO, that enables efficient retrieval and composition of document chunks by pre-processing their individual states. Without any training, ur approach can handle the composition of information contained in up to 10 documents in a manner that is order-invariant. PICASO notably requires zero online model processing time, since generation can begin directly from the composed states. When models are further fine-tuned with our proposed learning objective, states composed using PICASO perform comparably to those produced from the concatenation of document tokens, while offering on average a $5.4\times$ faster composition time.

Nevertheless, our method does have some limitations. When applied in a zero-shot manner, PICASO still lags slightly behind concatenation in terms of prediction accuracy. PICASO is also currently limited to architectures based on SSM layers. We leave as future work extension of PICASO towards recently popularized attention-based hybrid models, which require more sophisticated methods of composing key-value caches. Lastly, we also leave as future work the exploration of parameter-efficient fine-tuning methods such as adapters, which can be used to augment the model at inference time to enable state composition while preserving the original model's behavior.

References

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pp. 2206–2240. PMLR, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework

for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

Hila Gonen, Srini Iyer, Terra Blevins, Noah A Smith, and Luke Zettlemoyer. Demystifying prompts in language models via perplexity estimation. *arXiv preprint arXiv:2212.04037*, 2022.

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

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251): 1–43, 2023.

Andrew H Jazwinski. *Stochastic processes and filtering theory*. Courier Corporation, 2007.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

Arthur J Krener. Bilinear and nonlinear realizations of input-output maps. *SIAM Journal on Control*, 13(4):827–834, 1975.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33: 9459–9474, 2020.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

Tian Yu Liu and Stefano Soatto. Tangent model composition for ensembling and continual fine-tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 18676–18686, 2023.

Tian Yu Liu, Aditya Golatkar, and Stefano Soatto. Tangent transformers for composition, privacy and removal. *arXiv preprint arXiv:2307.08122*, 2023.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*, 2021.

Ian Grant Macdonald. *Symmetric functions and Hall polynomials*. Oxford university press, 1998.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016. URL http://arxiv.org/abs/1611.09268.

Pramuditha Perera, Matthew Trager, Luca Zancato, Alessandro Achille, and Stefano Soatto. Prompt algebra for task composition. *arXiv preprint arXiv:2306.00310*, 2023.

Maciej Pióro, Maciej Wołczyk, Razvan Pascanu, Johannes von Oswald, and João Sacramento. State soup: In-context skill learning, retrieval and mixing. *arXiv preprint arXiv:2406.08423*, 2024.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL http://arxiv.org/abs/1908.10084.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*, 2024.

Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Taylor Sorensen, Joshua Robinson, Christopher Michael Rytting, Alexander Glenn Shaw, Kyle Jeffrey Rogers, Alexia Pauline Delorey, Mahmoud Khalil, Nancy Fulda, and David Wingate. An information-theoretic approach to prompt engineering without ground truth labels. *arXiv preprint arXiv:2203.11364*, 2022.

Ashish Vaswani. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

René Vidal, Alessandro Chiuso, Stefano Soatto, and Shankar Sastry. Observability of linear hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pp. 526–539. Springer, 2003a.

René Vidal, Stefano Soatto, Yi Ma, and Shankar Sastry. An algebraic geometric approach to the identification of a class of linear hybrid systems. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 1, pp. 167–172. IEEE, 2003b.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pp. 23965–23998. PMLR, 2022.

Xin Xu, Yue Liu, Panupong Pasupat, Mehran Kazemi, et al. In-context learning with retrieved demonstrations for language models: A survey. *arXiv preprint arXiv:2401.11624*, 2024.

Luca Zancato, Arjun Seshadri, Yonatan Dukler, Aditya Golatkar, Yantao Shen, Benjamin Bowman, Matthew Trager, Alessandro Achille, and Stefano Soatto. B'mojo: Hybrid state space realizations of foundation models with eidetic and fading memory. *arXiv preprint arXiv:2407.06324*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pp. 12697–12706. PMLR, 2021.

Yun Zhu, Jia-Chen Gu, Caitlin Sikora, Ho Ko, Yinxiao Liu, Chu-Cheng Lin, Lei Shu, Liangchen Luo, Lei Meng, Bang Liu, et al. Accelerating inference of retrieval-augmented generation via sparse context selection. *arXiv preprint arXiv:2405.16178*, 2024.

A ALGORITHMS: PICASO-S AND PICASO-R

We show in Algorithm 1 how PICASO-S is computed in polynomial time via a dynamic programming approach based on Algorithm 2. In Algorithm 3, we also show how PICASO-R can be computed with linear time complexity. Time complexity is measured as the number of arithmetic operations required as a function of number of document states.

