QUANTIFYING MEMORY UTILIZATION WITH EFFECTIVE STATE-SIZE

Anonymous authors
Paper under double-blind review

ABSTRACT

As the space of causal sequence modeling architectures continues to grow, the need to develop a general framework for their analysis becomes increasingly important. With this aim, we draw insights from classical signal processing and control theory, to develop a quantitative measure of *memory utilization*: the internal mechanisms through which a model stores past information to produce future outputs. This metric, which we call ***effective state-size*** (ESS), is tailored to the fundamental class of *input-invariant* and *input-varying linear operators*, encompassing a variety of computational units such as variants of attention, convolutions, and recurrences. Unlike prior work on memory utilization, which either relies on raw operator visualizations (e.g. attention maps), or simply the total *memory capacity* (i.e. cache size) of a model, our metrics provide highly interpretable and actionable measurements. In particular, we show how ESS can be leveraged to improve initialization strategies, inform novel regularizers and advance the performance-efficiency frontier through model distillation. Furthermore, we demonstrate that the effect of context delimiter tokens (such as end-of-speech tokens) on ESS highlights cross-architectural differences in how large language models utilize their available memory to recall information. Overall, we find that ESS provides valuable insights into the dynamics that dictate memory utilization, enabling the design of more efficient and effective sequence models.

1 INTRODUCTION

In recent years, the success of auto-regressive sequence modeling in the context of deep learning has largely been driven by advancements in highly parallelizable causal architectures, such as the Transformer (Vaswani et al., 2023). However, despite their strong performance and hardware efficiency, understanding the inner workings of these neural networks remains a challenging task due to their non-linearity and the diversity of fundamental building blocks used. To this end, we leverage a new class of model abstractions, allowing for the development of a unified framework for the analysis of these computational units.

In particular, we note that the majority of sequence models of practical interest can formally be expressed as either linear operators or ***input-varying linear operators*** ($y = f(u)u$), generalizing the notion of adaptive, or *data-controlled* operators to a broader class than previously described in Massaroli et al. (2021); Poli et al. (2023). The input-varying linear operator framework decouples the input-varying *featurization* $u \mapsto T := f(u)$ and the linear mapping $y = Tu$ required to construct and apply the operator respectively.

This decomposition enables a wide array of deep learning primitives to be uniformly formulated as linear systems, including models like convolutions [33; 39; 30; 48], linear[1] recurrences [15; 17; 25; 53; 27; 24; 65; 42; 16], and attention variants [59; 29; 57].

Current approaches to analyzing the inner workings of input-varying linear operators often rely upon simple visualizations of the materialized operator T (or the aggregation of T across multiple layers and residuals) (Olsson et al., 2022a; Vig, 2019; Abnar & Zuidema, 2020; Ali et al., 2024; Xiao et al., 2024; Sun et al., 2024). However, these visualizations alone often fail to highlight critical properties that explain how different models construct internal representations of the input data. Moreover,

[1]Here, by *linear* we refer to the linearity of the state transition.

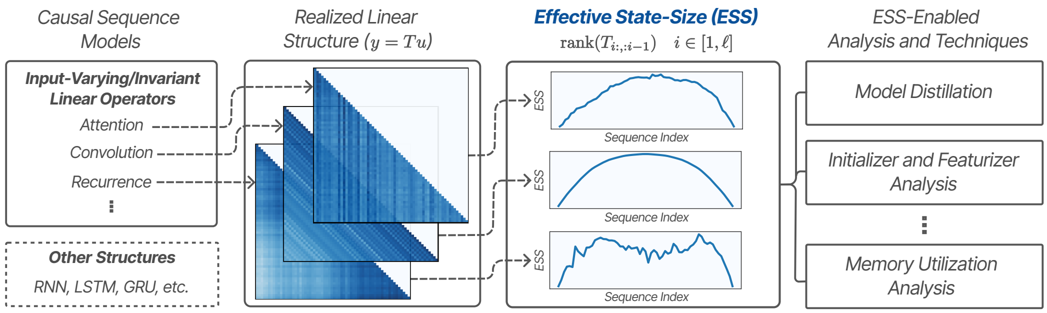


Figure 1: An overview of the effective state-size metric and its various downstream applications.

prior attempts in obtaining quantitative metrics, such as through spectral analysis of the operator T (Min & Li, 2024; Bhojanapalli et al., 2020), are either limited to a specific model class or do not appropriately take into account important conflating factors, such as the causal masking of T which significantly distorts the metric (Wu et al., 2024).

In this work, we focus our analysis on the working memory[2] of model architectures. We examine two aspects of model memory in particular: ***memory capacity*** (i.e. cache size) and ***memory utilization***. Notably, memory capacity alone can be misleading, as models with similar capacities may learn to utilize their available memory to varying degrees. Therefore, we introduce the notion of memory utilization – a measure that provides deeper insight into the differences between architectures with comparable computational efficiency.

By formalizing the duality between causal operators and recurrences (see Section 2.2), and drawing from classical signal processing and control theory, we propose a new metric called ***effective state-size*** (ESS). Extracted from the rank of specific submatrices of T, ESS serves as a proxy for the memory utilization of input-varying linear operators, encompassing the vast majority of models canonically used in causal sequence modeling. As such, it can serve as an analytical tool that can be used alongside, and compared to, memory capacity – the ***theoretically realizable state-size*** (TSS) – enabling a wide range of downstream applications (Figure 1).

In particular, our findings demonstrate the efficacy of the ESS metric in identifying undesirable memory utilization patterns at initialization, reducing inference cost via model-order reduction, mitigating poor training dynamics with regularization, and, more broadly, providing insights into the inner workings of modern sequence models.

Our technical contributions can be summarized as follows:

- We provide a theoretical derivation of the effective state-size and motivate it as a proxy for *memory utilization* in the context of both input-invariant and input-varying linear operators (Section 2).
- We motivate effective state-size beyond its interpretability by demonstrating its correlation with performance across a wide range of models and memory intensive synthetic tasks (Section 3).
- We explore the use of the effective state-size metric as a means of enhancing the performance-efficiency trade off by showcasing its application across various phases of model training (Sections 4.1, 4.2, 4.3).
- We extend the utility of effective state-size to language, demonstrating how it captures a previously uncharacterized property of LLMs: state modulation (Section 4.4).

2 Theory

In this section, we begin with a brief overview of *input-invariant* and *input-varying linear operators*, highlighting the unifying role of the linear systems formulation $y = Tu$ in analyzing modern sequence models. We proceed by showing how the operator T can be used to extract a metric that

[2]Here, we refer to "memory" in the sense commonly associated with the "state" of dynamical systems, as described by Willems (1989), as opposed to the notion of language models memorizing some fact encountered during training (Allen-Zhu & Li, 2024).

serves as a proxy for memory utilization. Namely, we prove that for any causal, input-invariant operator T, the rank of its submatrices determine the minimally realizable state-size for a linear recurrence to express T. We refer to this metric as the effective state-size of the operator, and show that even in the more complex and general case of input-varying operators (for which the minimal state-size is difficult to determine), this metric remains valuable as it provides a reliable lower bound for the minimally realizable state-size.

2.1 Preliminaries

Using the flattened notation, we let $T \in \mathbb{R}^{d\ell \times d\ell}$, $u, y \in \mathbb{R}^{d\ell}$ denote the operator, inputs, and outputs respectively, ℓ denote the sequence length and d denote the channel dimension. Here, we index sequence indices with subscripts, i.e. $T_{ij} \in \mathbb{R}^{d\times d}$, $u_i \in \mathbb{R}^d$ and channels with superscripts, i.e. $T^{\alpha\beta} \in \mathbb{R}^{\ell\times\ell}$, $u^\alpha \in \mathbb{R}^\ell$. For additional details on notation, refer to Section B.1.

A unified representation of sequence models. While typically nonlinear, most sequence models of interest can effectively materialize a linear operator T, where the equation $y = Tu$ faithfully expresses the computation performed by the model (see Section C.2 for further elaboration):

$T_{ij} = C_i B_j$ *linear attention,* $T_{ij} = C_i A_{i-1} \cdots A_{j+1} B_j$ *recurrence,*
$T_{ij} = K_{i-j}$ *convolution,* $T_{ij} = C_i K_{i-j} B_j$ *gated convolution,*
$T_{ij} = \sigma(C_i B_j)$ *attention.*

Here, we make a distinction between input-invariant operators (such as convolutions) and input-varying operators (such as attention and gated convolutions), for which the latter are constructed via *causal featurizers* that map past inputs into features, i.e. $f_B : u_{:i} \mapsto B_i$, which are then used to construct the elements of T as outlined above.

2.2 The Realization Problem

We seek to establish a connection between the operator T_{ij} of both input-invariant and input-varying linear systems and the operator corresponding to the application of linear recurrences. In doing so, we demonstrate the generality of both frames of reference, motivating the analysis of T_{ij} through its dual recurrent realizations, and in particular its dual recurrence with minimum state size.

Consider a general input-invariant linear recurrence formulated as follows:

$$\begin{aligned} s_{i+1} &= A_i s_i + B_i u_i \\ y_i &= C_i s_i + D_i u_i, \end{aligned} \tag{1}$$

where $(A_i \in \mathbb{R}^{n_{i+1}\times n_i}, B_i \in \mathbb{R}^{n_{i+1}\times d}, C_i \in \mathbb{R}^{n_i \times d}, D_i \in \mathbb{R}^{d\times d})_{i\in[\ell]}$; s_i and n_i are the state and state-size at time-step i respectively. As discussed in various prior works (Chen, 1998; DeWilde & van der Veen, 1998), system (1) realizes the following operator (see Section B.2.1 for derivations):

$$T_{ij} = \begin{cases} 0 & i < j \\ D_i & i = j \\ C_i A_{i-1} A_{i-2} \cdots A_{j+1} B_j & i > j \end{cases}. \tag{2}$$

Conversely, various instances of the recurrent realizations (of both input-varying and input-invariant operators) have been proposed for finite impulse response convolutions, lumped infinite impulse response convolutions, attention and linear attention (Chen, 1998; Katharopoulos et al., 2020; Orvieto et al., 2023b; Parnichkun et al., 2024). Here, we demonstrate that given an input-invariant operator T, there exists infinite recurrent realization variations, motivating the search for the minimal one.

Theorem 2.1. *Given any causal input-invariant operator T, there exist infinite variations of linear recurrences in the form of Equation (1) that realize an equivalent input-output operator.*

Refer to Section B.2.4 for the proof.

2.3 Effective State-Size

Now that we have established that any operator can be formulated using recurrences, we proceed by demonstrating how the minimal state-size can be determined from the structure of T.

Theorem 2.2. *The rank of the operator submatrix ($H_i \equiv T_{i:,:i-1}$) determines the minimal state size required to represent the causal operation ($y = Tu$) as a recurrence.*

Proof. The proof of Theorem 2.1 demonstrates that the operator submatrices H_i can be decomposed arbitrarily into two state-projection matrices, $\mathcal{O}_i$ and $\mathcal{C}_i$, whose inner product dimension defines the state size of its recurrent realization at time-step i. By the rank-nullity theorem, $\mathrm{rank}(H_i)$ represents the minimum inner product dimension of any such state-projection matrices, and thus corresponds to the minimally realizable state size of the operator T at time-step i. □

Therefore, decomposition methods that have minimal inner product dimensions (such as SVD) can be used to construct minimal state-projection matrices from H_i that subsequently realize minimal recurrence features $(A_i^*, B_i^*, C_i^*, D_i^*)_{i\in[\ell]}$.

Interpretation of effective state-size. Importantly, due to the input-dependence of general input-varying linear operators ($T = f(u)$), the same minimal decomposition of H_i is not guaranteed to obtain state-projection matrices in which the features do not violate causality (i.e., A_k^* depends on future inputs). Therefore, the realization process outlined in Section B.2.4 is not universally viable for obtaining minimal input-varying recurrences. One may instead resort to the trivial recurrent realization (Equation B.2.5), where the causality of the *featurization process* (the process of computing recurrent features $(A_i, B_i, C_i, D_i)_{i\in[\ell]}$) is always preserved. However, this comes with the cost of realizing a state-size that grows with the sequence length ($n_i = i$), like attention (Vaswani et al., 2023).

Despite this, $\mathrm{rank}(H_i)$ still serves as a lower bound for the state-size n_i (see Section B.2.2). This means that for any input-varying operator, an equivalent recurrence must necessarily materialize a state-size at least as large as $\mathrm{rank}(H_i)$. To this end, we formally refer to $n_i^* = \mathrm{rank}(H_i)$ as the ***effective state-size*** (ESS), and the original state size n_i as the ***theoretically realizable state size*** (TSS)[3]. We use these metrics as a proxy for analyzing various aspects of the operator, including its memory utilization, its ability to model complex long-range dependencies, and more.

2.4 Computing Effective State-Size

Computing the effective state-size requires a few additional considerations due to the numerical errors and approximations involved in practice. We propose two approaches that provide complementary perspectives on the same metric.

Tolerance-ESS. Here, a tolerance value is manually selected to threshold the singular values ($\boldsymbol{\Sigma}_i$) of H_i, determining the ESS metric as follows:

$$\text{tolerance-ESS} \coloneqq |\{\sigma_i^m : \sigma_i^m > \tau,\ \sigma_i^m \in \boldsymbol{\Sigma}_i\}|. \tag{3}$$

According to the Eckart–Young–Mirsky theorem, the tolerance-ESS metric can be interpreted as the minimum state size necessary for an input-invariant recurrence to approximate the original operator, such that the spectral norm of the approximation error remains below the specified tolerance level ($||T_{ij} - T_{ij}^*||_2 \leq \tau$).

Entropy-ESS. One drawback of tolerance-ESS is its reliance on the somewhat arbitrary selection of a tolerance value. One can instead compute the effective rank (Roy & Vetterli, 2007), which involves exponentiating the normalized spectral entropy (perplexity) of H_i:

$$\text{entropy-ESS} \coloneqq \exp\Big(-\sum_m p_i^m \log(p_i^m)\Big), \quad \text{where } p_i^m = \frac{\sigma_i^m}{\|\sigma_i\|_1}. \tag{4}$$

In contrast to the tolerance-based metric which is discrete, entropy-ESS can assume continuous values ranging from 1 to $|\boldsymbol{\Sigma}_i|$, and does not require the selection of a tolerance value. However, the normalization applied to the singular values results in the loss of absolute values, which may be significant for per-sequence-index comparisons of state size. Nonetheless, both the tolerance-based and entropy-based forms of ESS are valuable for model analysis. Entropy-ESS is particularly useful

[3] More details regarding TSS can be found in Section B.3.

for summarizing metrics across the entire tolerance space, whereas tolerance-ESS provides a more precise and readily-interpretable depiction of rank in relation to approximation error. Our code for computing ESS can be found in Section C.1.1.

3 Empirical Validation of Effective State-Size

To demonstrate the practical utility of ESS beyond its theoretical interpretation discussed in Section 2, we next turn to an empirical analysis. In this section, we examine ESS across a wide range of tasks and models in order to understand how it varies across different regimes, with particular focus placed on its relationship with model performance on memory intensive tasks.

Task space. In order to explore ESS in an extensive, yet controlled, manner, we iterate on a set of synthetic tasks proposed by Poli et al. (2024) which have been shown to effectively approximate model performance on large-scale language tasks. Specifically, we train models on the multi-query associative recall (MQAR), selective copying and compression tasks, each of which probes the ability of models to effectively utilize their working memory. We note that here, we restrict the presentation of our results to MQAR and refer the reader to Section D.1 for the results on selective copying and compression which showcase analogous trends.

Model space. We explore four models as is pertains to the scope of this analysis: gated linear attention (GLA), weighted linear attention (WLA), linear attention (LA) and softmax attention (SA). We choose this set of frameworks since, together, they capture a large portion of the space of modern sequence models. The key distinctions between these models are as follows (more details can be found in Section C.2):

- GLA layer: This layer implements the gated linear attention formulation described in Yang et al. (2024a), where the recurrent feature A (gating term) is input-varying, placing it in the same class as models like Liquid-S4 (Hasani et al., 2022) and Mamba (Gu & Dao, 2024; Dao & Gu, 2024).
- WLA layer: This layer is nearly identical to GLA, but with an input-invariant A matrix. This lies in the same class as Hyena-S4D (Poli et al., 2023), RetNets (Sun et al., 2023), and gated-convolutions in general.
- LA layer: This layer is based off Katharopoulos et al. (2020); A is not trainable and is instead fixed as the identity matrix.
- SA layer: This is the canonical attention layer which is similar to linear attention, but with the addition of a softmax non-linearity applied to the attention matrix (Vaswani et al., 2023), enabling unbounded TSS (see Section C.2 for more details).

Experimental setup. In our analysis, we exhaustively sweep across the tasks and models (which are comprised of two sequence mixing and two channel mixing layers) detailed above. Within each task, we also sweep across varying task difficulties. In the case of MQAR, we do so by modulating the number of key-value (kv) pairs the models are tasked to match, as well as the total sequence length of the prompt. Within each model, we sweep across varying TSS. For each task-model configuration, we compute the ESS and accuracy on a validation set every 10 epochs. We will refer to the entire space of task and models across which we sweep as the task-model space. Finally, we split our profiling of ESS into two sections: cross task-model analysis (Section 3.1) and within task-model analysis (Section 3.2). For more details on the setup, refer to Section C.3.

3.1 Cross Task-Model Analysis

Our first goal is to understand how ESS empirically captures memory utilization by studying its correlation with post-training MQAR performance across the entire task-model space. To appropriately analyze ESS across tasks, we normalize it by the memory demands of MQAR, constructing an adjusted form of ESS given by ESS/kv.[4]

[4]For SA, we compute (total ESS) and (total TSS) instead of (average ESS) and (average TSS) like we do for GLA/LA/WLA. This is because in SA, TSS=seqlen which does not change as a function of the model. For more details, refer to Section C.3 and C.1.

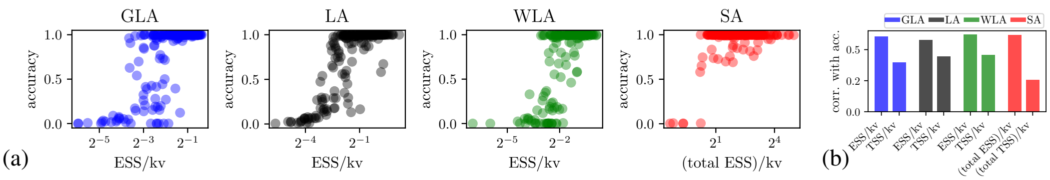


Figure 2: (a) Scatter plots of accuracy vs ESS/kv across featurizers. Within each featurizer plot, all task-model configurations from the sweep corresponding to each featurizer are shown. (b) ESS/kv vs TSS/kv as a proxy for model performance as measured by correlation.

Finding 1: Measured over entire task-model space, ESS/kv exhibits significantly higher correlation with accuracy than TSS/kv (Figures 2a, 2b, 7a, 7b).

Note that the strong correlation between ESS/kv and accuracy highlights the efficacy of ESS as a proxy for memory utilization. Furthermore, this finding underscores a significant gap in the explanatory power between ESS and TSS, emphasizing the importance of analyzing models beyond just their memory capacity.

3.2 Within Task-Model Analysis

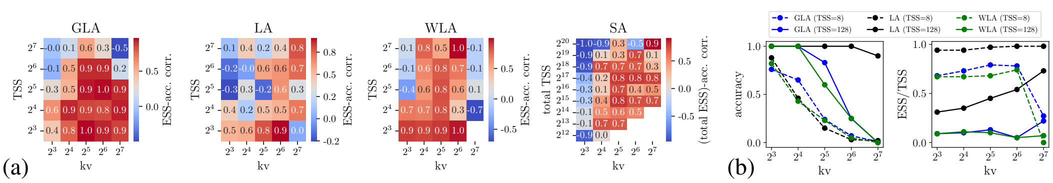


Figure 3: (a) Correlation between ESS and accuracy over course of model training bucketed by TSS and kv. (b) Accuracy and state utilization as a function of kv for low and high TSS models.

Next, to further establish ESS as a proxy for memory utilization, we study how ESS evolves as a function of MQAR performance in a regime where TSS is kept fixed and, therefore, does not correlate with accuracy. We do this by analyzing ESS-accuracy correlation on a per-model, per-task basis over the course of training, uncovering several insights that serve as the basis for our subsequent analysis.

Finding 2: For less memory-intensive tasks trained using models with high TSS, we observe a lower correlation between ESS and performance compared to more memory-intensive tasks trained using a lower TSS (Figure 3a).