Algorithm 1 PICASO-S- $\mathcal{O}(n^3)$

Require: States $x = \{x_i\}_{i=0}^{n-1}$, Decays $A = \{A_i\}_{i=0}^{n-1}$
  return $\sum_{i=0}^{n-1}$ `PICASO-S-DP`$(A_{-i}) \cdot x_i$
  ▷ A_{-i} denotes all elements of A except A_i

Algorithm 2 PICASO-S-DP - $\mathcal{O}(n^2)$

Require: Decays $A = \{A_i\}_{i=0}^{n-1}$
  DP[:,:] $\leftarrow$ zeros(n, n)
  DP[0,:] $\leftarrow 1$
  $w \leftarrow 0$
  for $i = 1, \ldots, n-1$ **do**
    for $j = i, \ldots, n-1$ **do**
      DP$[i][j] \leftarrow$ DP$[i][j-1] + A_{j-1} \cdot$ DP$[i-1][j-1]$
    end for
  end for
  for $i = 0, \ldots, n-1$ **do**
    $w \leftarrow w + \frac{1}{n \cdot \binom{n-1}{i}} \cdot$ DP$[i][n-1]$
  end for
  return w

Algorithm 3 PICASO-R - $\mathcal{O}(n)$

Require: States $x = \{x_i\}_{i=0}^{n-1}$, Decays $A = \{A_i\}_{i=0}^{n-1}$
  $\hat{x} \leftarrow 0, \quad \hat{A} \leftarrow [A_1, \ldots, A_n, A_1, \ldots, A_n]$
  $\hat{A}_{\Pi} = \text{cumprod}(\hat{A})$
  $\hat{A}_{\Sigma\Pi} = \text{cumsum}(\hat{A_{\Pi}})$
  for $i = 1, \ldots, n-1$ **do**
    $w_i \leftarrow \frac{1}{n} \cdot \left(\left(\hat{A}_{\Sigma\Pi}[n+i-1] - \hat{A}_{\Sigma\Pi}[i] \right) \hat{A}_{\Pi}[i]^{-1} + 1 \right)$
    $\hat{x} \leftarrow \hat{x} + w_i \cdot x_i$
  end for
  return $\hat{x}$

B FURTHER ANALYSIS

B.1 COMPUTATIONAL COSTS OF PICONCAT

In Figure 5, we visualize the computational costs incurred by PIConcat, which we show to dominate that of other methods despite resulting in the best performance on the WikiText dataset.

B.2 SCALING BEYOND EFFECTIVE CONTEXT LENGTH

In Figure 6, we show that as the total length of retrieved documents scale beyond a certain threshold (effective context size of the model), the loss from concatenation blows up and rapidly increases beyond the no-retrieval baseline. On the other hand, performance of PICASO remains stronger than that of the baseline when composing 50 document chunks.

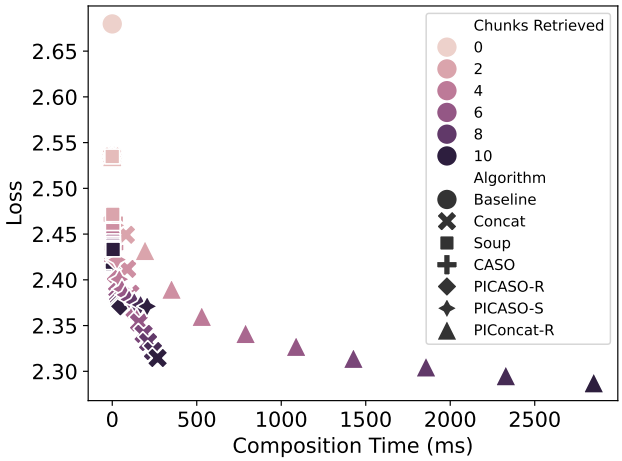

Figure 5: Timings for different composition algorithms evaluated on WikiText using Mamba-2 2.7B (zero-shot), including that of PIConcat-R. While PIConcat results in the best performance (y-axis), its computational cost (x-axis) is significantly higher than that of other methods. We refer to Figure 3 for a more condensed plot to compare the remaining methods.

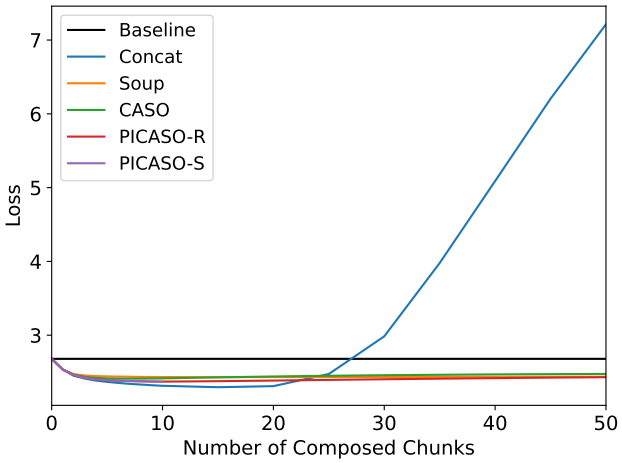

Figure 6: Concatenation scales poorly with total size of retrieved documents beyond training context length. PICASO yields greater stability even composing up to 50 document chunks retrieved from WikiText.

B.3 Inference vs Processing Time

In Figure 7, we show that the context processing time for Mamba-2 comprises a significant proportion of the total generation time. For large sequence lengths beyond 6K tokens, the processing time even dominates the inference time for generating 32 tokens.

B.4 Performance on LLM Evaluation Tasks

In Table 2, we show that fine-tuning Mamba2-2.7B with BTPC/BP2C objectives do not degrade existing LLM capabilities when evaluated on several LLM evaluation benchmarks - HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), ARC-E, ARC-C (Clark et al., 2018), WinoGrande (Sakaguchi et al., 2021), and OpenbookQA (Mihaylov et al., 2018).

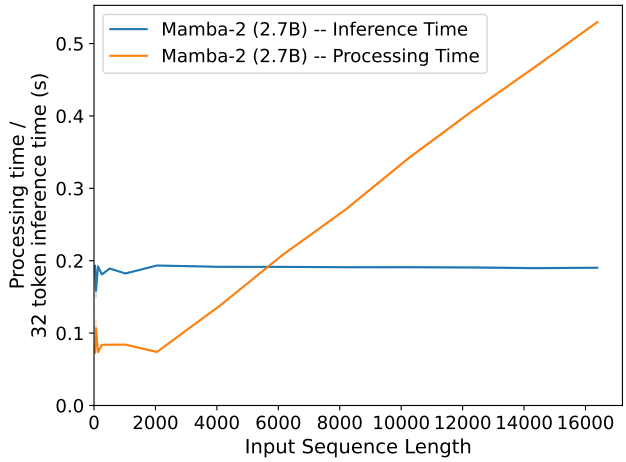

Figure 7: Mamba-2 Processing vs Inference Time of 32 tokens. Processing time (orange) occupies a significant proportion of the total time taken to generate from an input sequence, even dominating the constant inference time from the processed state (blue) as number of tokens in the input grows.

Table 2: Evaluation of Mamba2-2.7B trained with BPTC and BP2C on LLM evaluation tasks. Here, we show that fine-tuning for composition does not degrade existing LLM capabilities. In this table, we report the length-normalized accuracy for each task.

	HellaSwag	PIQA	ARC-E	ARC-C	WinoGrande	OpenbookQA
Mamba2-2.7B Base	66.6 ± 0.5	76.3 ± 1.0	64.8 ± 1.0	36.3 ± 1.4	63.9 ± 1.4	38.8 ± 2.2
BP2C-WikiText	66.7 ± 0.5	76.3 ± 1.0	64.9 ± 1.0	37.5 ± 1.4	63.6 ± 1.4	39.8 ± 2.2
BPTC-WikiText	66.7 ± 0.5	75.6 ± 1.0	64.9 ± 1.0	37.2 ± 1.4	63.2 ± 1.4	40.2 ± 2.2

B.5 Ablation on Choice of Retriever

In Figure 8, we ablate the impact of difference retriever choices on PICASO-R. In particular, we evaluate the performance of PICASO-R on WikiText when using the following embedding models from Sentence-Transformers (Reimers & Gurevych, 2019): average_word_embeddings_glove.6B.300d, all-MiniLM-L6-v2, and all-mpnet-base-v2, arranged in increasing order of performance on 14 different sentence embedding tasks (Reimers & Gurevych, 2019). As expected, Figure 8 shows that the performance of PICASO-R highly correlates with the strength of the retriever, where stronger retrievers yields better results on WikiText.