This is in line with the interpretation of ESS as a measure for memory utilization. For easier tasks that are learned by a model with high memory capacity, the model is not incentivized to increase its memory utilization beyond where it resides at initialization. In contrast, for difficult tasks that operate in a memory constrained regime, the model is forced to increase its memory utilization in order to learn, resulting in strong positive correlations between accuracy and ESS over training. [5]

Digging a bit deeper, we find that this form of ESS analysis reveals two failure modes of model learning: **state saturation** and **state collapse**. State saturation refers to the scenario in which a model has insufficient TSS to fully learn a task, resulting in its ESS converging near its TSS. This is reflected in its ESS/TSS (which we refer to as state utilization) residing near 1. We observe this in Figure 3b where we note that models with a TSS of 8 perform worse on the task as its difficulty scales due to a saturated state. State collapse, on the other hand, refers to the scenario in which a model has sufficient TSS to learn (or partially learn) a task, but its ESS fails to increase during training, resulting in a heavily underutilized state. With respect to state collapse, we observe the following:

[5] In Figure 3, the empty spot in the WLA grid corresponds to a NaN from entropy ESS computation. The empty spots in the SA grid correspond to MQAR task constraints discussed in Section C.3.

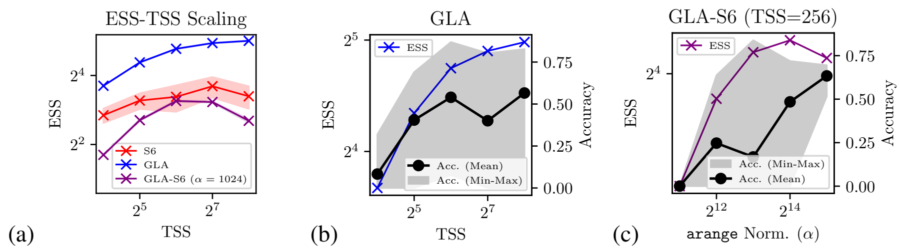


Figure 4: (a) ESS-TSS scaling in the S6, GLA and GLA-S6 featurizers. (b) ESS and accuracy on MQAR as a function of TSS in GLA. (c) ESS and accuracy on MQAR as a function of normalization factor for initialization in GLA-S6.

Finding 3: For GLA and WLA, state collapse occurs in the high kv bucket of task-model space (i.e. kv = 2^7) whereas for LA it does not (Figure 3b). For further discussion on this result, refer to Section D.1.

While state saturation can only be solved by increasing TSS, state collapse can in principle be solved by increasing ESS. Unlike TSS which is a fixed hyperparameter of the model, one can modulate ESS by changing various aspects of the model pipeline. Furthermore, even outside of the state collapse regime, given the positive correlation between ESS and performance across the task-model space, increasing ESS is a generally viable approach to improving model performance without sacrificing efficiency. We explore this idea in the results to follow.

4 Applications of Effective State-Size

In Section 3, we showed that changes in ESS are correlated with changes in performance, both across models and during model training, indicating its importance beyond just interpretability. In this section, we aim to push this insight further by understanding how we can leverage ESS to improve upon the existing performance-efficiency frontier in sequence models. We partition our results based on the stage of model training at which we apply ESS analysis: initialization-phase (Section 4.1), mid-training (Section 4.2), and post-training (Section 4.3).

4.1 Initialization-Phase Analysis

Initialization in weight space plays a crucial role in machine learning, significantly impacting model convergence and training stability (Glorot & Bengio, 2010). We extend this concept to the initialization of recurrent models in state space, leaning on the intuition from Figure 2a that suggests higher ESS can enhance performance. Namely, we illustrate how ESS at initialization can be used to inform featurizer selection – the selection of the function that maps the input to the operator $T = f(u)$ or equivalently the recurrent features $(A_i(u_{:i}), B_i(u_{:i}), C_i(u_{:i}), D_i(u_{:i}))_{i\in[\ell]}$ – and initialization schemes. In doing so, we uncover design flaws of a prominent model, S6 (Mamba) (Gu & Dao, 2024).

ESS-informed featurizer selection. To study the relationship between memory capacity and memory utilization in S6, we remove the short convolutional layer in the Mamba block and stack two of these modified blocks between SwiGLUs (Shazeer, 2020). Under the default MQAR task settings outlined in Poli et al. (2024) (see Tables 2, 3, and 4 for details), we observe that S6 is entirely unable to learn MQAR (accuracy ≈ 0) across multiple scales of TSS (16 - 256) as shown in Figure 23. This is in line with the results in Yang et al. (2024b), which also independently showed poor performance of the S6 layer without the additional short convolutional layer on a different in-context recall task. To investigate the cause, we look into how S6 is preconditioned to utilize its memory by computing its ESS when processing a Gaussian noise input, prior to training.

Finding 4: Figure 4a demonstrates that the ESS of S6 layers at initialization scales poorly with respect to TSS, notably failing to increase monotonically. In contrast, GLA layers (Yang et al., 2024a), configured with hyperparameters to match the TSS, model width, number of layers, and hidden-state normalization of the S6 model (see Section C.2 and Table 2), exhibit greater and monotonically increasing ESS-TSS scaling at initialization (Figure 4a). Despite the architectural similarities between the S6 and GLA layers, Figure 4b demonstrates that unlike S6, GLA achieves accuracy improvements that correlate with increases in both TSS and ESS. We observe even higher degrees of correlation in an alternative MQAR setting shown in Figure 24.

Based on these findings, we conjecture that the poor ESS-TSS scaling of S6 prevents the model from effectively utilizing all of its states, irrespective of increases in memory capacity.

ESS-informed initialization scheme. To further investigate the differences between the aforementioned S6 model and GLA model, we construct a composite model termed GLA-S6. This model adopts the feature sharing structure of GLA (dividing dimensions into heads and sharing computations within a head), but applies the S6 featurization to the A matrix as follows:

$$\text{GLA (original):} \qquad A = \text{diag}(\text{sigmoid}(Wu)^{1/\beta}) \tag{5}$$

$$\text{GLA-S6:} \qquad A = \text{diag}(\exp(-([1/\alpha \quad 2/\alpha \quad \dots \quad n/\alpha]^T \odot \text{softplus}(Wu)))). \tag{6}$$

Like S6, GLA-S6 fails to learn MQAR across the same range of TSS (see Figure 23) and exhibits poor initialization-ESS scaling as shown in Figure 4a. Upon further inspection, we identify the cause of poor ESS scaling: with each new state introduced, the `arange` term ($[1 \quad 2 \quad \dots \quad n]$) exponentially pushes new entries of A towards zero, negating the effects of additional states despite the increase in TSS. Therefore, to ameliorate the poor ESS scaling, we propose a simple solution: increase the normalization factor.

Finding 5: By scaling the normalization factor (α), Figure 4c shows that GLA-S6 achieves improvements in MQAR accuracy post-training, reflecting the impact of increasing its initialization-ESS, despite the models having identical memory capacities.

These experiments demonstrate the efficacy of analyzing ESS at initialization, as it reveals how different models are preconditioned to utilize their working memory. This analysis helps identify potentially weak featurization and initialization schemes, enabling us to pinpoint shortcomings in the S6 featurizer and implement a straightforward fix.

4.2 Mid-Training Analysis

To motivate the idea of increasing ESS mid-training, we revisit to the concept of state collapse – a phenomenon that arises due to trainability issues (Figure 25), as discussed in Section 3.2. Recall that state collapse describes a failure mode of learning in GLA and WLA which, unlike LA, have learnable A_i matrices (where i denotes the index along the sequence dimension). To see why this contributes to state collapse, we note that the values of the operator submatrices H_i are disproportionately influenced by A_i, due to the presence of terms in the form of $A_{i-1} \dots A_1$ for each i. Hence, the closer A_i lies to the 0-matrix, the faster these terms decay, reducing the numerical rank of H_i. We demonstrate this empirically in Figure 5a, which shows that for both GLA and WLA, ESS/kv and $\| \prod_i A_i \|_F$ decrease as a function of sequence length. In contrast, for LA, whose A matrix is given by the identity, ESS/kv remains large as sequence length grows.

Given this insight, one approach to addressing state collapse in GLA and WLA is pushing the A matrices towards the identity by adding the following term to the loss function: $\lambda \|A - I\|_F$, where λ denotes the strength of the regularizer and I denotes the identity . In doing so, we are effectively decaying the model towards LA, increasing its ESS and giving us the following:

Finding 6: GLA and WLA trained using the ESS-based regularization scheme described above outperform LA. When trained without it, they perform worse than LA (Figure 5b).

For more commentary on this result, please refer to Section D.3.

4.3 Post-Training Analysis

Recall from Section 3.1 that we observed a strong correlation between ESS and post-training performance. Building on this insight, a natural question arises: can ESS be used for more than just performance analysis in the post-training setting? In this section, we answer this question by exploring two additional post-training applications: model-order reduction and hybridization.

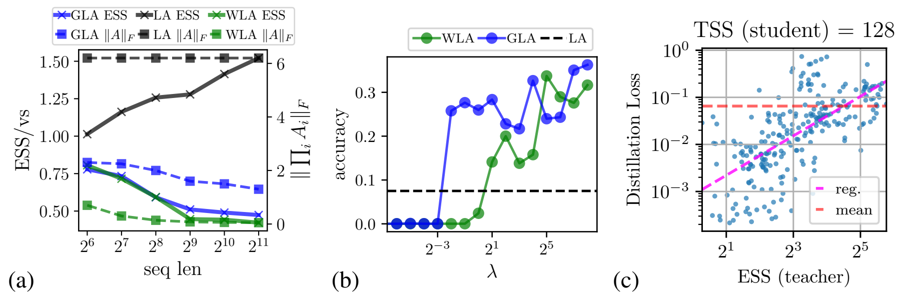


Figure 5: (a) ESS/kv and $\|\prod_i A_i\|_F$ as a function of sequence length. (b) Accuracy of models as a function of ESS-based regularizer strength. (c) Distillation loss vs ESS of the teacher model. Refer to Figure 28 for additional results.

ESS-informed model-order reduction. Model-order reduction refers to the process of improving model efficiency by reducing state-size while retaining performance. Previous works, such as Massaroli et al. (2023), have explored the distillation of time-invariant operators ($T_{ij} = T_{i+k,j+k}$) into linear recurrences with small state-sizes using backpropagation. Other techniques for model-order reduction such as modal truncation and balanced truncation (Beliczynski et al., 1992; Gawronski & Juang, 1990) are also applicable to time-invariant operators.

In this study, however, we are concerned with improving the efficiency of general *input-varying linear operators*. Since ESS serves as a lower-bound for the minimally realizable TSS (Section 2), we postulate that ESS can be used as a heuristic for conducting model-order reduction.

To test this, we distill multiple GLA models (with TSS $= 256$) across various task regimes to understand how the ESS of the original model (i.e. the teacher model) influences its ability to be distilled into a smaller student model. We apply the technique outlined in Bick et al. (2024), where the process can be divided into two-steps. 1) matching the operators ($\min(||T_{(\mathrm{s})} - T_{(\mathrm{t})}||_F^2/||T_{(\mathrm{t})}||_F^2)$) and 2) matching the output activations ($\min(||y_{(\mathrm{s})} - y_{(\mathrm{t})}||_2^2/||y_{(\mathrm{t})}||_2^2)$). More details can be found in Section C.6.

Figure 5c (and more comprehensively Figure 28) shows the relationship between the ESS of the teacher model and the final activation loss during distillation.

Finding 7: Higher teacher ESS correlates with greater activation loss. The downstream performance after single-layer distillation depends on both the teacher model's average ESS and student model's TSS, with higher teacher ESS and lower student TSS resulting in greater performance loss (Figure 27).

We also note that directly comparing student ESS against teacher ESS provides additional insights into the effectiveness of the distillation process (Figure 29). These findings position ESS as a useful heuristic for predicting model compressibility, enabling efficient estimation of the potential for state-size reduction without extensive experimentation.

ESS view on hybridization. Another application of post-training ESS analysis is network hybridization, the process of arranging different operators in a multi-layer sequence model (Lieber et al., 2024). Specifically, we measure the per-layer ESS across various hybrid networks and find that the precise ordering of layers significantly influences ESS dynamics, offering intuition as to why certain hybrids outperform others. We refer the reader to Section D.4.2 for these results.

4.4 State Modulation of Large Language Models

In contrast to synthetic tasks like MQAR, selective copying, and compression, we find that strong recall performance on language depends not only on a model having sufficient ESS, but also on its ability to dynamically modulate its ESS in response to inputs. We demonstrate that this explains why linear attention, though effective on synthetic experiments (Section 3), is widely known to perform poorly on more complex language tasks (Katharopoulos et al., 2020; Arora et al., 2024).

We begin by evaluating the total ESS (computed across layers and channels) of open-weight pre-trained models. Our analysis shown in Figure 6a (and more broadly in Section D.5) reveals an

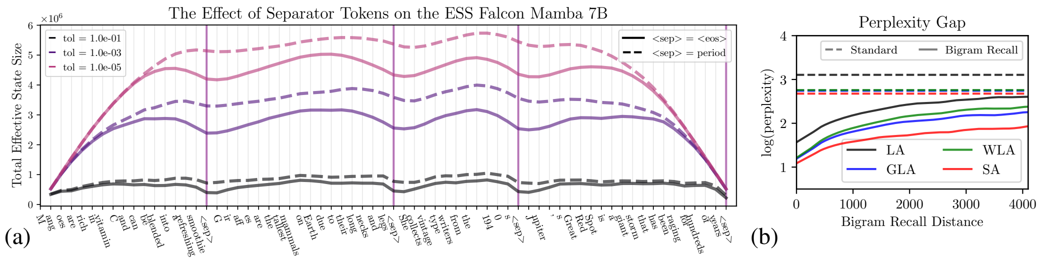


Figure 6: (a) The effect of separator tokens over Falcon Mamba 7B. See Section D.5 for plots of other open-weight models. (b) Comparison of standard perplexity and bigram recall perplexity (Arora et al., 2023).

intriguing phenomenon: ESS undergoes a noticeable dip whenever an end-of-speech (EOS) token is encountered (refer to Section C.8 for experimental details). This behavior aligns with our intuition regarding the role of EOS tokens and provides a quantitative measure of how effectively a model can ‘reset’ or ‘forget’ past contexts when transitioning between distinct segments of text[6].

To investigate these effects in a more controlled environment, we trained four 1B parameter models (LA, WLA, GLA, and SA as described in Section 3) under identical conditions (see Table 9).

Finding 8: We observe a clear hierarchy in the degree of state modulation, which can be summarized as follows: SA > GLA > WLA > LA (Figure 38).

SA exhibits the most pronounced state modulation, beginning at a tolerance level of $1\mathrm{e}{-2}$, while also realizing the largest ESS. GLA follows, with modulation emerging at a tolerance of $1\mathrm{e}{-1}$. WLA shows minimal modulation, only detectable at a tolerance of 1.0, while LA displays no discernible state modulation in response to separator tokens, demonstrating a clear lack of ability to modulate ESS.

The importance of state modulation becomes apparent when examining model performance. Figure 6b illustrates that although standard perplexities (computed over a subset of the FineWeb dataset (Penedo et al., 2024)) are similar across SA, WLA, and GLA, significant differences emerge when considering the bigram recall perplexity metric introduced by Arora et al. (2023).

Finding 9: The ability of a model to recall information, as measured by bigram recall perplexity across a pre-training dataset (rather than within a narrow task space), reveals a performance hierarchy that closely mirrors the observed state modulation capabilities.

This relationship between bigram recall perplexity and state modulation suggests that state modulation serves as a key mechanism enabling models to effectively manage complex context dependencies commonly found in language, directly impacting their training dynamics and performance on recall heavy tasks. Further implications and details are discussed in Section D.5.

5 Conclusion

In this work, we propose effective state-size (ESS), a measure of memory utilization in sequence models derived using dynamical systems theory. We motivate this metric as a valuable tool for analyzing memory utilization by demonstrating its strong correlation with performance across a wide range of synthetic tasks. In doing so, we find that ESS offers a versatile framework for understanding both the performance and efficiency of causal sequence models. Leveraging these insights, we are able to construct novel, ESS-informed initializers, regularizers and distillation strategies that improve beyond the existing performance-efficiency trade-offs in recurrent models. Finally, we extend the ESS framework to language tasks, introducing the idea of state modulation – a concept which proves crucial for performance on bigram recall tasks. Overall, this work establishes ESS as a foundational tool for understanding and improving sequence model performance, opening new avenues for optimizing memory utilization and, more generally, model efficiency.

[6] We also observe a similar behavior with scope delimiters in code (Section D.6.1).

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we utilized open-source models and tasks, adhering to default task configurations unless otherwise specified. All crucial configurations are detailed in either the main text or the appendix. Additionally, our code for computing both the tolerance-ESS and entropy-ESS is provided in the appendix (Section C.1.1).

REFERENCES

Samira Abnar and Willem Zuidema. Quantifying attention flow in transformers, 2020. URL https://arxiv.org/abs/2005.00928. (pages 1, 18).

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints, 2023. URL https://arxiv.org/abs/2305.13245. (page 23).

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. In-context language learning: Architectures and algorithms, 2024. URL https://arxiv.org/abs/2401.12973. (page 19).

Ameen Ali, Itamar Zimerman, and Lior Wolf. The hidden attention of mamba models, 2024. URL https://arxiv.org/abs/2403.01590. (pages 1, 18).

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 3.3, knowledge capacity scaling laws, 2024. URL https://arxiv.org/abs/2404.05405. (page 2).

Norah Alzahrani, Hisham Abdullah Alyahya, Yazeed Alnumay, Sultan Alrashed, Shaykhah Alsubaie, Yusef Almushaykeh, Faisal Mirza, Nouf Alotaibi, Nora Altwairesh, Areeb Alowisheq, M Saiful Bari, and Haidar Khan. When benchmarks are targets: Revealing the sensitivity of large language model leaderboards, 2024. URL https://arxiv.org/abs/2402.01781. (page 19).

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models, 2023. URL https://arxiv.org/abs/2312.04927. (pages 10, 10, 19, 19, 19, 27, 27, 27).

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff, 2024. URL https://arxiv.org/abs/2402.18668. (pages 9, 19).

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past, 2016. URL https://arxiv.org/abs/1610.06258. (page 19).

Bartlomiej Beliczynski, Izzet Kale, and Gerald D Cain. Approximation of fir by iir digital filters: An algorithm based on balanced model reduction. *IEEE Transactions on Signal Processing*, 40 (3):532–542, 1992. (page 9).

Satwik Bhattamishra, Arkil Patel, Phil Blunsom, and Varun Kanade. Understanding in-context learning in transformers and llms by learning to learn discrete functions, 2023. URL https://arxiv.org/abs/2310.03016. (page 19).

Srinadh Bhojanapalli, Chulhee Yun, Ankit Singh Rawat, Sashank J. Reddi, and Sanjiv Kumar. Low-rank bottleneck in multi-head attention models, 2020. URL https://arxiv.org/abs/2002.07028. (pages 2, 19).

Aviv Bick, Kevin Y. Li, Eric P. Xing, J. Zico Kolter, and Albert Gu. Transformers to ssms: Distilling quadratic knowledge to subquadratic models, 2024. URL https://arxiv.org/abs/2408.10189. (page 9).

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. Thread: Circuits. *Distill*, 2020. doi: 10.23915/distill.00024. https://distill.pub/2020/circuits. (page 18).

Chi-Tsong Chen. *Linear System Theory and Design*. Oxford University Press, Inc., USA, 3rd edition, 1998. ISBN 0195117778. (pages 1, 3, 3).

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL https://arxiv.org/abs/2405.21060. (pages 1, 5, 18, 18).

P. DeWilde and A.J. van der Veen. *Time-Varying Systems and Computations*. Springer US, 1998. ISBN 9780792381891. URL https://books.google.co.jp/books?id=n3bEniJ2Wx8C. (pages 1, 3, 18, 22).

Yihe Dong, Jean-Baptiste Cordonnier, and Andreas Loukas. Attention is not all you need: Pure attention loses rank doubly exponentially with depth, 2023. URL https://arxiv.org/abs/2103.03404. (page 18).

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes, 2019. URL https://arxiv.org/abs/1904.01681. (page 19).

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html. (page 19).

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models, 2023. URL https://arxiv.org/abs/2212.14052. (page 19).

Wodek Gawronski and Jer-Nan Juang. Model reduction in limited time and frequency intervals. *International Journal of Systems Science*, 21(2):349–376, 1990. doi: 10.1080/00207729008910366. URL https://doi.org/10.1080/00207729008910366. (page 9).

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html. (page 7).

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752. (pages 1, 5, 7, 18, 18, 27).

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2022a. URL https://arxiv.org/abs/2111.00396. (pages 1, 18, 23).

Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the parameterization and initialization of diagonal state space models, 2022b. URL https://arxiv.org/abs/2206.11893. (page 18).

Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models, 2022. URL https://arxiv.org/abs/2209.12951. (pages 1, 5, 18).

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300. (page 19).