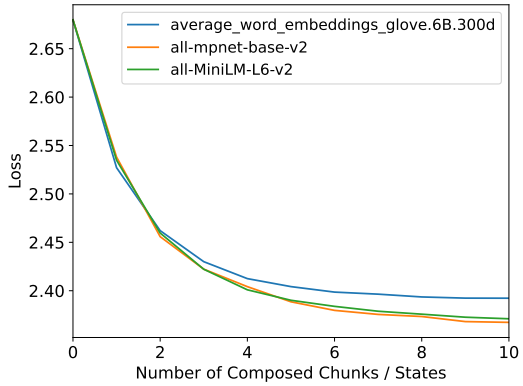

Figure 8: Ablation study on how choice of retriever model impacts performance of PICASO-R on WikiText. As expected, stronger retriever models result in better downstream performance.

B.6 Evaluation on Multiple Choice Tasks

In this section, we evaluate PICASO-R on the OpenbookQA (Mihaylov et al., 2018) multiple-choice task, where we retrieve from a document database of full passages from WikiText-V2. While OpenbookQA provides the ground truth fact for each evaluation sample, we discard this in our evaluations following standard practice in Gao et al. (2024). We leverage the same retrieval model used for the main WikiText experiments. Table 3 shows that PICASO-R achieves performance close to concatenation, with a $8\times$ speed-up in composition time.

Naive		Concat		PICASO-R	
Acc (↑)	Time (↓)	Acc (↑)	Time (↓)	Acc (↑)	Time (↓)
38.8%	NA	40.0%	233 ms	39.9%	29 ms

Table 3: Evaluation on OpenbookQA dataset, augmented with retrieved passages from WikiText. We use normalized accuracy as our evaluation metric, and report the time taken to compose retrieved passages. Numbers shown in the table are averaged across retrieving between 1 to 10 full documents from WikiText (as opposed to half documents in our main paper experiments).

B.7 Document Chunk Statistics

In Figure 9, we plot the distribution over the lengths (tokens and characters) of retrieved document chunks used in the main paper WikiText retrieval dataset.

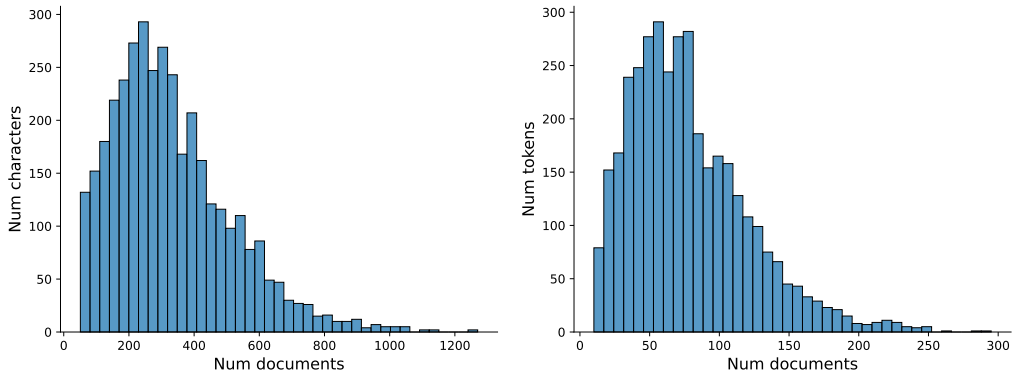

Figure 9: Histogram of the lengths, in terms of (**Left**) characters and (**Right**) tokens, of database document chunks used in the main paper WikiText experiments.

C Data Attribution

Table 4: Zero-shot Data Attribution on MSMARCO with Mamba2-2.7B, measured by precision. We compare Leave-One-In (LOI) and Leave-One-Out (LOO), where we implement LOO with varying methods for state composition.

LOI	Concat	Soup	CASO	PICASO-R	PICASO-S
0.699	0.690	0.629	0.725	0.732	0.731

Consider a question-answer pair $(\boldsymbol{u}_q, \boldsymbol{u}_a)$, and a sequence of potentially relevant documents $S = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_n)$. We would like to select the most relevant document for inferring the answer. There are at least two ways to do so with model f_θ:

The first method, which we term "Leave-one-in", is to prepend each candidate document to the question, and evaluate the loss on the answer. Equivalently, $\arg\min_i L_{CE}(f_\theta(\boldsymbol{u}_q, x(\boldsymbol{u}_i)), \boldsymbol{u}_a)$, where we abuse notation to denote loss on the sequence (instead of token) $\boldsymbol{u}_a$.

The second method, which we term "Leave-one-out", is to compare the marginal increase in loss of the answer when removing each candidate from the composition of all of them. Equivalently, $\arg\max_i\{L_{CE}(f_\theta(\boldsymbol{u}_q, \hat{x}(S_{-i})), \boldsymbol{u}_a) - L_{CE}(f_\theta(\boldsymbol{u}_q, \hat{x}(S)), \boldsymbol{u}_a)\}$, where $\hat{x}(S_{-i})$ denotes a state composed from all documents in S other than $\boldsymbol{u}_i$.

Intuitvely, the former measures "absolute" influence of a document, while the latter measures "relative" influence computed as the marginal improvement from adding it to the set of other documents.

There are several different ways to implement the latter by varying the composition method used. We show in Table 4 that not only does Leave-One-Out perform best on the MSMARCO dataset, but implementing Leave-One-Out with PICASO-S and PICASO-R not only accelerates processing, but also surpasses the performance of conatenation. We attribute this to the permutation-invariance property of PICASO, which unlike concatenation, does not introduce irrelevant biases arising from arbitrary document orders.

D Concatenation for SSMs: Connection to jump-linear systems

Consider a collection of document segments retrieved based on relevance to a query, and sorted randomly as context to the query. While these segments, or 'chunks' share information, they are independent given the query, and their order is accidental and uninformative.

We are interested in a model that can efficiently process inputs in this format and extract all shared information from the input. Attention-based models are a natural choice because of the permutation-invariance of attention mechanisms (ignoring positional encoding), but they would have to process the entire input (all chunks) with quadratic inference cost. On the other hand, SSMs have linear cost, but they are ill-fit to process this kind of input because of the context switches, which make the Markov assumption implicit in the state representation invalid.

We consider a broader class of models, namely switching dynamical systems (or jump-Markov, jump-diffusion, or linear hybrid, or jump-linear systems) as the class of interest. A jump-linear system is one that has a continuous state, say x_t that evolves synchronously, and a dicrete state that changes the value of x_t, for instance

$$x_{t+1} = \begin{cases} Ax_t + Bu_t & \text{if } t \in \mathcal{Z}\backslash\Omega \\ x_{t+1} \sim P & \text{if } t \in \Omega \end{cases}$$

Learning and inference for this model class corresponds to indentifcation and filtering for this class of Jump-Markov models. In addition to a random swtiching, the switch can be triggered by a particular 'flag' (value) of the input:

$$x_{t+1} = \begin{cases} Ax_t + Bu_t & \text{if } u_t \neq u_{\text{trigger}} \\ x_{t+1} \sim P & \text{if } u_t = u_{\text{trigger}} \end{cases}$$

If the value of u_{trigger} is known, then a given identification and filtering scheme can be applied by switching the estimated state according to the trigger.

Since modern state-space models are input-dependent, they automatically fit the latter class of models and can handle switches without modifications. However, what they cannot handle is the fact that the order of the chunks is uninformative. As a result, presenting the same chunks in different order would yield different states. Accordingly, our goal is to enable SSMs to learn from segments up to permutations, so we can accommodate sequences where the ordering within chunks is informative and respected, while the ordering of chunks is uninformative and factored out.

E General Recurrence Structure

In the main paper, we introduced a specific recursive relation satisfied by Elementary Symmetric Polynomials. Here, we introduce a more general form which can potentially be used for more efficient implementations:

Proposition 5. *For any choice of* $1 \leq q \leq n-1$

$$e_m(A_1, \cdots, A_{n-1}) = \sum_{j=max(q+m-n+1,0)}^{min(m,q)} e_{m-j}(A_1, \cdots, A_{n-1-q}) e_j(A_{n-q}, \cdots A_{n-1})$$

Proof. We compute $e_m(A_1, \cdots, A_{n-1})$ using a Dynamic Programming (DP) approach, where we break the problem into smaller problems, and merge the solutions. First we split the $n-1$ variables at some random index q to create two partitions, $(A_1 \cdots, A_{n-1-q})$ and $(A_{n-q}, \cdots A_{n-1})$, and then compute e_{m-j} and e_j on each partition respectively. For a given value of j, $e_{m-j}(A_1, \cdots, A_{n-1-q}) e_j(A_{n-q}, \cdots A_{n-1})$ will only compute a subset of values from $e_m(A_1, \cdots, A_{n-1})$, and hence we sum over all possible values for j. □

In particular, taking $q = 1$, we obtain the following:

$$e_m(A_1, \ldots, A_{n-1}) = A_{n-1} e_{m-1}(A_1, \ldots, A_{n-2}) + e_m(A_1, \ldots, A_{n-2})$$

which we use for our implementation of PICASO-S.