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236. (pages 1, 3, 5, 9, 18, 18, 23, 25).

Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1d convolutional neural networks and applications: A survey, 2019. URL https://arxiv.org/abs/1905.03554. (page 1).

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model, 2024. URL https://arxiv.org/abs/2403.19887. (pages 9, 31, 47).

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101. (pages 29, 30, 31, 32).

Paul A. Lynn and Wolfgang Fuerst. *Introductory digital signal processing with computer applications (revised ed.)*. John Wiley & Sons, Inc., USA, 1994. ISBN 0471943746. (page 1).

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes, 2021. URL https://arxiv.org/abs/2002.08071. (page 1).

Stefano Massaroli, Michael Poli, Daniel Y. Fu, Hermann Kumbong, Rom N. Parnichkun, Aman Timalsina, David W. Romero, Quinn McIntyre, Beidi Chen, Atri Rudra, Ce Zhang, Christopher Re, Stefano Ermon, and Yoshua Bengio. Laughing hyena distillery: Extracting compact recurrences from convolutions, 2023. URL https://arxiv.org/abs/2310.18780. (pages 9, 18).

Zeping Min and Zhong Li. On the efficiency of transformers: The effect of attention rank, 2024. URL https://openreview.net/forum?id=U9sHVjidYH. (pages 2, 18).

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022a. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html. (pages 1, 18).

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022b. https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html. (page 19).

Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab. *Signals & systems (2nd ed.)*. Prentice-Hall, Inc., USA, 1996. ISBN 0138147574. (page 1).

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences, 2023a. URL https://arxiv.org/abs/2303.06349. (page 18).

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pp. 26670–26698. PMLR, 2023b. (page 3).

Rom N. Parnichkun, Stefano Massaroli, Alessandro Moro, Jimmy T. H. Smith, Ramin Hasani, Mathias Lechner, Qi An, Christopher Ré, Hajime Asama, Stefano Ermon, Taiji Suzuki, Atsushi Yamashita, and Michael Poli. State-free inference of state-space models: The transfer function approach, 2024. URL https://arxiv.org/abs/2405.06147. (pages 1, 3, 18).

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale, 2024. (pages 10, 33).

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y. Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models, 2023. URL https://arxiv.org/abs/2302.10866. (pages 1, 5, 18).

Michael Poli, Armin W Thomas, Eric Nguyen, Pragaash Ponnusamy, Björn Deiseroth, Kristian Kersting, Taiji Suzuki, Brian Hie, Stefano Ermon, Christopher Ré, Ce Zhang, and Stefano Massaroli. Mechanistic design and scaling of hybrid architectures, 2024. URL https://arxiv.org/abs/2403.17844. (pages 5, 7, 19, 27).

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177. (page 18).

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need, 2021. URL https://arxiv.org/abs/2008.02217. (page 19).

David W. Romero, Anna Kuzina, Erik J. Bekkers, Jakub M. Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data, 2022. URL https://arxiv.org/abs/2102.02611. (page 1).

Olivier Roy and Martin Vetterli. The effective rank: A measure of effective dimensionality. In *2007 15th European Signal Processing Conference*, pp. 606–610, 2007. (page 4).

Noam Shazeer. Fast transformer decoding: One write-head is all you need, 2019. URL https://arxiv.org/abs/1911.02150. (page 23).

Noam Shazeer. Glu variants improve transformer, 2020. URL https://arxiv.org/abs/2002.05202. (page 7).

Huitao Shen. Mutual information scaling and expressive power of sequence models, 2019. URL https://arxiv.org/abs/1905.04271. (page 18).

Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling, 2023. URL https://arxiv.org/abs/2208.04933. (pages 1, 18, 23).

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864. (pages 26, 27).

Mingjie Sun, Xinlei Chen, J. Zico Kolter, and Zhuang Liu. Massive activations in large language models, 2024. URL https://arxiv.org/abs/2402.17762. (pages 1, 18).

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023. URL https://arxiv.org/abs/2307.08621. (pages 5, 18).

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: A unified understanding of transformer's attention via the lens of kernel, 2019. URL https://arxiv.org/abs/1908.11775. (pages 1, 18).

Neehal Tumma, Mathias Lechner, Noel Loo, Ramin Hasani, and Daniela Rus. Leveraging low-rank and sparse recurrent connectivity for robust closed-loop control, 2023. URL https://arxiv.org/abs/2310.03915. (page 18).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762. (pages 1, 1, 4, 5, 18, 18, 23).

Jesse Vig. A multiscale visualization of attention in the transformer model, 2019. URL https://arxiv.org/abs/1906.05714. (pages 1, 18).

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark, 2024. URL https://arxiv.org/abs/2406.01574. (page 19).

Jan C. Willems. *Models for Dynamics*, pp. 171–269. Vieweg+Teubner Verlag, Wiesbaden, 1989. ISBN 978-3-322-96657-5. doi: 10.1007/978-3-322-96657-5_5. URL https://doi.org/10.1007/978-3-322-96657-5_5. (page 2).

Xinyi Wu, Amir Ajorlou, Yifei Wang, Stefanie Jegelka, and Ali Jadbabaie. On the role of attention masks and layernorm in transformers, 2024. URL https://arxiv.org/abs/2405.18781. (pages 2, 19).

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453. (pages 1, 18).

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training, 2024a. URL https://arxiv.org/abs/2312.06635. (pages 1, 5, 7, 18, 18, 30).

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length, 2024b. URL https://arxiv.org/abs/2406.06484. (page 7).

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence?, 2019. URL https://arxiv.org/abs/1905.07830. (page 19).

Supplementary Material

Contents

A Related Work

Causal sequence models. From classical linear recurrences to modern sequence models like Transformers, a vast array of causal model architectures have emerged (Vaswani et al., 2023; Tsai et al., 2019; Katharopoulos et al., 2020; Poli et al., 2023; Yang et al., 2024a; Gu & Dao, 2024; Dao & Gu, 2024; Sun et al., 2023). In recent years, the ability to process sequences in parallel has become increasingly critical, largely due to advancements in hardware accelerators such as GPUs. This need for parallelism likely explains the growing popularity of models like attention, Mamba, and S4.

We observe that all of these models, which support parallelization across the sequence dimension, can be formulated using a linear system representation ($y = Tu$) as detailed in the introductory and theoretical sections (Sections 1 and 2). For this work, we categorize these models into two types: input-invariant and input-varying linear operators. Input-invariant operators encompass both linear time-varying (LTV) and linear time-invariant (LTI) systems. The key distinction between these two frameworks is that the operator T in input-invariant models is composed of fixed system parameters as opposed to parameters being dynamically generated from the input. Although LTV systems have been relatively unexplored in deep learning, several LTI models have been studied (Gu et al., 2022a;b; Smith et al., 2023; Orvieto et al., 2023a; Parnichkun et al., 2024). Convolutional models use kernels h to construct Hankel matrices H, whose rank corresponds to the minimal state-size of the model (DeWilde & van der Veen, 1998). Massaroli et al. (2023) explored methods to reduce the order of models by leveraging the Hankel matrix. Notably, the submatrix H_i (defined in Theory 2.2) exhibits a Hankel structure in LTI models and provides per-sequence-index information. In this work, however, we do not explore Hankel matrices further, as they are not easily generalizable to LTV systems.

In contrast, input-varying linear operators are characterized by an operator T that is dynamically constructed through a featurizer and is defined by $T = f(u)$. Examples of such models include softmax attention (Vaswani et al., 2023), linear attention (Katharopoulos et al., 2020), Liquid-S4 (Hasani et al., 2022), Mamba (Gu & Dao, 2024; Dao & Gu, 2024), and gated linear attention (Yang et al., 2024a). Although these models may appear nonlinear in nature, they can still be represented as input-varying linear operators, enabling the application of linear analysis techniques. This forms the basis for the effective state-size metric.

Interpretability. Analysis tools for sequence models can be categorized into two types: extrinsic and intrinsic. Extrinsic tools focus solely on the input and output, treating the model's internal processes as black boxes. This approach is highly generalizable as it can be applied to any model, including those with non-linear recurrences. A notable example by Shen (2019) uses statistical measures such as mutual information to compute metrics that capture model "expressivity". While these methods are versatile and applicable to various datasets, their generality makes them less effective at capturing the inner workings of causal sequence models, which is the primary focus of this work.

Intrinsic tools, conversely, directly visualize the model's internal mechanisms. A recently popular framework known as mechanistic interpretability provides one such example (Power et al., 2022). Mechanistic interpretability involves dissecting complex models to understand how specific components contribute to the model's overall behavior (Cammarata et al., 2020). Unlike our work, mechanistic interpretability does not target the operator view of the model but instead emphasizes the functional roles and interactions of individual model components.

For our purposes, we are primarily concerned with the visualization and analysis of classical and modern causal sequence models through the unifying lens of input-invariant and input-varying linear operators. Most analyses of these operators rely on visualization techniques (Olsson et al., 2022a; Vig, 2019; Abnar & Zuidema, 2020; Ali et al., 2024; Xiao et al., 2024; Sun et al., 2024) to gain insights into the model's internal processes. Visualizing the operator T is advantageous, as it reveals important features like the formation of induction heads, strong activations, diagonal and block-diagonal patterns, and Toeplitz structures. However, raw visualizations are largely qualitative and often times do not provide the quantitative metrics necessary for effectively evaluating a model's internal mechanisms – a gap we aim to address in this work.

Other, more quantitative intrinsic methods include spectral analysis of the full operator, which has led to theoretical works like (Dong et al., 2023) and empirical studies (Min & Li, 2024; Tumma et al., 2023; Bhojanapalli et al., 2020). A limitation of these approaches is that they often disregard the causal masking of T, which significantly impacts the model's rank and singular values (Wu et al., 2024). As a result, the rank of the causal operator T alone lacks a clear interpretation.

The proposed effective state-size metric is an intrinsic method applicable to both input-invariant and input-varying linear operators. As a quantitative proxy for memory utilization, it offers insights into the inner workings of causal sequence models, ensuring generality, usability, and interpretability.

Synthetic and language benchmarks. In this work, we build on synthetic tasks from the mechanistic architecture design (MAD) framework introduced in (Poli et al., 2024). MAD defines a set of small-scale tasks designed to evaluate key model capabilities such as in-context recall (Akyürek et al., 2024; Bhattamishra et al., 2023; Elhage et al., 2021; Olsson et al., 2022b). Training models on these tasks is efficient, making them well-suited for exploring a large space of tasks and models, as demonstrated in several prior works (Dupont et al., 2019; Arora et al., 2024; Fu et al., 2023). In this work, we investigate the effective state-size across a subset of the MAD tasks: multi-query associative recall (MQAR), selective copying, and compression, varying the difficulty of each to gain a nuanced understanding of how effective state-size evolves across these task landscapes.

Among the synthetic tasks we examine, MQAR stands out in particular. Proposed by Arora et al. (2023), MQAR was designed to bridge the gap between synthetic and real language tasks explained by associative recall – the ability of a model to retrieve information based on relationships between different elements in its memory. This capability has long been sought after in the construction of sequence model architectures (Ramsauer et al., 2021; Ba et al., 2016); as such, we evaluate the performance of our models on MQAR to measure the benefits of using effective state-size to iterate on canonical frameworks used in sequence modeling.

One notable aspect of MQAR observed in Arora et al. (2023) is that the size of the model cache needs to scale with the difficulty of the task to maintain performance. While this observation holds, our work demonstrates that model cache size is an imperfect measure in this context due to the discrepancy between memory capacity, as measured by theoretically realizable state size, and memory utilization, as measured by effective state-size. At a higher level, this demonstrates how our work provides a new perspective on analyzing memory-intensive synthetic tasks.

While the MAD framework and synthetic tasks have shown correlations with model performance on large-scale language tasks, language itself poses a unique challenge. Models are tasked with predicting the next token given previous tokens – a simple yet general objective. New tasks can be created simply by altering the prompts, thereby expanding the range of possible task domains.

Although numerous language evaluation tasks – such as those in Hendrycks et al. (2021); Wang et al. (2024); Zellers et al. (2019) – have been proposed, they often probe a narrow task space and tend to be brittle. For example, shuffling the order of multiple choices in MMLU can drastically change model rankings Alzahrani et al. (2024).

Unlike narrow benchmarks, perplexity scores can be computed across an entire pre-training dataset, covering a much broader task domain. However, small perplexity gaps between models make it a challenging metric for evaluation. Recently, Arora et al. found that much of the difference in perplexity between models can be attributed to bigram perplexity – a measure of a model's ability to utilize the context and predict a successor token (second token of a bigram) given a repeated context token (first token of a bigram) within a sequence. They demonstrate that most of the average perplexity difference between a gated convolution model and an attention model stems from differences in bigram perplexity, suggesting that recall is a key capability for language models.

The effective state-size analysis presented in this work reveals that strong recall performance as measured by bigram perplexity in language modeling tasks depends not only on memory capacity, but also on a model's ability to modulate its state-size within a given context.

B THEORETICAL BACKGROUND

B.1 NOTATION

We adopt the following notation in this paper:

- Inputs, outputs, and operators follow flattened notation. I.e., $u, y \in \mathbb{R}^{\ell d}$ and $T \in \mathbb{R}^{\ell d \times \ell d}$. In particular, the original inputs and outputs with shape $\ell \times d$ are flattened in row-major ordering, resulting in T having $\ell \times \ell$ sub-blocks, each of which are of size $d \times d$.

- Tensor subscripts index sequence indices (time-step) and superscripts index channel/hidden dimensions. I.e., for an input $u \in \mathbb{R}^{\ell d}$, $u_i \in \mathbb{R}^d$ denotes the input vector at sequence-index i, and $u^\alpha \in \mathbb{R}^\ell$ denotes the input vector for channel α. Similarly, $T_{ij} \in \mathbb{R}^{d \times d}$ denotes the linear weighing of u_j on to y_i.

- Indices within square brackets indicate matrix indices void of semantics (sequence index, channels, etc.). I.e., $A_{i[\alpha,\beta]}$ indexes row α and column β of matrix A_i.

- Semicolons within subscripts denote a product over ranges ($A_{1;3} = A_1 A_2 A_3$).

- Tensor slices are denoted with colons and are inclusive over the ranges. I.e., $u_{0:2} = u_0 u_1 u_2$.

B.2 DERIVATIONS AND PROOFS

B.2.1 THE OPERATOR REALIZATION OF LINEAR RECURRENCES

Unrolling the recurrence in Equation 1 unveils the follow formulation:

$$\begin{aligned}
s_0 &= 0 \\
s_1 &= B_0 u_0 \\
s_2 &= B_1 u_1 + A_1(B_0 u_0) \\
s_3 &= B_2 u_2 + A_2(B_1 u_1 + A_1(B_0 u_0)) \\
s_i &= \left(\sum_{j=0}^{i-1} \left[\prod_{k=i-1}^{j+1} A_k \right] B_j u_j \right), \qquad \text{(B.2.1)} \\
y_i &= C_i \left(\sum_{j=1}^{i-1} \left[\prod_{k=i-1}^{j+1} A_k \right] B_j u_j \right) + D_i u_i, \qquad \text{(B.2.2)}
\end{aligned}$$

which corresponds to the operator:

$$T_{ij} = \begin{cases} 0 & i < j \\ D_i & i = j \\ C_i A_{i-1;j+1} B_j & i > j \end{cases}. \qquad \text{(B.2.3)}$$

B.2.2 FACTORIZING THE OPERATOR REALIZATION SUBMATRIX H_i

Factorizing the strictly lower triangular submatrices of the operator ($T_{i:,:i-1}$) into causal and anti-causal factors, unveils that the theoretical state-size (n_i) upper bounds the inner product's dimensionality, and therefore the rank of the submatrix($n_i \geq \text{rank}(H_i)$):

$$
\begin{aligned}
T_{i:,:i-1} \equiv H_i &= \begin{bmatrix} C_i & & \\ & \ddots & \\ & & C_{\ell-1} \end{bmatrix} \begin{bmatrix} A_{i-1;1} & \dots & I \\ \vdots & \ddots & \vdots \\ A_{\ell-2;1} & \dots & A_{\ell-2;i} \end{bmatrix} \begin{bmatrix} B_0 & & \\ & \ddots & \\ & & B_{i-1} \end{bmatrix} \\
&= \begin{bmatrix} C_i & & \\ & \ddots & \\ & & C_{\ell-1} \end{bmatrix} \begin{bmatrix} I \\ A_i \\ A_{i+1;i} \\ \vdots \\ A_{\ell-2;i} \end{bmatrix} \begin{bmatrix} A_{i-1;1} & A_{i-1;2} & \dots & A_{i-1} & I \end{bmatrix} \begin{bmatrix} B_0 & & \\ & \ddots & \\ & & B_{i-1} \end{bmatrix} \\
&= \underbrace{\begin{bmatrix} C_i \\ C_{i+1}A_i \\ C_{i+2}A_{i+1;i} \\ \vdots \\ C_{\ell-1}A_{\ell-2;i} \end{bmatrix}}_{\substack{d(\ell-i) \times n_i \\ \textit{anti-causal}}} \underbrace{\begin{bmatrix} A_{i-1;1}B_0 & A_{i-1;2}B_1 & \dots & A_{i-1}B_{i-2} & B_{i-1} \end{bmatrix}}_{\substack{n_i \times di \\ \textit{causal}}} \equiv \mathcal{O}_i \mathcal{C}_i.
\end{aligned}
\tag{B.2.4}
$$

Besides unveiling the relationship between the rank of the realized operator and the original state-size n_i, the following insights can be drawn from the decomposition:

- The causal portion $\mathcal{C}_i$ is the input-state projection matrix at time-step i (i.e., $s_i = \mathcal{C}_i u_{:i-1}$) corresponding to Equation (B.2.1).
- ESS ($\text{rank}(H_i)$) is simply the minimum rank between the causal and anti-causal projections.
- In conjunction with Theorem 2.2, we observe that the causally determinable minimal state-size (causal ESS) is equivalent to the rank of the causal projection. This insight allows us to construct a more efficient realization of the recurrence:
  - We can minimally factorize the causal projection as $\mathcal{C}_i = L_i R_i$, where $L_i \in \mathbb{R}^{n_i \times r}$ and $R_i \in \mathbb{R}^{r \times di}$, with $r = \text{rank}(\mathcal{C}_i)$.
  - The right factor R_i becomes the new input-state projection matrix for H_i, effectively reducing the state dimension to the causal ESS.
  - A^*_{i-1} and B^*_{i-1} can be determined from R_i using the process outlined in Theorem 2.1, and $C^*_i = C_i L_i$.

B.2.3 The Trivial Recurrence Realization

Any input-varying and input-invariant causal operator can be trivially realized with the following recurrence:

$$
\begin{aligned}
s_{i+1} &= \begin{bmatrix} I_{(di)} \\ 0_{(d)} \end{bmatrix} s_i + \begin{bmatrix} 0_{(di)} \\ I_{(d)} \end{bmatrix} u_i, \\
y_i &= \begin{bmatrix} T_{i,0} & T_{i,1} & \cdots & T_{i,i-1} \end{bmatrix} s_i + T_{i,i} u_i.
\end{aligned}
\tag{B.2.5}
$$

In simple terms, the state s_i stores each input from $t \in [i-1]$, which is then mapped to the output with operator features at row i. Note that in the case where the operator is input varying, the trivial realization upholds the causality of the featurization process (i.e. the features $(A_i, B_i, C_i, D_i)_{i \in [\ell]}$ of the trivial realization are causally determined). Moreover, the causally determined ESS (see Section B.2.2) for the trivially realized recurrence is equivalent to its TSS, as $\mathcal{C}_i = I_{di}$.

B.2.4 Minimal Recurrent Realization (Proof of Theorem 2.1)

Theorem 2.1 *Given any causal input-invariant operator T, there exist infinite variations of linear recurrences in the form of Equation (1) that realize an equivalent input-output operator.*

Proof. We first categorize the operator into two portions: the memoryless portion, where $i = j$, and the dynamical portion, where $i > j$. The memoryless portion can be trivially realized by setting $D_i = T_{ii}$. For the dynamical portion, we draw inspiration from (DeWilde & van der Veen, 1998, ch. 3) and approach the proof of existence by ansatz. The following steps outline the proof:

1. Section B.2.2 demonstrates that, given a linear recurrence in the form of Equation (1), the operator submatrix can be factorized into causal and anti-causal parts, where the causal part represents the input-state projection matrix. We therefore proceed by making the ansatz that, for any operator submatrix $T_{i:,:i-1} \equiv H_i$, H_i can be arbitrarily factorized into $\mathcal{O}_i \in \mathbb{R}^{d(\ell-i)\times n_i}$ and $\mathcal{C}_i \in \mathbb{R}^{n_i \times di}$, and that $\mathcal{C}_i$ represents the input-state projection at time-step i (i.e., $s_i = \mathcal{C}_i u_{:i-1}$).
2. Construct the dynamic features $(A_i, B_i, C_i)_{i\in[\ell]}$ such that the assumption above holds. Note that we additionally assume the initial and final states to be 0 without loss in generality, therefore the realization of C_0, A_0, $A_{\ell-1}$, and $B_{\ell-1}$ could be ignored.
   (a) Set $C_i = \mathcal{O}_{i[:d-1]}$ to obtain $(C_i)_{i\in[1,\ell]}$, as given the assumptions above, the first set of rows of $\mathcal{O}_i$ linearly projects s_i onto $y_i - D_i u_i$, which is identical to C_i in Equation (1).
   (b) Set $B_{i-1} = \mathcal{C}_{i[:,-d:]}$ to obtain $(B_i)_{i\in[\ell-1]}$, for which the identity can be obtained by deconstructing the input-state projection matrix $\mathcal{C}_i$ and equating its assumed state s_i with Equation (1).
   $$\begin{aligned} s_i &= A_{i-1}s_{i-1} + B_{i-1}u_{i-1} \\ &= \mathcal{C}_{i[:,:-d-1]}u_{:i-2} + \mathcal{C}_{i[:,-d:]}u_{i-1}. \end{aligned} \tag{B.2.6}$$
   (c) Using the same state-dynamics equation, we could equate the assumed state-projection matrices with each other obtaining $(A_i)_{i\in[1,\ell-1]}$:
   $$\begin{aligned} s_{i+1} &= A_i s_i + B_i u_i \\ \mathcal{C}_{i+1}u_{:i} &= A_i\mathcal{C}_i u_{:i-1} + \mathcal{C}_{i+1[:,-d:]}u_i \\ \mathcal{C}_{i+1[:,:-d-1]}u_{:i-1} &= A_i\mathcal{C}_i u_{:i-1} \\ A_i &= \mathcal{C}_{i+1[:,:-d-1]}\mathcal{C}_i^+. \end{aligned} \tag{B.2.7}$$
3. Verify that the realized recurrence maps back to the original operator T_{ij}, proving that arbitrary factorizations (of which there are an infinite variations) of the operator submatrices can be used to construct equivalent operators.
   $$\begin{aligned} T_{ij} = C_i A_{i-1}\cdots A_{j+1}B_j &= \mathcal{O}_{i[:d-1]}\mathcal{C}_{i[:,:-d-1]}\cdots\mathcal{C}_{j+2}^+\mathcal{C}_{j+2[:,:-d-1]}\mathcal{C}_{j+1}^+\mathcal{C}_{j+1[:,-d:]} \\ &= \mathcal{O}_{i[:d-1]}\mathcal{C}_{i[:,:-d-1]}I_{[:,:(j+1)d-1]}I_{[:,-d:]} \\ &= \mathcal{O}_{i[:d-1]}\mathcal{C}_{i[:,jd:(j+1)d-1]} = H_{i[:d-1,jd:(j+1)d-1]} = T_{ij}. \end{aligned} \tag{B.2.8}$$

□

As an example, H_i can be factorized with SVD as follows:

$$\mathcal{O}_i\mathcal{C}_i = (U_{(r)}D_{(r)}^{1/2})(D_{(r)}^{1/2}V_{(r)}),$$

where $U_{(r)} \in \mathbb{R}^{m\times r}$, $D_{(r)} \in \mathbb{R}^{r\times r}$, $V_{(r)} \in \mathbb{R}^{r\times n}$ are the r-truncated SVD decompositions, and $r = \text{rank of } H_i \in \mathbb{R}^{m\times n}$. These factors can then used to realize a minimal recurrence as outlined above.

B.3 More on the Theoretically Realizable State-Size

As defined in Section 2, we define the theoretically realizable state-size as n_i in Equation 1. We make the distinction between the TSS and the algorithm-specific cache-size (i.e. number of elements in a key-value cache of an attention layer (Vaswani et al., 2023; Ainslie et al., 2023; Shazeer, 2019)), though they generally differ only by a scaling constant.

TSS is a formulation-specific metric. A good example to showcase this point is the difference between the formulation of attention and linear attention Katharopoulos et al. (2020). The former can only be realized trivially (Equation B.2.5), whereas the latter can be formulated either trivially or as a recurrence with a fixed state-size of d/h (per channel), where h is the number of "heads" (see Section C.2).

We note that irrespective of the particular formulation of the recurrence, the ESS metric unveils the fixed state-size nature of linear attention in stark contrast to the growing state-size of attention models, further motivating the use of the ESS metric (Figure 38).

C Methods

C.1 Computing ESS

In Section 2 and B.1, we introduced the flattened notation as it offers a general framework for formulating a wide range of operators and recurrences. As an example, an S5 layer (Smith et al., 2023), which mixes both the channels and sequence simultaneously, can be formulated as $y = Tu$ (with the operator realization outlined in B.2.1) in the same way an S4 layer can (Gu et al., 2022a), which only mixes the sequence. The difference between these two models lies in the structure of T: for models that only mix the sequence, such as S4, T_{ij} is diagonal, whereas for S5, it is not.

Note that since all of the models in our experiments have decoupled channel mixing and sequence mixing (like the S4 layer), we compute the effective state-size independently for each channel using the standard operator formulation $T \in \mathbb{R}^{\ell \times \ell}$. This approach is significantly more efficient than computing ESS for the multi-channel (flattened) representation. Furthermore, in the case of attention layers, the computation can be further reduced to only the h independent heads, as the operator (i.e. the attention matrix) is shared across channels within the same head.

In our experiments, the shape of the unprocessed ESS tensor is given by

$$(\text{batch-size, layers, heads or channels, sequence length} - 1),$$

for a multi-layered model that is processing a batch of sequences. Unless stated otherwise, we compute ESS metrics averaged across all dimensions with an exception made for softmax attention.

Due to the recurrent realization of softmax attention being constrained to that of the trivial form (Section B.2.3, C.2), the per-channel TSS (n_i) of these models depends only on the sequence length i. In this setting, the average TSS across channels remains constant regardless of the width of the model (even when Q and K expansion factors are applied), and therefore no meaningful variations in TSS are captured by changing the model width. To appropriately capture differences in TSS, we instead sum over the ESS across the channels of each layer, then compute the average over that sum. We denote metrics computed in this manner with a prefix "total", i.e., "total ESS" and "total TSS" as it captures the total TSS or ESS of a model layer[7]. Additionally, for the analyses presented in Section 3, we average across 8 samples (batch-size), and for the rest, we average across 32 samples.

Regarding the distinction between the entropy and tolerance based forms of ESS, we note that entropy-ESS is a valuable summary metric because its computation is independent of any specific tolerance value chosen. However, it can potentially be misleading when comparing ESS across sequence indices due to the unequal normalization applied to the singular values. Conversely, when comparing entropy-ESS across different operators, it can be useful as the normalization removes the effect of the norm of the operator. In most of our experiments, we observe consistent trends between entropy-ESS and tolerance-ESS when the metrics are marginalized over the sequence length. Therefore, unless stated otherwise, our figures are presented using the entropy-ESS. In cases where we

[7]We note that ESS can capture differences in memory utilization under both metric marginalization approaches.

require ESS comparison across the sequence dimension, we instead plot ESS for multiple tolerance values.

C.1.1 PyTorch Implementation

Below, we provide a PyTorch implementation of various ESS metrics and helper functions that were leveraged in our analyses:

```python
import torch

def T2H_i(T, i, d=1):
    """
    Extract H_i from T.

    Args:
        - T: Flattened operator with shape [..., d*L, d*L].
        - i: Index of H (H_i) to retrieve.
        - d: Block size for multi-channel flattened operator
    representation (default is 1).

    Returns:
        - H_i: Submatrix of the operator at index i.
    """
    return T[...,d*i:,:d*i]

@torch.no_grad()
def T2Ss(T, d=1):
    """
    Converts an operator into a list of singular values (Ss).

    Args:
        - T: Flattened operator with shape [..., d*L, d*L]
        - d: Block size for multi-channel flattened operator
    representation (default is 1).

    Returns:
        - Ss: A list of singular values for each sequence index in T.
    """
    seqlen = T.size(-2)//d
    Ss = []
    for i in range(1, seqlen):
        H_i = T2H_i(T, i, d)
        _, S_i, _ = torch.svd(H_i)
        Ss.append(S_i)
    return Ss

@torch.no_grad()
def Ss2ToleranceESS(Ss, tol=1e-4):
    """
    Computes the tolerance-ESS from the list of singular values.

    Args:
        - Ss: List of singular values.
        - tol: Tolerance value.

    Returns:
        - tolerance-ESS
    """
    ranks = []
    for SV in Ss:
        rank = torch.sum(SV>=tol, dim = -1)
        ranks.append(rank)
    ranks = torch.stack(ranks, dim=-1)
    return ranks
```

```python
@torch.no_grad()
def Ss2EntropyESS(Ss, clip=1e-12):
    """
    Computes the entropy-ESS from the list of singular values.

    Args:
        - Ss: List of singular values.
        - clip: clips probabilities below this value avoiding numerical
    instabilities when the probabilities are too numerically close to 0.

    Returns:
        - entropy-ESS
    """
    ranks = []
    for SV in Ss:
        p = SV/SV.sum(dim=-1)[..., None]
        p = torch.clip(p, clip)
        H = -torch.sum(p * torch.log(p), dim=-1)
        rank = torch.exp(H)
        ranks.append(rank)
    ranks = torch.stack(ranks, dim=-1)
    return ranks
```

Example usage (Python-pseudocode):

```python
>>> out = model(u, output_attentions=True)
>>> # T shape: [bs, layers, heads, len, len]
>>> T = out.attention_matrix
>>> Ss = T2Ss(T) # List of singular values
>>> # ESS shape [bs, layers, heads, len-1]
>>> ESS = Ss2ToleranceESS(Ss, tol=1e-3)
>>> mean_ESS = torch.mean(ESS)
```

We note that calculating the effective rank may cause numerical instability when p_i^m approaches 0 due to the logarithmic term. This is partially mitigated by clipping the normalized singular values as shown above.

C.2 Formulation of the Featurizers

Linear attention and state-space model equivalence. We begin by demonstrating that linear attention models are state-space models, serving as the foundation for the subsequent formulation of featurizers for other models, such as gated linear attention and weighted linear attention.

A single linear attention head with dimension d/h, typically formulated as

$$y = qk^T v, \tag{C.2.1}$$

in which $q, k, v \in \mathbb{R}^{\ell \times d/h}$ are input features, can be reformulated as a recurrent model (Katharopoulos et al., 2020):

$$\begin{aligned} s_i &= s_{i-1} + k_i v_i^T \\ y_i &= q_i^T s_i, \end{aligned} \tag{C.2.2}$$

where the recurrent state is matrix-valued $s_i \in \mathbb{R}^{d/h \times d/h}$. Without loss of generality, applying column-major flattening to the matrix-valued state and treating v_i as the input u_i, the recurrence can be formulated as in Equation (1), where $A_i = I_{(d/h)^2}$, and B_i and C_i are constructed as follows:

$$B_{i-1} = \begin{bmatrix} k_i^1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ k_i^{d/h} & 0 & \cdots & 0 \\ 0 & k_i^1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & k_i^{d/h} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & k_i^1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & k_i^{d/h} \end{bmatrix} \quad C_i = \begin{bmatrix} q_i^1 & \cdots & q_i^{d/h} & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & q_i^1 & \cdots & q_i^{d/h} & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & q_i^1 & \cdots & q_i^{d/h} \end{bmatrix} \tag{C.2.3}$$

Notice that the recurrence is SISO, as there is no channel mixing within the recurrence itself. Additionally, each input/output channel has a state-size of d/h.

Now that we have established the equivalence between linear attention and state-space models in the form of Equation 1, we proceed with the formulation of the remaining featurizers.

Formulation of the featurizers. To characterize the "values" feature in attention-based models, we additionally show formulations for the "input-featurizer", $f_u(u)$, which is applied to the input of the recurrence as follows:

$$\begin{aligned} s_{i+1} &= A_i s_i + B_i f_u(u_i) \\ y_i &= C_i^T s_i + D_i f_u(u_i). \end{aligned} \tag{C.2.4}$$

Note that this is simply for the sake of completeness, and is not necessary for the study of effective state-size.

The following lists the formulations of the recurrent featurizers studied in this paper:

- Gated Linear Attention (GLA):

$$\begin{aligned} A_{i-1}^k &= \text{diag}(\text{sigmoid}(W_{A_2}^k W_{A_1} u_i)^{1/\beta}), \\ B_{i-1}^k = W_B^k u_i, \quad C_i^k &= W_C^k u_i, \quad f_u(u_i) = W_u^k u_i, \end{aligned} \tag{C.2.5}$$

  where $W_{A_1} \in \mathbb{R}^{16\times d}$, $W_{A_2}^k \in \mathbb{R}^{d/h\times 16}$, and $W_C^k,\ W_B^k,\ W_u^k \in \mathbb{R}^{d/h\times d}$. d and h represent the number of channels and heads, respectively. Each channel $c \in [d]$ is grouped into heads, where the head index corresponding to the channel is given by $k = \lfloor ch/d \rfloor$, and within the same head, the recurrent dynamics are shared across each channel. By default, β is set to 16.

- Weighted Linear Attention (WLA):

$$\begin{aligned} A^k &= \text{diag}(\text{sigmoid}(\hat{A}^k)^{1/\beta}), \\ B_{i-1}^k = W_B^k u_i, \quad C_i^k &= W_C^k u_i, \quad f_u(u_i) = W_u^k u_i, \end{aligned} \tag{C.2.6}$$

  where W_C, W_B, and W_u are identical to those in GLA, and $\hat{A}^k \in \mathbb{R}^{d/h}$ is explicitly parameterized and initialized to 0.

- Linear Attention (LA):

$$A^k = I, \quad B_{i-1}^k = \text{RoPE}(W_B^k u_i), \quad C_i^k = \text{RoPE}(W_C^k u_i), \quad f_u(u_i) = W_u^k u_i, \tag{C.2.7}$$

  where W_C, W_B, and W_u are identical to those in GLA, and A is a fixed identity matrix. Rotational positional encoding (RoPE) is by default applied to the B and C projections (Su et al., 2023).

- Softmax Attention (SA):

$$\begin{aligned} \hat{B}_i^k = \text{RoPE}(W_B^k u_i), &\quad \hat{C}_i^k = \text{RoPE}(W_C^k u_i), \\ T^k = \text{softmax}(\hat{C}^k (\hat{B}^k)^T), &\quad f_u(u_i) = W_u^k u_i, \end{aligned} \tag{C.2.8}$$

where T can be converted into a recurrence using the trivial realization in Equation B.2.5. W_C, W_B, and W_u are identical to those in GLA. We note that this results in the TSS of each channel in SA growing solely as a function of sequence length ($n_i = i$). Rotational positional encoding (RoPE) is by default applied to the $\hat{B}$ and $\hat{C}$ projections (Su et al., 2023).

- S6 (Gu & Dao, 2024):

$$\begin{aligned} \Delta^c = \text{softplus}(W_\Delta^c u_i + b^c), \quad A_{i-1}^c = \text{diag}(\exp(-\hat{A}\Delta^c)), \\ B_{i-1}^c = \Delta^c W_B u_i, \quad C_i = W_C u_i, \end{aligned} \tag{C.2.9}$$

where $\hat{A} \in \mathbb{R}^n$ is initialized to $[1 \quad 2 \quad \cdots \quad n]^T$, c is the channel index, $W_C, W_B \in \mathbb{R}^{n\times d}$, and $W_\Delta^c \in \mathbb{R}^{1\times d}$.

- GLA-S6:

$$\begin{aligned} A_{i-1}^h = \text{diag}(\exp(-[1/\alpha \quad 2/\alpha \quad \ldots \quad n/\alpha]^T \odot \text{softplus}(W_{A_2}^h W_{A_1} u_i))), \\ B_{i-1}^h = W_B^h u_i, \quad C_i^h = W_C^h u_i, \quad f_u(u_i) = W_u^k u_i, \end{aligned} \tag{C.2.10}$$

GLA-S6 is similarly structured to GLA. It has the same channel grouping structure with "heads", and identical W_B, W_C, and W_u projections. However, the A matrix is featurized using the `arange` term like in S6.

C.3 Empirical Validation

Here, we provide details on the task-model sweep presented in Section 3. Table 1 lists the hyperparameters that were exhaustively swept across to generate the task-model space. Note that the hyperparameter controlling the task difficulty is task dependent (for more details, see Poli et al. (2024)).

For the MQAR and selective copying tasks, a default vocab size of 8192 (Arora et al., 2023) was used for all models. For the compression tasks, the vocab size was varied to modulate task difficulty as shown in Table 1. Any other task settings not specified here are defaulted to those presented in Arora et al. (2023). Two important constraints on the tasks from Arora et al. (2023) which we also utilize in our experiments are as follows: MQAR task requires that

$$4 * \text{num kv pairs} \leq \text{seq len}$$

and the selective copying task requires that

$$2 * \text{num tokens to copy} + 1 < \text{seq len}$$

Any of the task configurations from Table 1 that violate these conditions were not trained. This is why the SA plot in Figure 3 has empty spots in the grid.

Finally, we note that all architectures analyzed here consist of 4 layers: 2 sequence mixing layers (i.e. one of GLA, LA, WLA or SA) and 2 channel mixing layers (i.e. MLPs).

Configuration	Value(s)
Tasks	MQAR, selective copying, compression
Num. key-value pairs	8, 16, 32, 64, 128
Num. tokens to copy	8, 16, 32, 64, 128
Vocab size (compression)	8, 16, 32, 64, 128
Vocab size (MQAR and selective copying)	8192
Sequence length	64, 128, 256, 512, 1024, 2048
Model (featurizer)	GLA, LA, WLA, SA
Model width	64, 128, 256, 512
Number of heads	4, 8
Optimizer	AdamW
Learning Rate	0.002
Weight Decay	0.1
Batch Size	64
Epochs	70
Steps Per Epoch	2000
Num. Training Samples	128k
Num. Testing Samples	6.4k

Table 1: Set of hyperparameters for task-model sweep.

Regarding the post-hoc analysis performed on the sweep, we note the following:

- Since the average TSS computed over the channels (which equals $\frac{\text{model width}}{\text{number of heads}}$ for GLA, LA, and WLA) explains more meaningful variation with respect to memory utilization than model width and number of heads individually, we consolidate those two dimensions into one by analyzing across the average TSS axis. For SA, since average TSS is a function of the task rather than model hyperparameters (see Equation B.2.5 and Section C.2), we instead compute the sum of TSS over all d channels, given by the total TSS per layer $= d*i$. In any cases where the average/total qualifier is not specified, note that we are referring to the average ESS or TSS.
- Since we analyze the recurrent models across the average TSS dimension, we compute average ESS in the plots presented in Section 3.1 in order to compare ESS and TSS as proxies for performance. Similarly, since we analyze the SA models across the total TSS dimension, we compute total ESS for those plots. However, we note that plots for both the average/total ESS and TSS are presented in Section D.1.
- When we marginalize across dimensions, we average across all models in that bucket of task-model space. For example, in Figure 3, for each (TSS, kv) pair, we average over the correlations of all models that correspond to that pair. Note, however, that we never average across tasks (i.e. MQAR, selective copying, compression) or featurizers (i.e. GLA, LA, WLA, SA).
- When we compute cross-model correlations (Figure 2a) for SA, we filter out models which have an accuracy > 0.95. This is done in order to observe meaningful variation as a function of (total ESS)/kv and (total TSS)/kv since many of the SA models obtain an accuracy of 1.
- When we compute within-model correlations (Figure 3) for MQAR, we drop epoch 0 from the computation since we observe a phase at the start of training in which ESS tends to decrease but accuracy does not change. We elaborate on this phenomenon in Section D.1 and hope to characterize it further in future work.
- Regarding the task-adjusted forms of ESS and TSS which, in the case of MQAR, are computed by normalizing the raw ESS value by the number of kv-pairs in the task, we note that this normalization factor is critical for observing the cross task-model correlations

presented in Figure 2a. In particular, in Figure 7, we find that correlations across the task-model space break down when examining the unnormalized ESS. This points to the higher level notion that ESS is expected to scale with the memory demands of the task.

- We interpret the state utilization of a model, which is given by ESS/TSS, as a proxy for what portion of the memory capacity of the network is realized in practice. By definition, state utilization takes on values ranging continuously from 0 to 1. Recall that a state utilization near 1 is indicative of state saturation.
- While for most of the ESS analysis conducted on the sweep we use the entropy ESS, we note that for the state utilization plot presented in Figure 3b, we use the tolerance ESS with a tolerance level set at 1e-3. We do this because we find that entropy ESS fails to capture the state collapse phenomenon. This is because state collapse is primarily dictated by the magnitude of the singular values as opposed to the relative decay rate of the entire spectrum. In particular, if all of the singular values are close to 0, the layer is likely failing to learn an expressive state, resulting in poor performance. Due to the normalization applied to the spectrum, the entropy ESS metric may potentially present this state as having high effective rank; however, in practice we know that this is a misrepresentation of the true dynamics. Tolerance ESS, in contrast, appropriately captures the dynamics of the state with respect to the norm of the operator. Because of this, whenever we analyze ESS as it pertains to state collapse (e.g. Figure 5a), we present the tolerance ESS instead.

C.4 ESS-Informed Featurizer Selection and Initialization Scheme

Configuration	Value
Model width	128
Num. heads	8
`arange` Norm. (α)[a]	1000
Logit Norm. (β)	16
K-expansion[b]	1

Table 2: Default GLA hyperparameters.

Configuration	Value
Model width	128
State expansion (`d_state`)	16

Table 3: Default S6 hyperparameters.

[a]For GLA-S6.

[b]K-expansion is used to vary TSS in the featurizer experiments.

Configuration	Value
Sequence length	2048
Num. KV Pairs	128
KV Dist. Const.	0.1
Optimizer	AdamW [a]
Learning Rate	0.002
Weight Decay	0.1
Batch Size	64
Epochs	70
Steps Per Epoch	2000
Num. Training Samples	128k
Num. Testing Samples	6.4k
Vocabulary Size	8192

Table 4: Default MQAR task settings employed throughout the featurizer and initialization experiments in Section 4.1.

[a]Loshchilov & Hutter (2019)

C.5 ESS-Informed Regularization

We use the following MQAR configuration for the regularization experiments presented in Section 4.2.

Configuration	Value
Sequence length	4096
Num. KV Pairs	128
KV Dist. Const.	0.1
Optimizer	AdamW [a]
Learning Rate	0.002
Weight Decay	0.1
Batch Size	64
Epochs	70
Steps Per Epoch	2000
Num. Training Samples	128k
Num. Testing Samples	6.4k
Vocabulary Size	8192
Model width	128
Num. heads	8

Table 5: MQAR task settings and model hyperparameters employed throughout the mid-training experiments in Section 4.2.

[a] Loshchilov & Hutter (2019)

Regarding the regularization scheme itself, since we examine models with two sequence mixing layers, we explore the following strategies: regularizing both layers, only regularizing the first layer and only regularizing the second layer. Empirically, we find that only regularizing the second layer performs the best and is thus the result presented in Figure 5b. We elaborate on why this is the most successful strategy in Section D.3.

C.6 ESS-Informed Model-Order Reduction

The teacher models used in the distillation experiments are 2 layer GLA models (Yang et al., 2024a) with dimension = 128 and TSS = 256 (num_heads = 8 and expand_k = 16). We checkpointed the models every 10 epochs while training on MQAR across different task difficulties. The task ranges are given as follows:

- Sequence length: [512, 1024, 2048]
- Number of Key-Value Pairs: [64, 128]

Other settings follow the defaults shown in Table 4. For each task difficulty pair, we repeated the training run with three different seeds. For each teacher model checkpoint, both layers were distilled independently with student models of different state-sizes (16, 32, 64, and 128). Distillation settings are shown in Table 6.

The ESS metric in Figures 5c, 28, and 29 was computed by taking the minimum across input samples and model channels, evaluated at the mid-point of the sequence ($\ell/2$). Using the mid-point of the sequence as a summary statistic was done in order to save compute. The midpoint in particular was chosen as it is the point in the sequence at which H_i has the greatest dimensions, retaining the largest amount of information from the original operator. Other approaches such as taking the maximum or average across the sequence also show similar trends, but we found taking the minimum to be the clearest.

Configuration	Value
Optimizer	AdamW
Batch Size	1
Learning Rate	0.001
Weight Decay	0.0
Training Steps (Operator)	800
Dropout (Operator)	0.2
Training Steps (Activation)	3200
Dropout (Activation)	0.2

Table 6: Distillation settings used for the results presented in Section 4.3.

C.7 ESS Analysis for Hybrid Networks

In our ESS analysis applied to hybrid networks, we restrict our scope to GLA-SA hybrids. In particular, we explore the following two settings:

- 8 layer hybrid networks in which 4 layers are sequence mixers (i.e. one of GLA or SA) and 4 layers are channel mixers (i.e MLPs). We exhaust all possible hybrid networks (of which there are 16) and perform post-training, per-layer ESS analysis on the networks. We train these hybrid models on MQAR with task-model settings given below in Table 7.
- 16 layer hybrid networks in which 8 layers are sequence mixers (i.e. one of GLA or SA) and 8 layers are channel mixers (i.e MLPs). Here, we explore all combinations of hybrid networks that follow the Jamba hybridization policy (Lieber et al., 2024) and perform post-training, per-layer ESS analysis on the networks. We train these hybrid models on MQAR with task-model settings given below in Table 8.

Configuration	Value
Sequence length	2048
Num. KV Pairs	512
KV Dist. Const.	0.1
Optimizer	AdamW [a]
Learning Rate	0.002
Weight Decay	0.1
Batch Size	64
Epochs	70
Steps Per Epoch	2000
Num. Training Samples	128k
Num. Testing Samples	6.4k
Vocabulary Size	8192
Model width	64
Num. heads	4

Table 7: Default MQAR task settings employed throughout the hybridization experiments conducted in the first setting described above.

[a] Loshchilov & Hutter (2019)

Configuration	Value
Sequence length	4096
Num. KV Pairs	1024
KV Dist. Const.	0.1
Optimizer	AdamW [a]
Learning Rate	0.002
Weight Decay	0.1
Batch Size	64
Epochs	70
Steps Per Epoch	2000
Num. Training Samples	128k
Num. Testing Samples	6.4k
Vocabulary Size	8192
Model width	16
Num. heads	2

Table 8: Default MQAR task settings employed throughout the hybridization experiments conducted in the second setting described above.

[a] Loshchilov & Hutter (2019)

Results for these experiments can be found in Section D.4.2.

C.8 State Modulation of Large Language Models

State modulation of open-weight models. The following randomly generated sentences were used to study the effects of separator tokens on state modulation in open-weights pre-trained language models.

<bos>*Mangoes are rich in vitamin C and can be blended into a refreshing smoothie*<sep> *Giraffes are the tallest mammals on Earth due to their long necks and legs*<sep> *She collects vintage typewriters from the 1940s*<sep> *Jupiter's Great Red Spot is a giant storm that has been raging for hundreds of years*<sep>

State modulation on custom-trained 1B models. For our custom-trained 1B language models, we used longer sentences, as state modulation patterns were less discernible with shorter sequences. A collection of randomly generated sentences is shown below:

<bos>*The deep blue ocean, teeming with an extraordinary array of marine life, from the smallest plankton to the largest whales, stretches out infinitely towards the horizon, a vast and mysterious expanse that has captivated the imaginations of explorers, scientists, and poets for centuries, hiding within its depths secrets yet to be discovered and stories yet to be told*<sep> *In a bustling city where skyscrapers tower over narrow streets filled with the constant hum of cars and the chatter of pedestrians, a small café, nestled between two imposing buildings, offers a quiet refuge for those seeking a moment of peace, with the comforting aroma of freshly brewed coffee and the soft sound of jazz music playing in the background, creating a cozy ambiance that feels like a world away from the urban chaos outside*<sep> *The ancient oak tree, with its gnarled branches stretching wide and its thick, sturdy trunk standing firm against the passage of time, has witnessed generations of families grow, seasons change, and countless stories unfold beneath its expansive canopy, becoming a silent guardian of the park, offering shade to those who seek solace and a sense of continuity in a rapidly changing world*<sep>

We note that the specific sentences and their order are not crucial to this analysis. Similar patterns have emerged with various sentence arrangements, provided the sentences are sufficiently long.

Training settings are outlined in Table 9.

Configuration	Value
Batch Size	16
Max Sequence Length	32k
Training Steps	160k
Optimizer	AdamW
Learning Rate	0.001
Weight Decay	0.1
Num. Layers	24
Dimension	2048

Table 9: 1B LLM settings.

The perplexity scores shown in Figure 6b were computed on 16k randomly sampled sequences over the FineWeb (Penedo et al., 2024) dataset. The raw perplexity samples were smoothed via a kernel density estimation method.

D Extended Experimental Results

D.1 Empirical Validation

In this section, we provide additional results and commentary from the sweep detailed in Section C.3 that were not presented in the main portion of the paper. One thing to note is that the most of the ESS results presented in Section 3 were computed using the entropy ESS. However, we also computed ESS using the tolerance-based approach to affirm that both forms of ESS showcase similar trends. In particular, we examined tolerances of 1e-1, 1e-3 and 1e-5. Since we observe similar trends across tolerances, we provide plots for a tolerance of 1e-3 below and omit the others for the sake of brevity.

D.1.1 State Collapse Continued

Here, we continue our discussion on the state collapse phenomenon presented in Section 3.2. In particular, while we assert that state collapse is observable across all TSS in the high kv bucket for GLA/WLA, Figure 3b shows that accuracy differences between LA and GLA/WLA are only evident in the high TSS/high kv bucket of the task-model space. This is because state saturation is acting as a confounder, worsening performance in LA (see Figure 3b when TSS is 8). Therefore, although state collapse in GLA/WLA does not result in worse performance than LA in this specific task-model setting, it remains an issue even for models with smaller states when trained on sufficiently difficult tasks. This is the motivation behind the task-model setting explored in Section 4.2.

D.1.2 Entropy ESS MQAR Results Continued

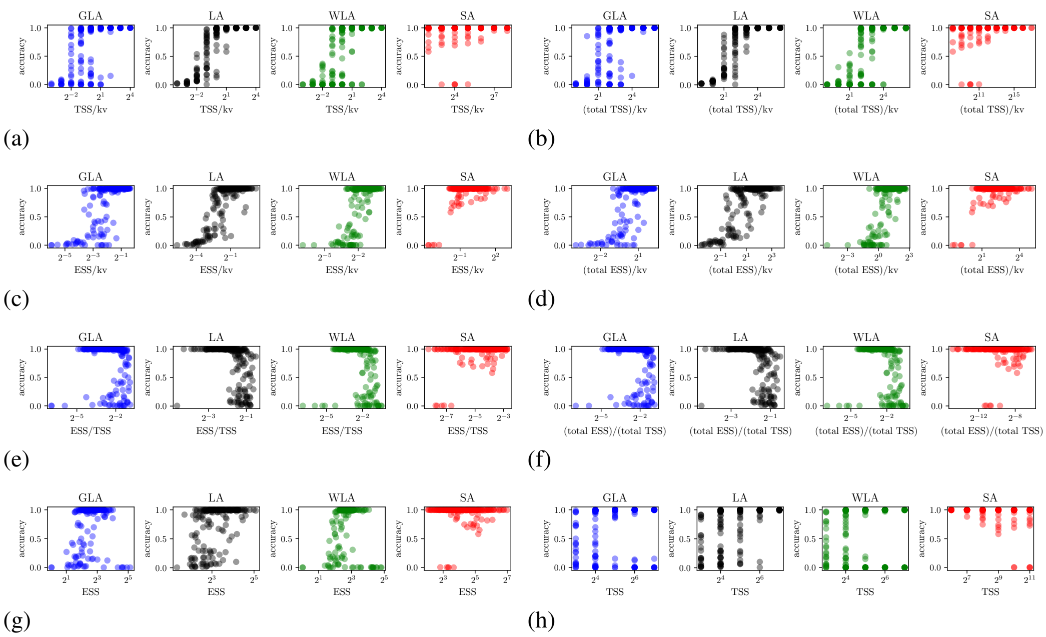


Figure 7: (a) TSS/kv vs accuracy across featurizers. This demonstrates that TSS/kv (i.e. memory capacity) is a worse proxy for model performance than ESS/kv as discussed in Section 3. (b) (total TSS)/kv vs accuracy across featurizers. This demonstrates that (total TSS)/kv is a worse proxy for model performance than (total ESS)/kv. (c) ESS/kv vs accuracy across featurizers. (d) (total ESS)/kv vs accuracy across featurizers. (e) ESS/TSS (i.e. state utilization) vs accuracy across featurizers. We note that models that saturate their state tend to perform worse on the task which is evidence of the state saturation phenomenon discussed in Section 3.2. The models that do not saturate their state but still perform poorly are the models that undergo state collapse. (f) (total ESS)/(total TSS) vs accuracy across featurizers. (g) ESS vs accuracy across featurizers. Note that without normalizing by kv (i.e. the task memory), the correlation with accuracy breaks down substantially. (h) TSS vs accuracy across featurizers.

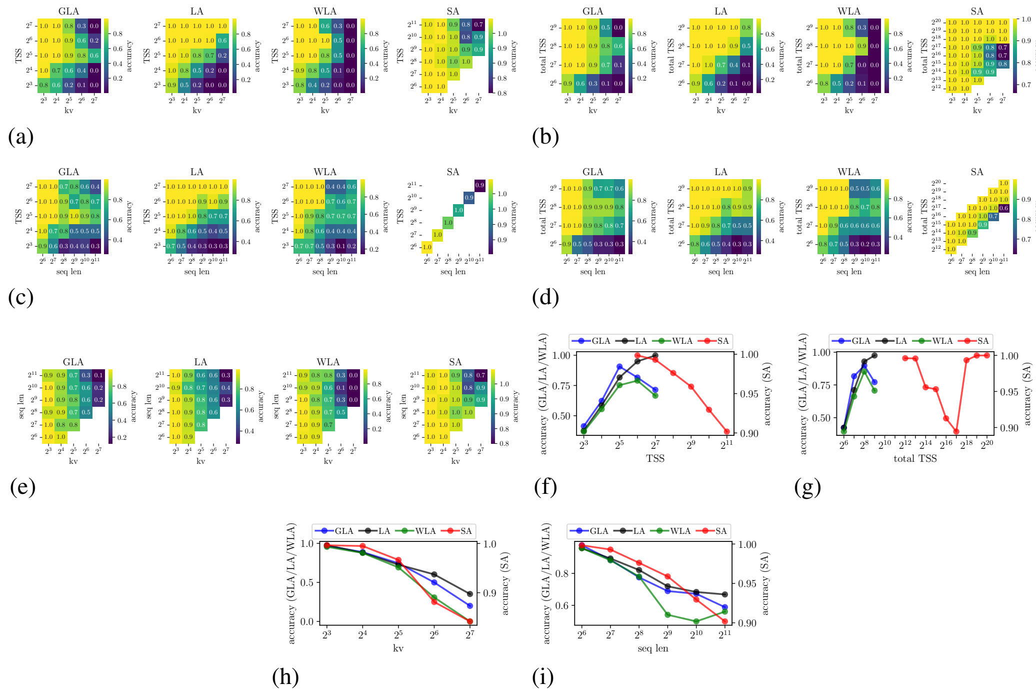


Figure 8: MQAR accuracies marginalized across different dimensions.

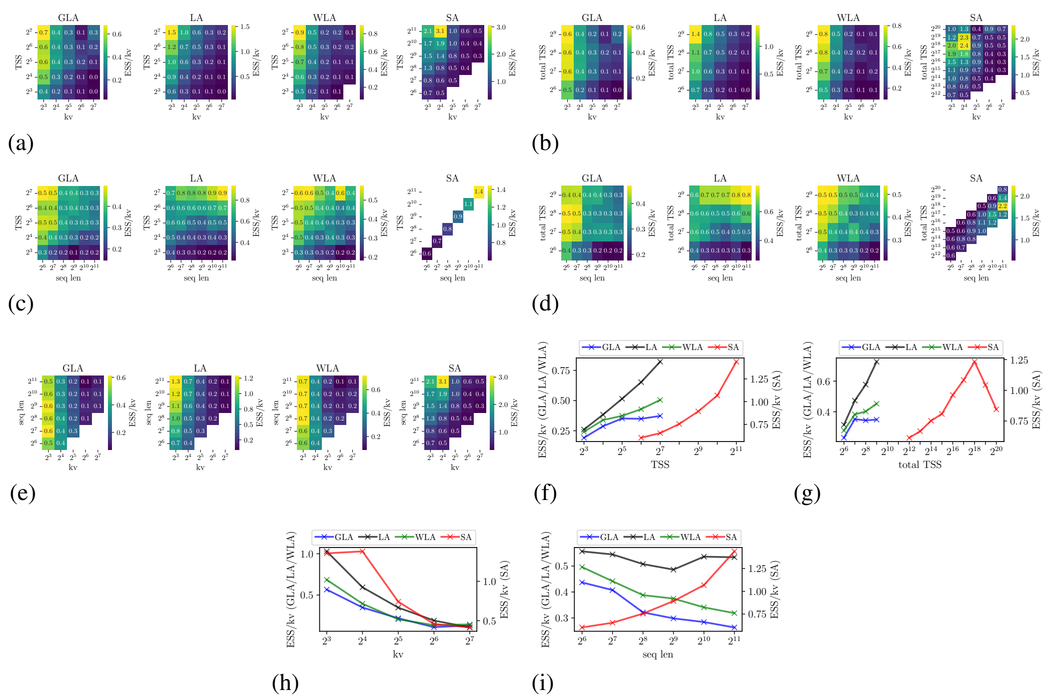


Figure 9: MQAR ESS/kv marginalized across different dimensions.

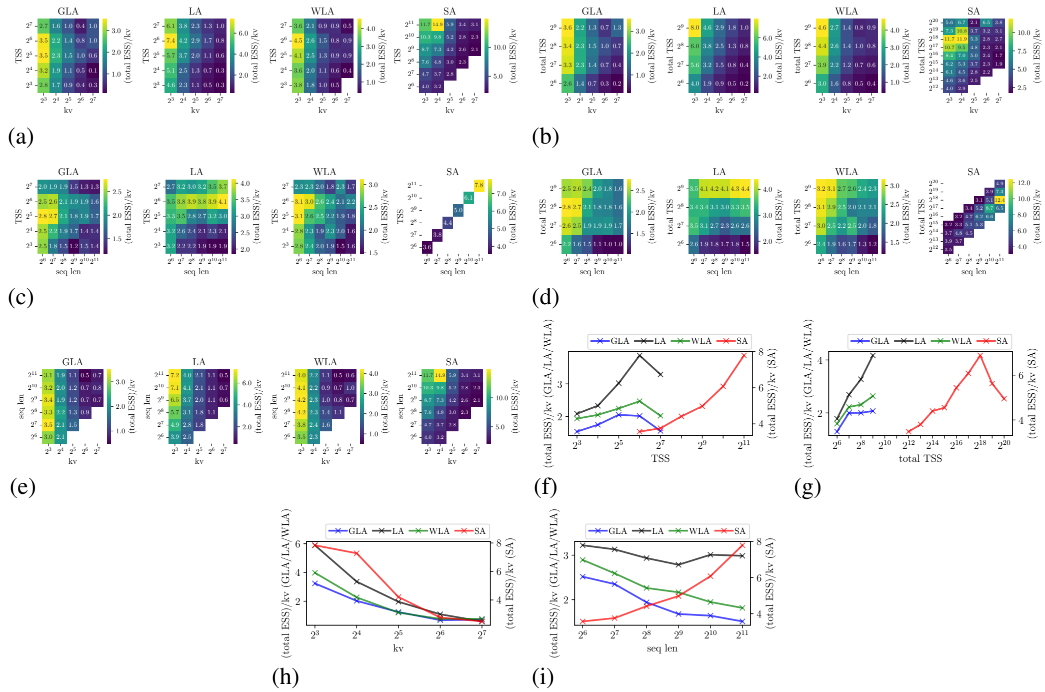


Figure 10: MQAR (total ESS)/kv marginalized across different dimensions.

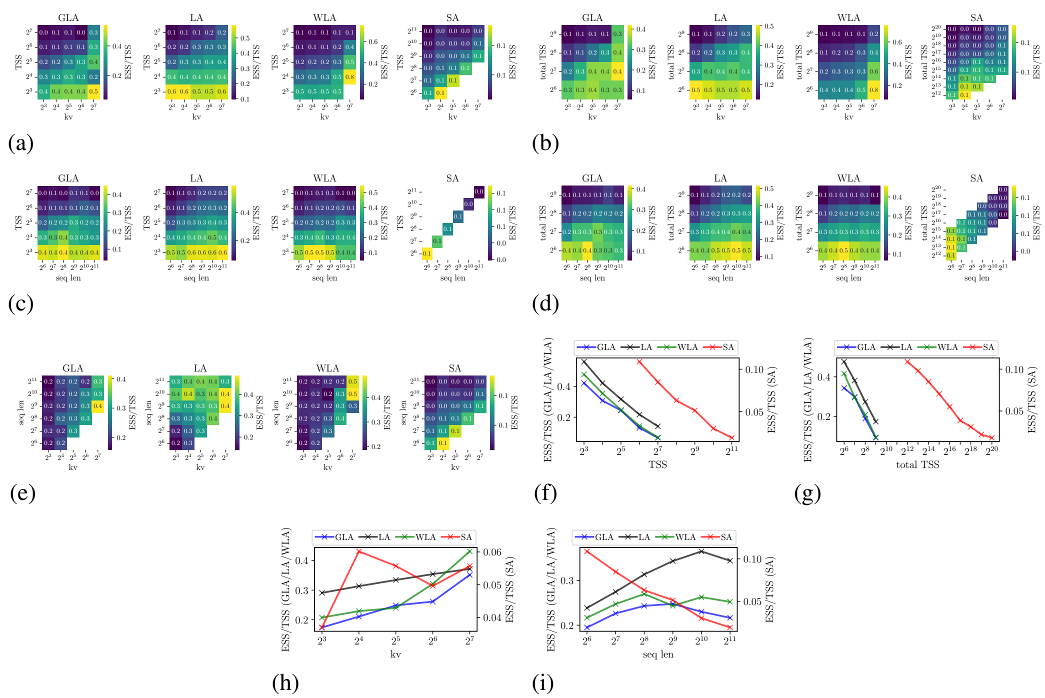


Figure 11: MQAR ESS/TSS marginalized across different dimensions.

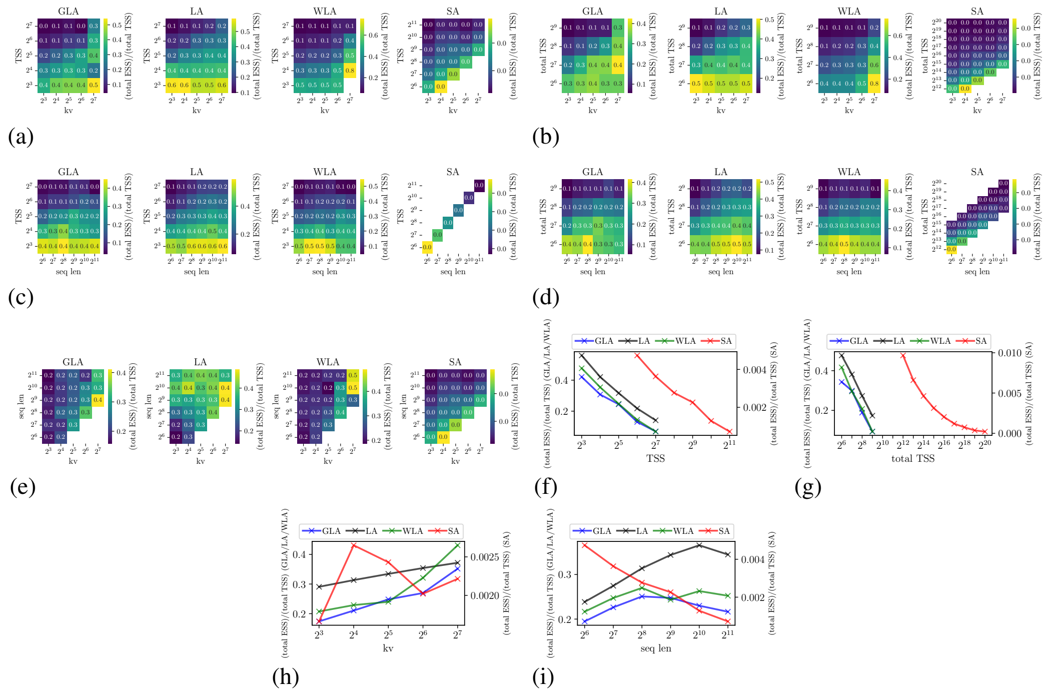


Figure 12: MQAR (total ESS)/(total TSS) marginalized across different dimensions.

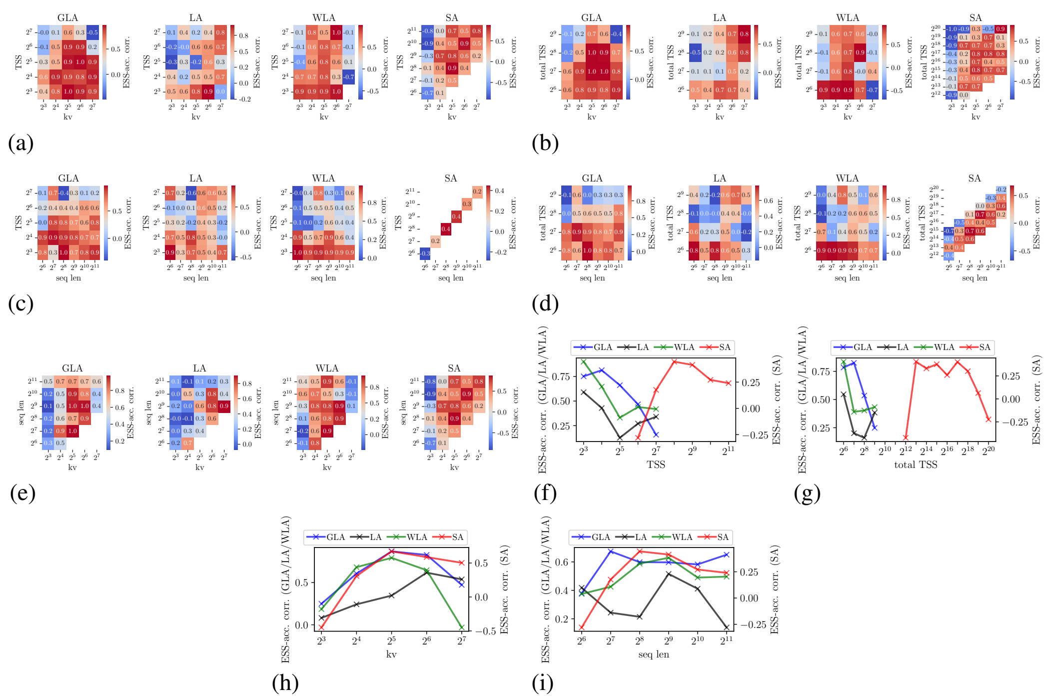


Figure 13: MQAR ESS-accuracy correlations computed over training marginalized across different dimensions.

D.1.3 Tolerance ESS MQAR Results

Below are plots from the MQAR sweep using tolerance ESS (tol=1e-3) instead of entropy ESS. We note that all of the prevailing trends remain the same.

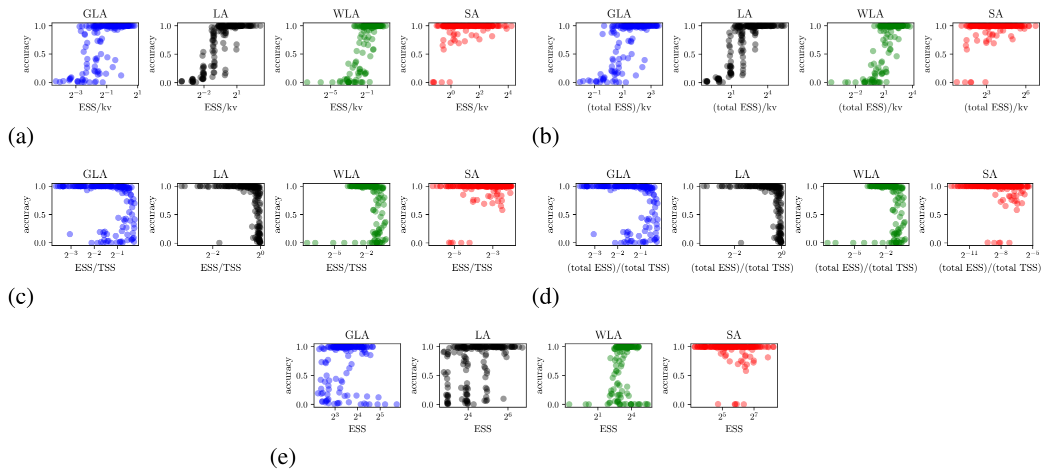


Figure 14: Accuracy vs various forms of tolerance ESS across task-model space. Plots are entirely analogous to those shown in Figure 7.

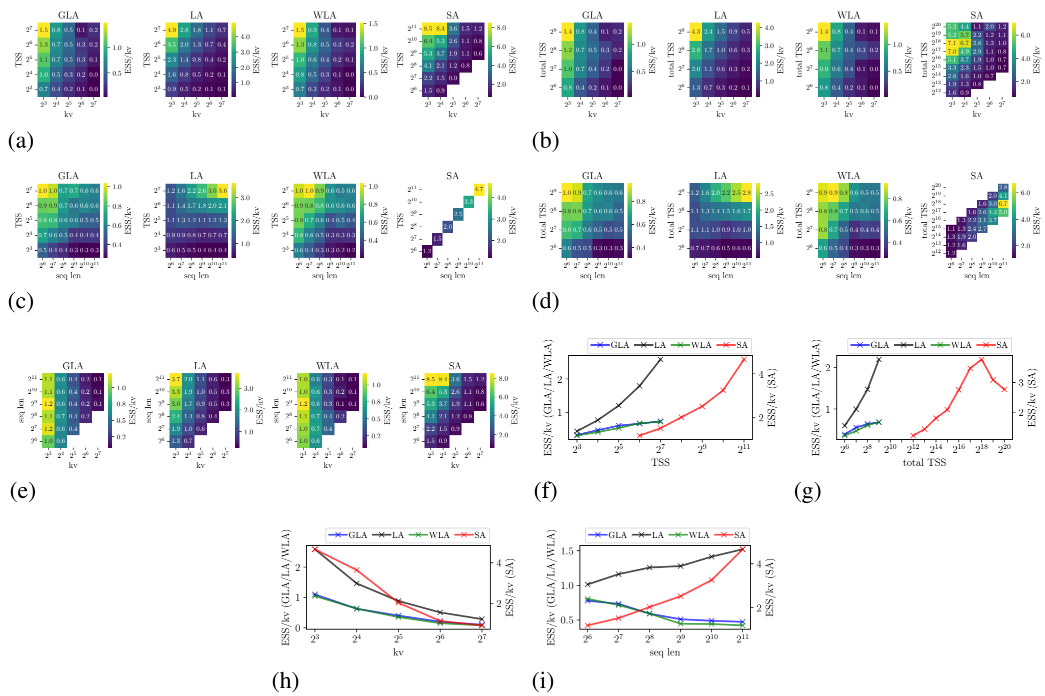


Figure 15: MQAR ESS/kv marginalized across different dimensions.

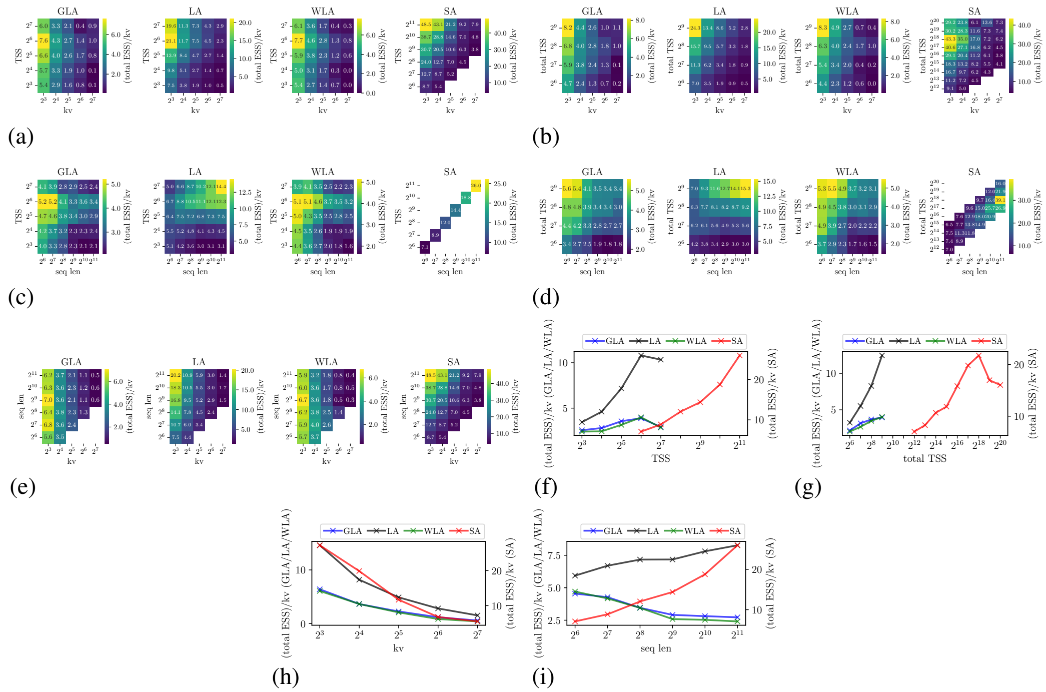


Figure 16: MQAR (total ESS)/kv marginalized across different dimensions.

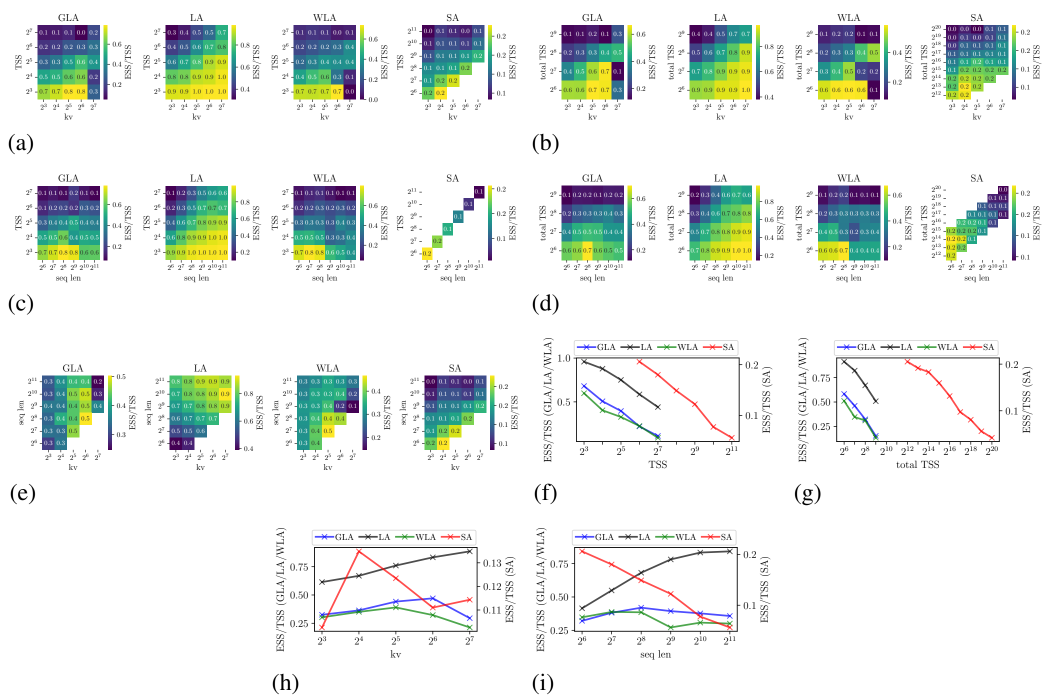


Figure 17: MQAR ESS/TSS marginalized across different dimensions.

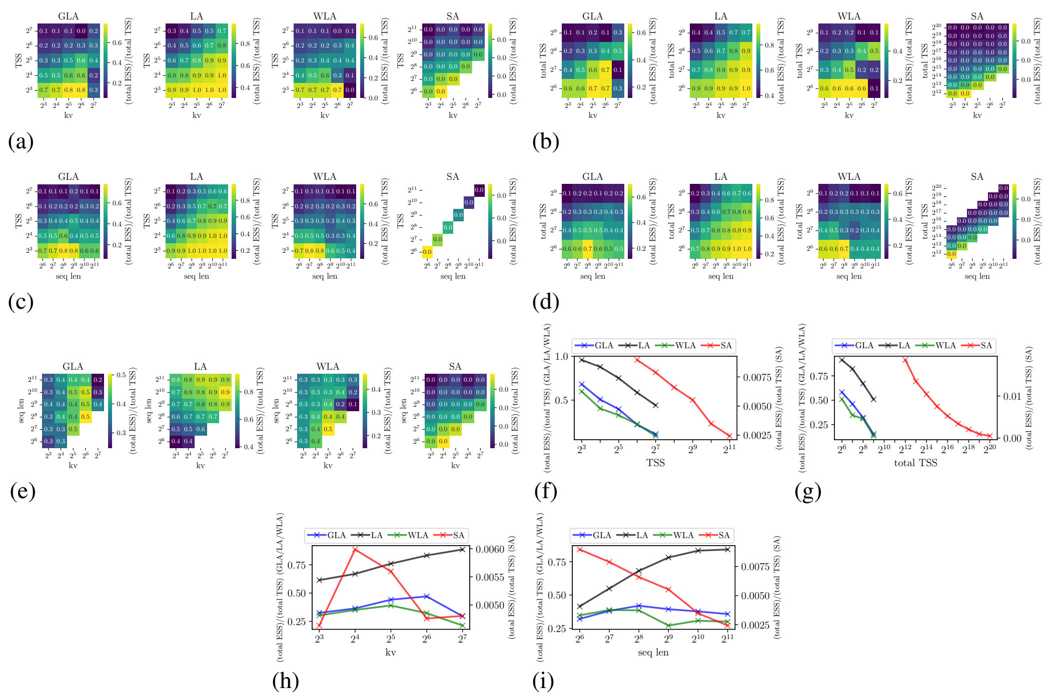


Figure 18: MQAR (total ESS)/(total TSS) marginalized across different dimensions.

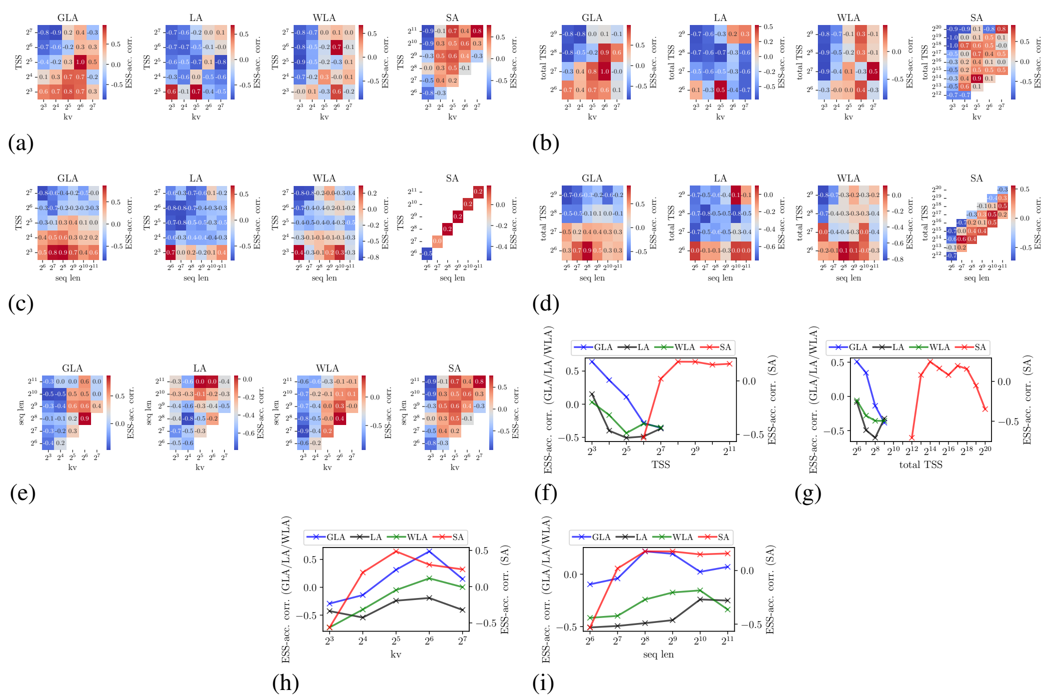


Figure 19: MQAR ESS-accuracy correlations computed over training marginalized across different dimensions.

D.1.4 SELECTIVE COPYING AND COMPRESSION RESULTS

Below, we present results for the selective copying and compression tasks, analogous to the ones presented in Section 3 on MQAR.

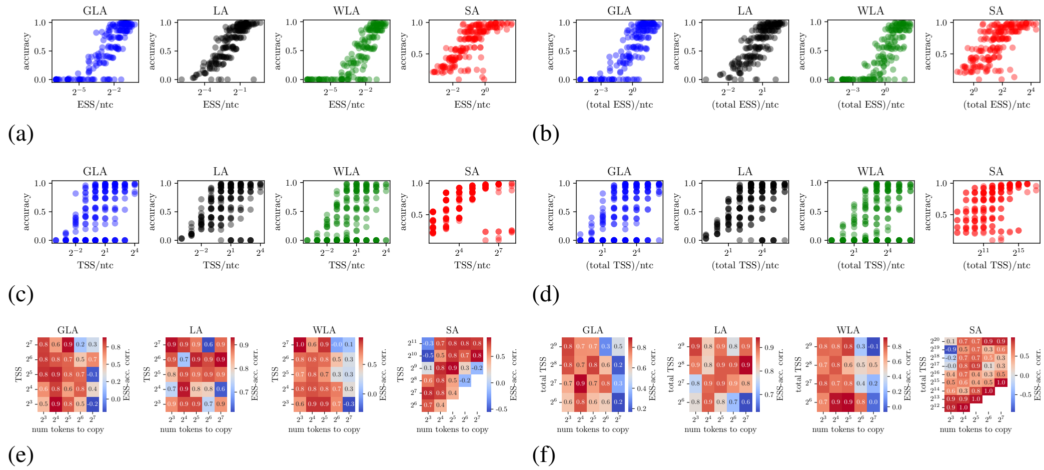


Figure 20: Selective copying results. Note that ESS here refers to entropy ESS and we abbreviate num. tokens to copy as ntc in plots above. (a) ESS/ntc vs accuracy across featurizers. (b) (total ESS)/ntc vs accuracy across featurizers. (c) TSS/ntc vs accuracy across featurizers. (d) (total TSS)/ntc vs accuracy across featurizers. (e) ESS-accuracy correlation computed over the course of training in (TSS, kv) buckets. (f) ESS-accuracy correlation computed over the course of training in (total TSS, kv) buckets.

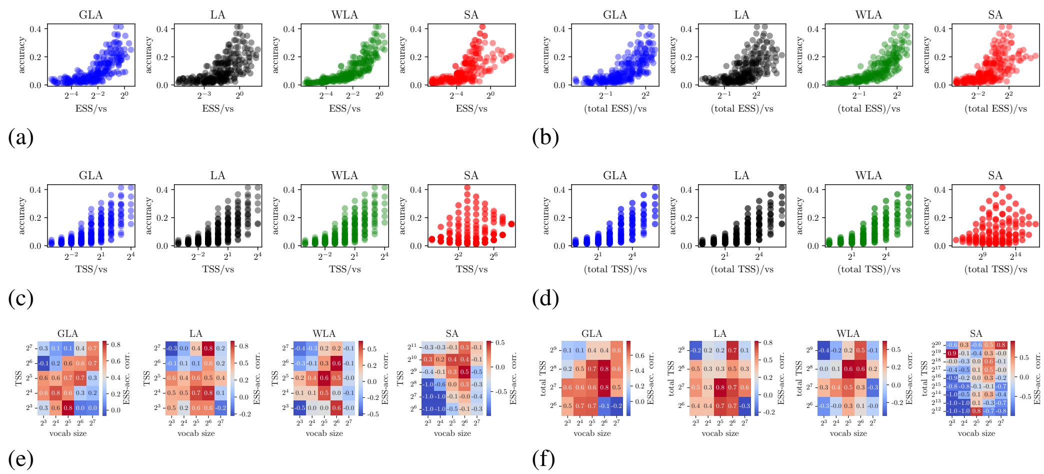


Figure 21: Compression results. Note that ESS here refers to entropy ESS and we abbreviate vocab size as vs in plots above. (a) ESS/vs vs accuracy across featurizers. (b) (total ESS)/vs vs accuracy across featurizers. (c) TSS/vs vs accuracy across featurizers. (d) (total TSS)/vs vs accuracy across featurizers. (e) ESS-accuracy correlation computed over the course of training in (TSS, kv) buckets. (f) ESS-accuracy correlation computed over the course of training in (total TSS, kv) buckets.

We note that with respect to the cross task-model trends, we find that in both selective copying and compression, task-adjusted ESS is a better proxy for model performance than task-adjusted TSS (Figures 20a, 20c, 21a, 21c). This is substantial as it demonstrates the utility of the ESS metric beyond just MQAR.

Regarding within task-model trends, we observe similar patterns for selective copying as those seen in MQAR (Figure 20e), with one notable distinction. Namely, ESS and accuracy are positively correlated across a larger portion of the task-model space in selective copying than in MQAR. For compression, however, the within task-model trends look a bit different than what we observe in selective copying and MQAR (Figure 21e). One potential reason for this is that the compression task is significantly more difficult than the MQAR and selective copying tasks (as noted by the lower accuracies in Figure 21a), leading to more instabilities over the course of training. But in any case, this does highlight the fact that the strength of ESS as a proxy for model performance changes as a function of the task. The precise nature of this relationship in something we hope to explore in future work.

D.1.5 ESS Training Dynamics in MQAR

As mentioned in Section C.3, we observe a phase at the start of training in MQAR in which ESS tends to decrease. This is shown in Figure 22 in which we select an arbitrary task-model configuration from the sweep and plot its ESS and accuracy over the course of training on a per featurizer basis.

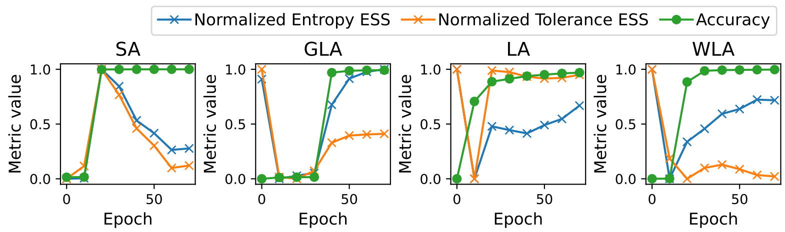
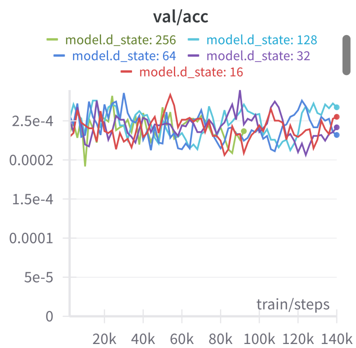


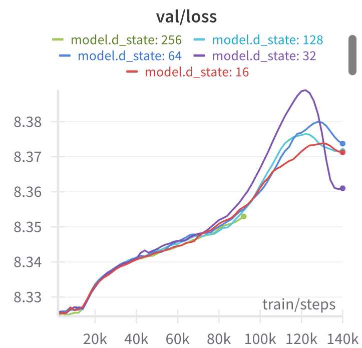
Figure 22: Training dynamics of ESS in select models (dmodel=256, heads=8) trained on MQAR (seqlen=2048, kv=64). We min-max normalize the ESS curves over the course of training to emphasize the shape of the curve as opposed to its magnitude. Note that the tolerance ESS shown here is computed using a tolerance of 1e-3.

We find that at the start of training (i.e. in between epochs 0 and 10), even if the accuracy is not evolving, the ESS is. In particular, in the recurrent frameworks (GLA, LA and WLA), we note a sharp decrease in the ESS before it begins to rise later in training (and along with it the model accuracy). In contrast, in SA we observe the opposite: a sharp increase at the start of training following by a steady decrease (even after it has solved the task). This points to a level of nuance in the training dynamics of MQAR ESS that we have yet to characterize and is something we hope to explore in future work.

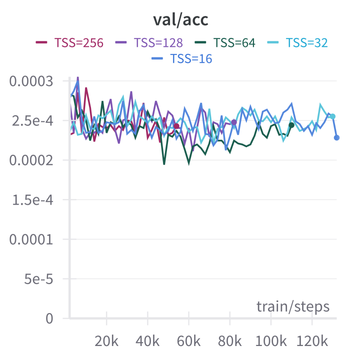
D.2 Initialization-Phase Analysis



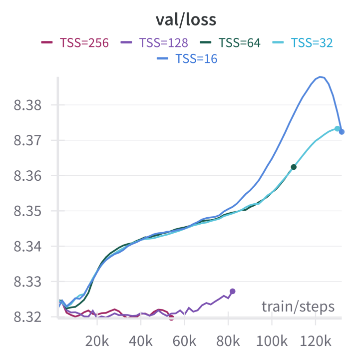
(a) Validation accuracy of S6



(b) Validation loss of S6



(c) Validation accuracy of GLA-S6



(d) Validation loss of GLA-S6

Figure 23: Loss curves of S6 and GLA-S6 showing that the models are unable to improve beyond random guessing on MQAR, across various state-sizes.

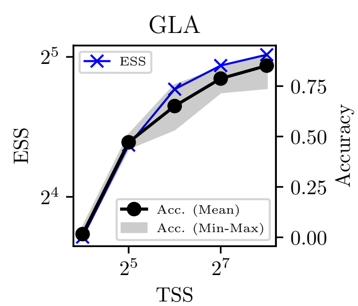


Figure 24: ESS and MQAR accuracy as a function of TSS on a custom task regime (sequence length = 1024, num. kv pairs = 256). This figure illustrates a strong correlation between MQAR accuracy, ESS and TSS.

D.3 Mid-Training Analysis

First, we provide some additional commentary on the ESS-based regularization results discussed in Section 4.2. Recall we showed that decaying the A matrices in GLA and WLA towards the identity matrix enables these models to outperform LA in the state collapse regime. Our intuition for this result is that by ameliorating state collapse, GLA and WLA can better leverage their increased expressivity, which stems from their learnable A matrices – a feature absent from LA.

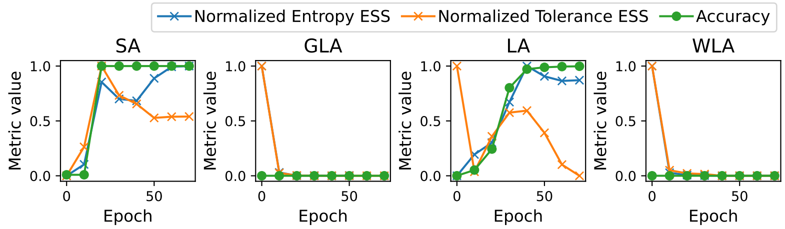


Figure 25: An example of the training dynamics of ESS in select models (dmodel=512, heads=4) trained on MQAR (seqlen=2048, kv=128) that undergo state collapse (i.e. GLA and WLA). We min-max normalize the ESS curves over the course of training to emphasize the shape of the curve as opposed to its magnitude. Note that the tolerance ESS shown here is computed using a tolerance of 1e-3.

Next, as mentioned in Section C.5, we provide some intuition behind the efficacy of regularizing only the second layer of the network as opposed to the first or both layers.

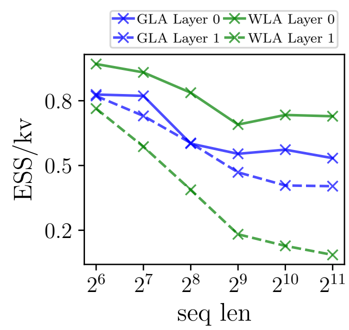


Figure 26: Per-layer ESS/kv as a function of MQAR sequence length for the GLA and WLA featurizers. ESS shown here is computed using a tolerance of 1e-3. Layers are 0-indexed.

Using 0-indexing for the layers, Figure 26 shows that layer 1 realizes a lower ESS/kv than layer 0, particularly in the case of WLA. This suggests that layer 1 contributes disproportionately to the observed state collapse (Figure 25); consequently, it makes sense that layer 1 would need to be regularized more heavily. Now, this begs the question as to why only regularizing the second layer leads to better performance than regularizing both layers (results of which were not shown). We have two possible hypotheses for this outcome. First, introducing regularization terms for both layers may complicate optimization by creating potentially conflicting objectives. Second, excessive decay of the A matrices towards the identity matrix may cause the model to revert back to the LA regime, which – as shown in Figure 5b – performs worse than GLA and WLA (when sufficiently regularized). Nonetheless, we hope to further explore this intuition and investigate other ESS-based forms of regularization in future work.

D.4 Post-Training Analysis

D.4.1 Model-Order Reduction

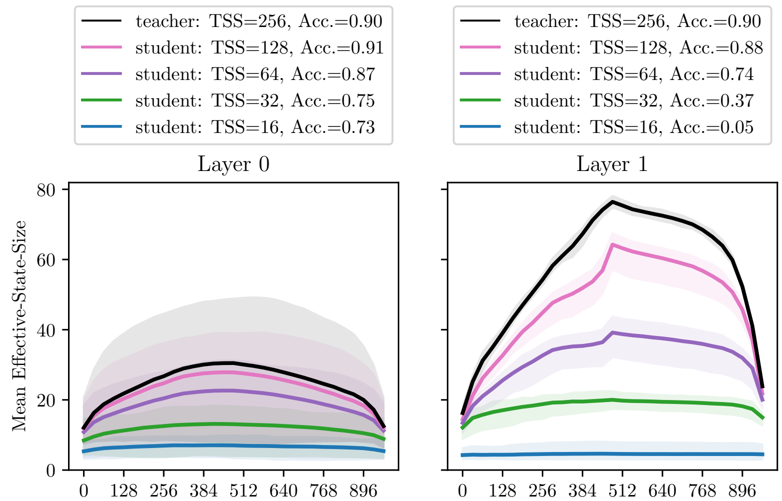


Figure 27: This figure compares MQAR accuracy and ESS across reduction scales for layers 0 and 1. The lower ESS in layer 0 of the teacher model leads to better downstream performance after distillation compared to distilling layer 1.

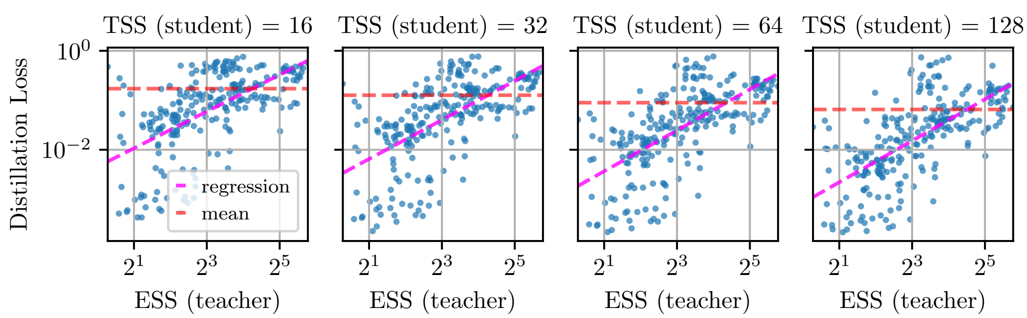


Figure 28: Correlation between ESS and distillation loss across multiple student TSSs (reduction ratios). The original teacher models have a TSS of 256.

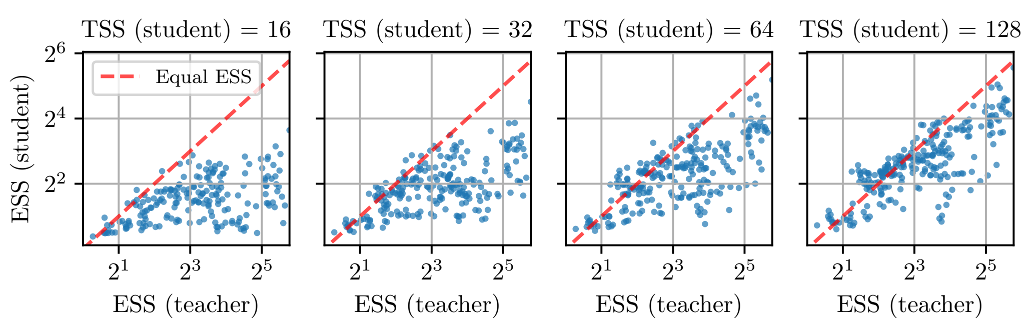


Figure 29: Teacher ESS vs distilled student ESS. As expected, we observe a clear trend: an increase in the student TSS results in the student's ESS more closely matching the teacher's ESS. Plots like these can help provide additional context during the distillation process.

D.4.2 HYBRIDIZATION

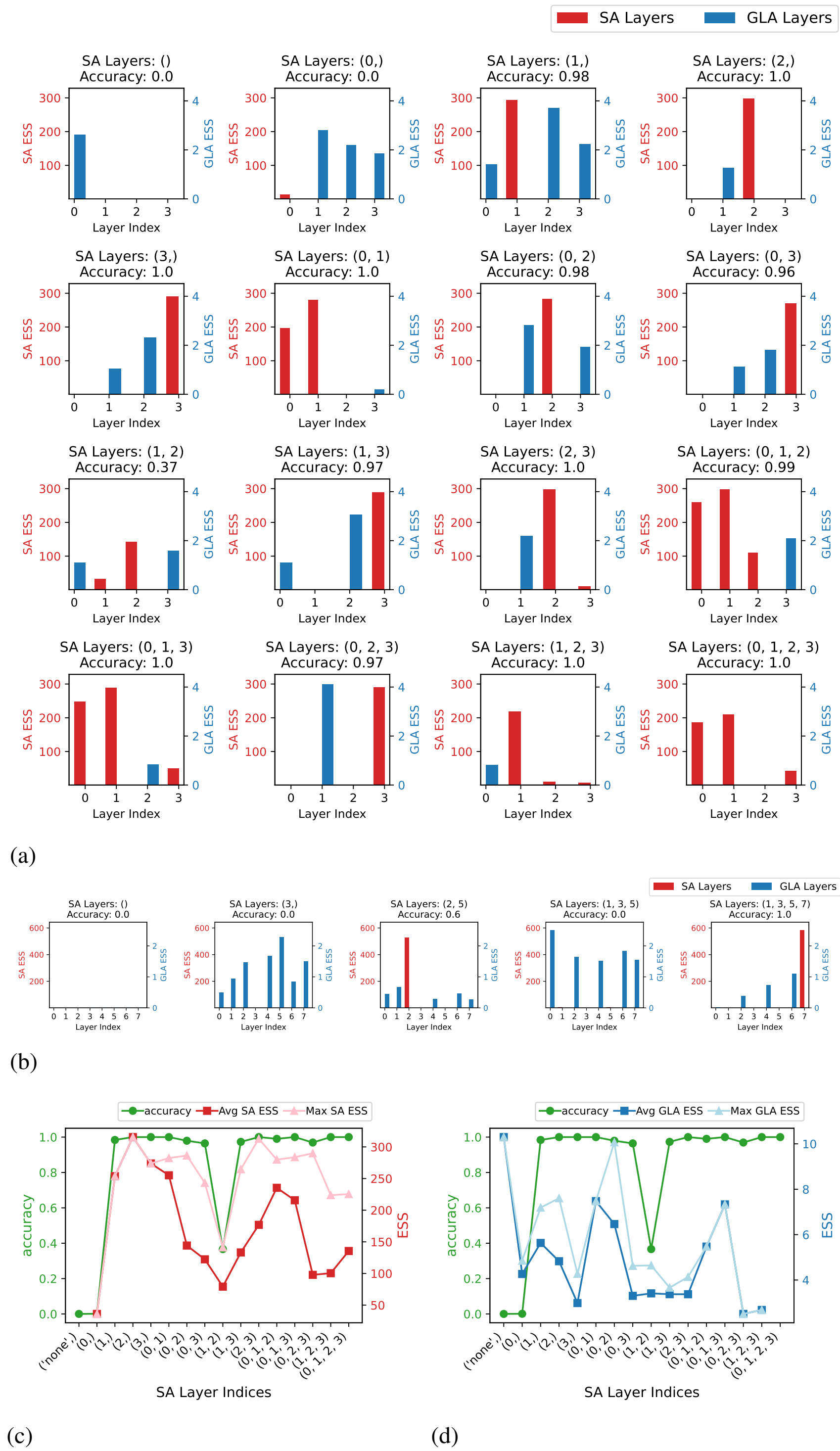


Figure 30: All results presented here are computed using tolerance-based ESS with a tolerance set at 1e-1. Network layers are 0-indexed. (a) Per-layer ESS of all possible 4-layer GLA-SA hybrid networks. Experimental settings can be found in Section C.7. (b) Per-layer ESS of all possible 8-layer GLA-SA Jamba-inspired hybrid networks. Experimental settings can be found in Section C.7. (c) Model accuracy and max/average ESS of SA layers in the 4-layer GLA-SA hybrid networks. (d) Model accuracy and max/average ESS of GLA layers in the 4-layer GLA-SA hybrid networks.

In this section, we present results from a post-training ESS analysis applied to GLA-SA hybrid networks to demonstrate the ability of ESS to capture differences among hybrid networks with varying topologies.

In the first experimental setting, we train all possible 4-layer GLA-SA hybrid networks and compute the per-layer ESS on each model. We use the tolerance-based ESS since we want to analyze failure modes of learning in hybrid networks. In Figure 30a, we first note that in the pure GLA model, many of the layers fail to learn expressive states (as evidenced by the tolerance ESS being 0), offering intuition as to why the model performs so poorly. Moving on to the hybrid networks with a single attention layer, we note that all of them perform quite well with the exception of the network which has attention in the first layer. Interestingly, we find that when attention is placed in the first layer, it suffers from state collapse. At a higher level, this substantiates why many state-of-the-art hybrid networks (such as Jamba) do not place attention as the first layer of the network. However, such hybrids are typically constructed purely on the basis of performance: here, ESS is able to provide a distinct perspective. Next, examining the hybrids with 2 SA layers, we find that the only poor performing topology is with attention placed in the second and third layers. Again, we find that the ESS of the attention layers is lower than what we observe in the hybrids that solve the task, indicating its usefulness as a proxy for performance beyond the 2-layer non-hybrid networks we explored in Section 3.

To clarify this, we examine the maximum/average ESS (computed across layers) of the SA and GLA layers separately to understand how each relates to model performance. Notably, we find that maximum ESS across attention layers best correlates with accuracy (Figure 30c). Interestingly, the average SA layer ESS is a worse proxy for performance, potentially indicating that having a single layer with high memory utilization in hybrid networks is more important than having many layers with lower memory utilization. This offers support as to why hybrid networks like Jamba have a 1:7 ratio between attention and non-attention layers. Regarding the GLA layers, we find that despite both the maximum and average SS varying across models, they do not correspond to changes in accuracy. One possible explanation for this is that since the attention layers are responsible for driving the total ESS of the network up due to their unbounded state size, the role of non-attention layers in hybrid networks may not be captured entirely by the magnitude of their ESS. Nonetheless, this is something we hope to explore in future work.

In the second experimental setting, we move beyond 4-layer GLA-SA hybrids to 8-layer GLA-SA hybrids. Here, instead of iterating over all possible topologies, we restrict the space of networks to those constructed via the hybridization policy proposed by Jamba. The Jamba hybridization policy takes in the number of layers as input and provides a particular hybrid topology as output (refer to Lieber et al. (2024) for more details). Since most topologies explored in the 4-layer setting solved the task, we both reduce the model dimension of the network and make the task more difficult to see if we can observe performance differences across the architectures (model settings can be found in Table 8). Unsurprisingly, we find that the pure GLA network is unable to solve the task and also realizes a tolerance-based ESS of 0 in all layers (Figure 30b). However, more interesting is the fact that while the 2 SA-layer Jamba hybrid partially learns the task, the 3 SA-layer does not. Examining the ESS shows that the attention layers in the 3 SA-layer hybrid suffer from state collapse which we know is highly correlated with poor performance on MQAR. This points to a deficiency of fixed-topology hybridization policies like Jamba which do not take into account factors like network trainability which can significantly influence model performance. Furthermore, this suggests that the ESS metric can be used to better inform the construction of hybrid networks. We hope to further elucidate these per-layer ESS trends and leverage these insights to construct novel ESS-informed hybridization policies in future work.

D.5 State Modulation of Large Language Models

State modulation patterns on various open-weight models are illustrated in Figures 31, 32, 33, 34, 35, and 36.

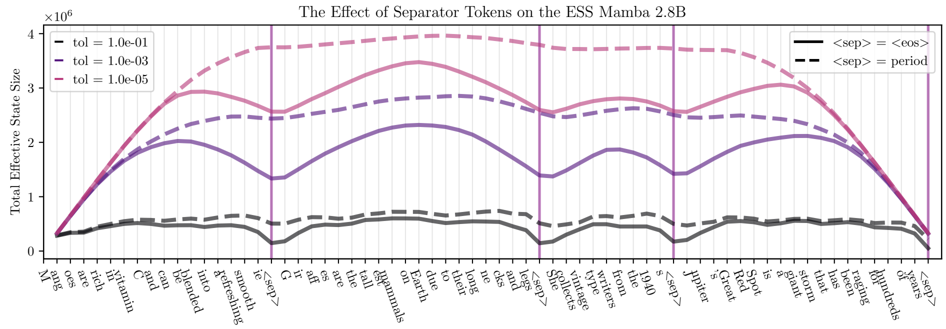


Figure 31

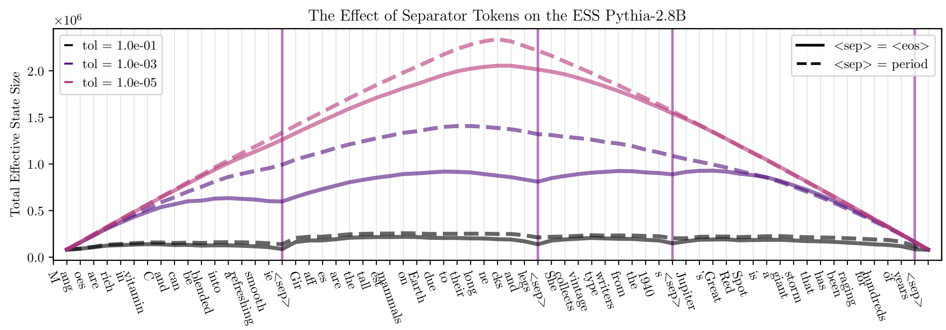


Figure 32

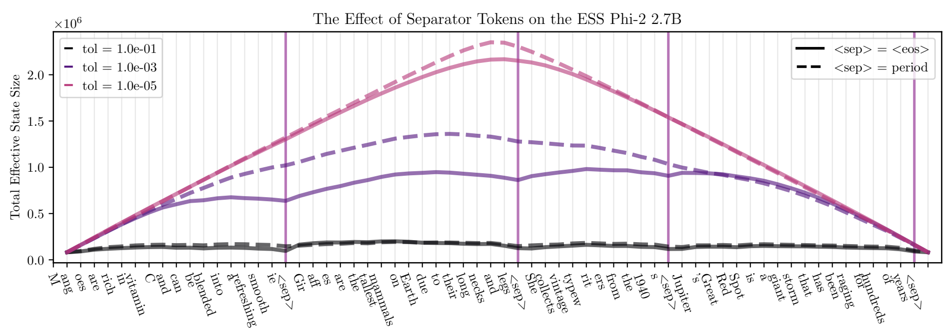


Figure 33

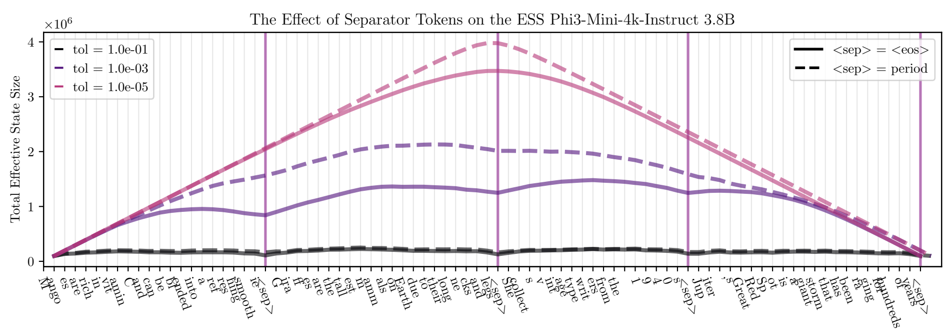


Figure 34

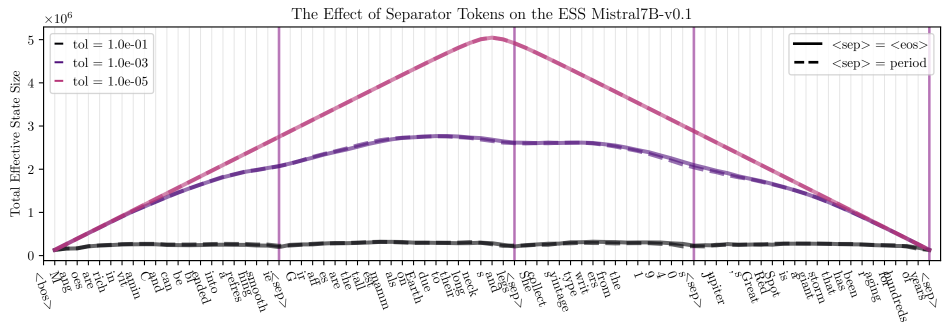


Figure 35

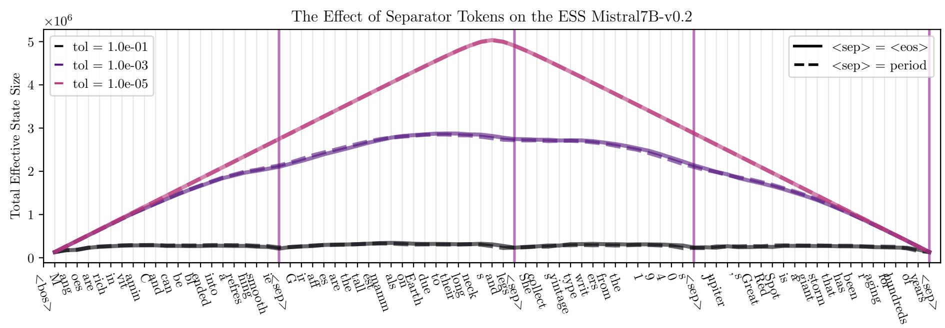


Figure 36

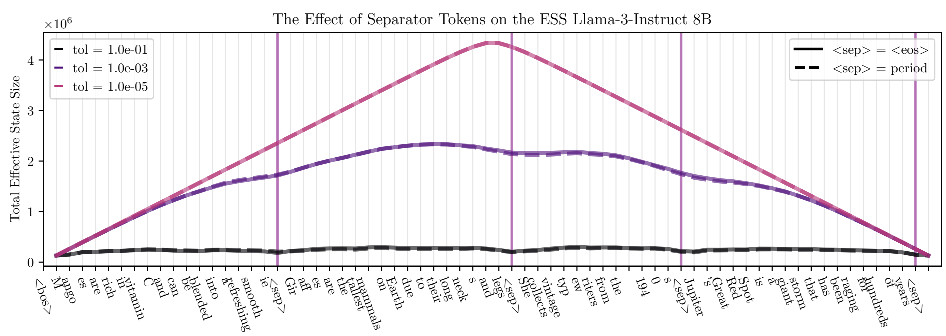


Figure 37

The figures above reveal significant cross-architectural differences in context processing. The attention-based model at a similar 7B scale (Figure 35, 36, and 37) shows minimal change in its ESS pattern when an EOS token is replaced with a period ("."). In contrast, the limited cache-size state-space model (Falcon Mamba 7B, Figure 6a) exhibits a substantial reduction in state modulation under the same token substitution.

We attribute this difference to a phenomenon we term "preemptive state modulation" in limited state-size models, which stems from fundamental architectural differences. State-space models (SSMs) with limited cache must efficiently manage their finite memory capacity and learn to preemptively modulate state-size to optimize information retention, relying on explicit signals like EOS tokens to trigger context resets. In contrast, attention models with linearly increasing cache can store all past information without the need for selective forgetting, do not require preemptive state modulation, and show less sensitivity to explicit demarcation tokens. This distinction highlights the different strategies employed by various model architectures in managing context across diverse inputs, potentially influencing their performance on tasks requiring long-range recall or context separation.

However, a subset of attention models demonstrated varying state modulation patterns in response to different separator tokens, with this effect being more pronounced in smaller model sizes (see Figure 32, 33, and 34). This phenomenon, while not consistent across all attention architectures, merits deeper exploration.

Figure 38 illustrates the state modulation patterns at different tolerance levels for the four 1B language models (LA, WLA, GLA, SA), trained under identical conditions.

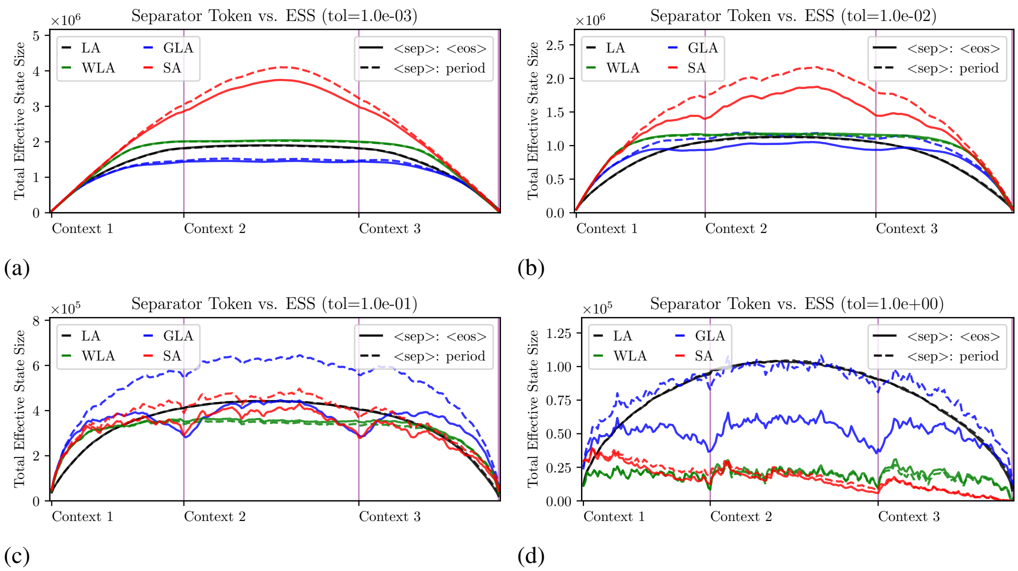


Figure 38: An illustration of the effect of different separator tokens over different layers across different tolerances. Softmax attention exhibits the most pronounced state modulation, beginning at a tolerance level of $1\text{e}{-2}$, followed by gated linear attention with significant modulation starting at a tolerance of $1\text{e}{-1}$. Weighted linear attention shows minimal modulation, only detectable at a tolerance of 1.0, while linear attention displays no discernible separator token-induced state modulation.

Notably, GLA exhibits a substantial variation in state modulation depending on the separator token, consistent with our earlier observations in Falcon Mamba, with regards to preemptive state modulation. In contrast, SA shows a smaller, yet non-trivial, effect. WLA and LA show no discernible differences across separator tokens, which may be attributed to their overall limited ability to modulate state size.

D.6 MISCELLANEOUS

D.6.1 EFFECTIVE STATE-SIZE ON C++ CODE

Beyond sentence delimiters such as periods and end-of-speech tokens (discussed in Section 4.4), we observe similar "dips" in effective state-size where there are scope delimiter tokens such as "}".

The following plots demonstrate the ESS pattern of Llama3-8B processing the C++ code of a fast inverse square root algorithm and a Fibonacci sequence generator algorithm.

Quake fast inverse square-root algorithm:

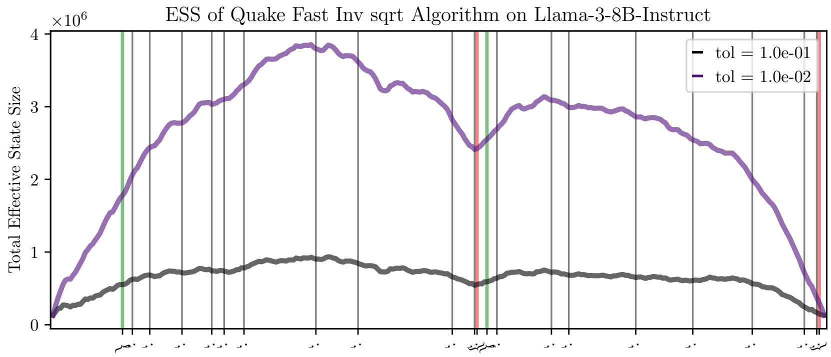


Figure 39: Effective state-size over a quake fast inverse square root algorithm's code.

```cpp
#include <iostream>
#include <cmath>

// Quake Fast Inverse Square Root function
float quakeFastInvSqrt(float number) {
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y = number;
    i = *(long*)&y;             // Bit-level hacking: convert float to long
    i = 0x5f3759df - (i >> 1);  // Initial magic number and bit shift
    y = *(float*)&i;            // Convert back from long to float

    // Newton's method step for refining the result
    y = y * (threehalfs - (x2 * y * y));  // First iteration

    return y;
}

int main() {
    float number;

    // Input: Get the number from the user
    std::cout << "Enter a number: ";
    std::cin >> number;

    // Output: Display the result using the Quake fast inverse sqrt
    float quake_result = quakeFastInvSqrt(number);
    std::cout << "Quake Fast Inverse Sqrt: " << quake_result << std::endl;

    // Compare with standard sqrt function
    float std_result = 1.0f / std::sqrt(number);
```

```
    std::cout << "Standard Inverse Sqrt: " << std_result << std::endl;

    return 0;
}
```

Fibonacci sequence generating algorithm:

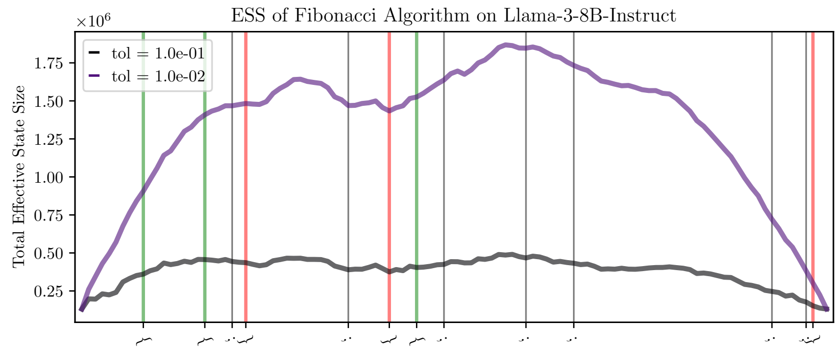


Figure 40: Effective state-size over a Fibonacci sequence generator algorithm's code.

```
#include <iostream>
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    }
    return fibonacci(n - 1) + fibonacci(n - 2);  // Recursive case
}
int main() {
    int n;
    std::cout << "Enter a positive integer: ";
    std::cin >> n;
    std::cout << "Fibonacci number at position " << n << " is: " <<
    fibonacci(n) << std::endl;
    return 0;
}
```

D.6.2 How the Number of Prompting Shots Affects the Effective State-Size of Language Models

Here, we explore how varying the number of shots when prompting large language models affects their effective state-size patterns. We use Phi-2 as the candidate attention model and Mamba-2.8B as the state-space model. The task we tested this on is MMLU (elementary mathematics).

At the start of the Q&A section for the attention model, there is a noticeable difference in state size between 0-shot and 1-shot prompts. Beyond 1-shot, the difference in ESS appears minimal. For the state-space model, varying the number of shots has minimal impact on the effective state-size.

Although these sparse experimental results require further investigation, we note the stark difference in the effective state-size patterns between these two architectures, which provides additional insights into understanding the fundamental differences in the way prompts are processed across models.

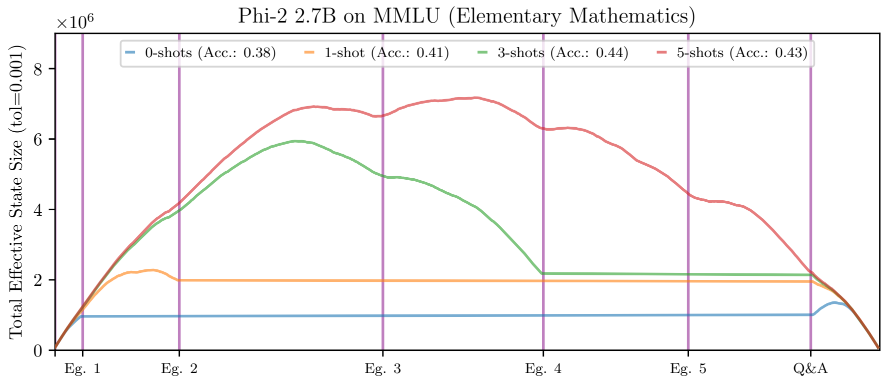


Figure 41: The variation in effective state-size with a varying number of shots (2.7B Attention).

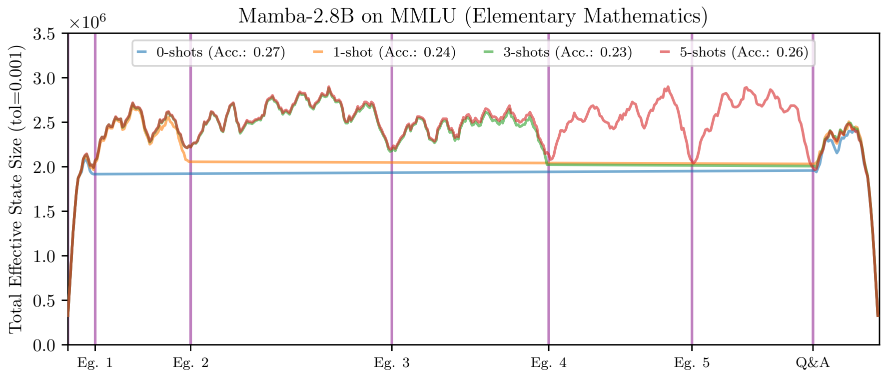


Figure 42: The variation in effective state-size with a varying number of shots (2.8B State-Space Model).